CHAPTER 1

Rewriting

Nachum Dershowitz
David A. Plaisted

SECOND READERS: Bernhard Gramlich, Mitch Harris, and Konstantin Korovin.

Contents

1 Introduction .................................................... 3
2 Terminology ................................................... 7
3 Normal Forms and Validity ................................... 10
4 Termination Properties ........................................ 11
5 Church-Rosser Properties ..................................... 25
6 Completion ..................................................... 33
7 Relativized Rewriting .......................................... 41
8 Equational Theorem Proving ................................ 47
9 Conditional Rewriting ......................................... 51
10 Programming .................................................. 59
   Bibliography .................................................. 63
   Index .......................................................... 74
1. Introduction

Equations lie at the foundation of mathematics and the sciences. Sometimes one needs to determine if an identity follows logically from axioms; other times, one seeks solutions to an equation; oftentimes one wishes to compute an equivalent, simpler form for a given expression. These equational reasoning abilities are indispensable in many computer applications, including symbolic algebraic computation, automated deduction, program specification and verification, and high-level programming languages and environments.

*Rewriting* is a very powerful method for dealing computationally with equations. Oriented equations, called *rewrite rules*, are used to replace equals by equals, but only in one direction. The theory of rewriting centers around the concept of *normal form*, an expression that cannot be rewritten any further. Computation consists of rewriting to a normal form; when the normal form is unique, it is taken as the value of the initial expression. When rewriting equal terms always leads to the same normal form, the set of rules is said to be *convergent* and rewriting can be used to decide validity of identities in the equational theory. Rewriting has the computational power of Markov algorithms—and of recursive functions and Turing machines. The basic ideas hark back to Axel Thue [1914].

Within automated theorem provers, derived equations can be used freely for *simplification* provided progress toward normal forms is guaranteed. This process is sometimes called *demodulation*. The trick is to ensure that one can delete pre-simplified formulæ without compromising completeness of the prover.

We begin with several motivating examples.

1.1. Example (*Hercules and Hydra*). Hydra is a bush-like creature with multiple heads attached by stems to the ends of branches. Branches have nodes with buds along their length, and may ramify at any node. Each time Hercules hacks off a head of Hydra’s, one of the nodes along the branch leading to that head sprouts some number of new branches identical to—and adjacent to—the weakened branch that previously supported the severed head, together with all its remaining heads and nodes. Furthermore, when any node loses all its branches and stems, its bud ripens into a new head. But when the root loses a stem with a head, nothing is regenerated.

Suppose Hydra starts off as a lone stalk with 10 buds along its length and one head at the apex. Chopping that head results in the original stalk topped by the stump of a head, which ripens, leaving a stalk of length 9 with 1 head. Then one of the buds along the stalk sprouts many copies of the branch above it, together with its fresh head. Chopping off one of these new heads again causes a bud below
the chopped head to regenerate a number of copies of the branch supporting the
severed head, together with its heads, buds, and branches.

If ever Hercules eliminates all of Hydra's heads, Hydra loses. The question is:
How can Hercules defeat Hydra?

Establishing termination of rewriting processes is the topic of Section 4.

1.2. Example (Grecian Urn). An urn holds 150 black beans and 75 white. Two
beans are removed at a time: if they're the same color, a black one is placed in the
urn; if they're different, the white one is returned. The process is repeated as long
as possible. This problem may be expressed as the following system of replacement
rules:

\[
\begin{align*}
BB & \rightarrow B \\
WW & \rightarrow B \\
WB & \rightarrow W \\
BW & \rightarrow W \\
WB & \rightarrow BW
\end{align*}
\]

the bottom two of which indicate that beans shift around in the urn. Is the color
of the last bean in the urn predetermined and, if so, what is it?

Determinism of output is covered in Section 5. How to handle permutative rules
such as the last two is the subject of Section 7. In Section 6, ordered rewriting, using
rewrite relations that are defined in terms of a set of unordered equations plus a
well-founded partial ordering, is considered.

1.3. Example (Chameleon Island). The chameleons on this strange island come
in three colors, red, yellow, and green, and meander about continuously. Whenever
two chameleons of differing colors confront each other, they both change to the
neutral third color. This setup can be expressed by six rules:

\[
\begin{align*}
RY & \rightarrow GG \\
GY & \rightarrow RR \\
RG & \rightarrow YY
\end{align*}
\]

Suppose there are initially 15 red chameleons, 14 yellow, and 13 green. Can their
haphazard meetings lead to a stable, monochromatic state?

Analyzing the interaction of rules is the topic of Section 6.

1.4. Example (Insertion Sort). As a more prosaic example of rewrite system, con­
ider the following program to rearrange a list of natural numbers in non-increasing
order by inserting elements one-by-one into position:

\[
\begin{align*}
\text{max}(0, x) & \rightarrow x \\
\text{max}(x, 0) & \rightarrow x \\
\text{min}(0, x) & \rightarrow 0 \\
\text{min}(x, 0) & \rightarrow 0 \\
\text{sort}(\epsilon) & \rightarrow \epsilon \\
\text{insert}(x, \epsilon) & \rightarrow x : \epsilon \\
\text{sort}(x : y) & \rightarrow \text{insert}(x, \text{sort}(y)) \\
\text{insert}(x, y : z) & \rightarrow \text{max}(x, y) : \text{insert}(\text{min}(x, y), z))
\end{align*}
\]
Lists are represented as nested pairs [e.g. \((3, 1, 4, 1)\) as \(3 : (1 : (4 : (1 : e)))\)] and numbers in tally (unary) notation [e.g. \(4\) is short for \(s(s(s(0)))\)]. Computation proceeds by computing the normal form of an input term of the form \(\text{sort}(3 : 1 : 4 : 1 : e)\). We would like to ascertain that every well-sorted ground term constructed from \(\text{sort}, \cdot, e, s,\) and \(0\) is equal to a unique term not containing \(\text{sort}\), nor the auxiliary symbols, \(\text{insert}, \text{max},\) and \(\text{min}\). In addition, one might wish to prove inductive properties of the program, such as \(\text{sort}(\text{sort}(\ell)) = \text{sort}(%\ell)\) for all lists \(\ell\) of natural numbers.

This is an example of an orthogonal rewrite system, defined in Section 5 and studied in Section 10. For computation, rewrite rules are usually applied nondeterministically, since, in general, more than one rule can be applied, and any one rule may apply at more than one position within a term. Regarding proofs of inductive properties, see [Comon 2001] (Chapter 14 of this Handbook).

1.5. **Example (Loops).** Consider the following dozen rules:

\[
\begin{align*}
x\backslash x & \rightarrow e \\
x/x & \rightarrow e \\
ce \cdot x & \rightarrow x \\
x \cdot e & \rightarrow x \\
e \backslash x & \rightarrow x \\
x/e & \rightarrow x
\end{align*}
\]

Each rule follows by algebraic manipulation from some combination of the following seven axioms for algebraic structures called loops, which are groups without associativity:

\[
\begin{align*}
x \cdot (x \backslash y) &= y \\
x \backslash (x \cdot y) &= y \\
x \backslash (x \cdot y) &= y \\
y/y &= x \backslash x
\end{align*}
\]

along with the defining equation

\(y/y = e\)

Rewrite systems were designed to be used as decision procedures for truth of an equation in all models of the theory. To decide whether an arbitrary equation is a valid identity for all loops—in which case it can be proved by purely equational reasoning, we need to ascertain that any two terms equal in the theory have the same normal forms. Section 3 deals with deciding validity by rewriting. Constructing such systems is the subject of Section 6.

1.6. **Example (Interpreter).** Rewrite systems can be used to interpret other programming languages. The machine state of a Turing-equivalent two-counter device
Thue systems, or string-rewriting systems, are a related formalism of equivalent computational strength. They originated historically in an attempt to investigate computability and can be simulated by rewrite systems in which every function symbol is monadic (has arity 1). Such systems have an implicit variable at the right end; for example, \( f(g(f(f(g(x)))) \)), though written as \( fggf \), rewrites any suffix \( fggf \) having prefix \( fgf \). For such systems, rewriting corresponds to substring replacement. Another way of viewing such systems is as presentations of monoids, where

\[
\begin{align*}
eval(zap_0, (x, y)) & \rightarrow (0, y) & \eval(zap_1, (x, y)) & \rightarrow (x, 0) \\
eval(incl_0, (x, y)) & \rightarrow (s(x), y) & \eval(incl_1, (x, y)) & \rightarrow (x, s(y)) \\
eval(dec_0, (0, y)) & \rightarrow (0, y) & \eval(dec_1, (x, 0)) & \rightarrow (x, 0) \\
eval(ifpos_0, (0, y)) & \rightarrow (0, y) & \eval(ifpos_1, (p, x, 0)) & \rightarrow (x, 0) \\
eval(ifpos_0, (s(x), y)) & \rightarrow \eval(p, (s(x), y)) & \eval(ifpos_1, (x, s(y))) & \rightarrow \eval(p, (x, s(y))) \\
whilepos_0 p & \rightarrow (ifpos_0 p; whilepos_0 p) & whilepos_1 p & \rightarrow (ifpos_1 p; whilepos_1 p) \\
\eval((p; q), u) & \rightarrow \eval(q, \eval(p, u)) \\
\end{align*}
\]

Figure 1: Two-counter machine interpreter.

can be represented as a pair \((x, y)\). The semantics of its instruction set can be defined by the rules for an interpreter, shown in Fig. 1. To simulate a machine computation, one rewrites a term of the form

\[
\eval\text{program, } \langle\text{input, 0}\rangle
\]

until a normal form is reached. The penultimate rule, for example, can clearly be applied \textit{ad infinitum}. A specific strategy of rule application is needed to guarantee that a normal form is attained whenever one exists.

Programming issues are discussed in Section 10.

1.7. Example \((Stack)\). In some cases, it is convenient to attach conditions to rules. For example, stack operations can be implemented as follows:

\[
\begin{align*}
top(push(x, y)) & \rightarrow x & pop(push(x, y)) & \rightarrow y \\
empty?(x) & \rightarrow T & empty?(push(x, y)) & \rightarrow F \\
empty?(x) = F & | & push(top(x), pop(x)) & \rightarrow x
\end{align*}
\]

The condition of the last rule ensures that only nonempty stacks are popped. Conditional rewriting and related inference techniques are the subject of Chapter 9.

Semi-Thue systems, or string-rewriting systems, are a related formalism of equivalent computational strength. They originated historically in an attempt to investigate computability and can be simulated by rewrite systems in which every function symbol is monadic (has arity 1). Such systems have an implicit variable at the right end; for example, \( f(g(f(f(g(x)))) \)), though written as \( fggf \), rewrites any suffix \( fggf \) having prefix \( fgf \). For such systems, rewriting corresponds to substring replacement. Another way of viewing such systems is as presentations of monoids, where
there are a finite number of individual constants and an associative concatenation operator. Semi-Thue systems will not be further considered here. The monographs [Benninghofen, Kemmerich and Richter 1987, Book and Otto 1993] describe methods and results for strings.

Several other important topics in rewriting are only touched upon (modularity, for example), or not covered at all (higher-order rewriting and “sequentiality”, for example), in this chapter, which concentrates on the use of rewriting techniques in the context of automated deduction. For some other surveys of rewriting, see [Dershowitz and Jouannaud 1990, Klop 1992, Plaisted 1993, Baader and Nipkow 1998].

2. Terminology

The central idea of rewriting is to impose directionality on the use of equations in proofs. A rewrite rule is an ordered pair, written \( l \rightarrow r \), of terms \( l \) and \( r \). Like equations, rules are used to replace instances of \( l \) by corresponding instances of \( r \); unlike equations, rules are not used to replace instances of the right-hand side \( r \).

A (free, first-order) term over symbols \( F \), constants \( C \), and variables \( X \) is either a variable \( x \in X \), an individual constant \( c \in C \), or an expression of the form \( f(t_1, t_2, \ldots, t_n) \), where \( f \in F \) is a function symbol of arity (number of arguments) \( n \) and the \( t_i \) are terms. Each function symbol has one or more arities, which are nonnegative integers indicating how many arguments it can take. For example, \( f(a, g(x, y)) \) is a term, with outermost function symbol \( f \).

Terms can be seen as labeled rooted ordered trees. Let the set of all such terms be denoted \( T \). A term \( t \) is linear if each variable appears at most once in \( t \). Thus the term \( f(x, x, z) \) is not linear. A ground term is a term containing no variables. To avoid confusion, we will use the symbol \( \equiv \) for syntactic identity of terms.

A (strict) partial ordering \( > \) is an irreflexive transitive relation. A quasi-ordering \( \succeq \) is a reflexive transitive relation. We try to use the sign \( > \) for partial orderings and \( \succeq \) for quasi-orderings, where convenient. As usual, \( y < x \) means the same as \( x > y \), \( x \geq y \) means either \( x > y \) or \( x = y \), and \( x \succ y \) means \( x \succ y \) but not \( y \succeq x \).

We say that a term \( u \) is a subterm of \( t \) and write \( t \triangleright_{\text{sub}} u \) if either \( t \equiv u \) or if \( t \equiv f(t_1, \ldots, t_n) \) and \( u \) is a subterm of \( t_i \) for some \( i \). We write \( t \triangleright_{\text{sub}} u \) to indicate that \( u \) is a proper subterm of \( t \). When needed, it is customary to specify positions of subterms as sequences of integers indicating the path to the subterm in the tree representation of the term. If \( \alpha \) is a position and \( t \) is a term, we would write \( t|_{\alpha} \) for the subterm of \( t \) at position \( \alpha \), defined by \( t|_{\epsilon} \equiv t \) for the empty sequence (top position) \( \epsilon \); otherwise, \( f(t_1, \ldots, t_n)|_{i\alpha} \equiv t_i|_{\alpha} \). A context is a term with some subterms replaced by holes, here signified by \( \cdot \). A lone \( \cdot \) is thus the simplest context. If \( t \) is a context with \( m \) holes (an \( m \)-context) and \( s_1, \ldots, s_m \)

---

1. In this chapter, we consider only unsorted terms, for simplicity.
are terms, then \( l[s_1, \ldots, s_m] \) indicates \( t \) with the \( o \)'s replaced by \( s_1, \ldots, s_m \) in a predetermined order. At the same time, \( l[s] \) indicates that the term \( t \) contains an occurrence of the subterm \( s \). Two occurrences of subterms are disjoint if neither is a subterm of the other.

A substitution is a mapping from variables to terms (or contexts) denoted by \( \{x_1 \mapsto s_1, x_2 \mapsto s_2, \ldots, x_n \mapsto s_n \} \) indicating that the variable \( x_i \) maps to the term \( s_i \). (The domain of a substitution is normally a finite set of variables.) If \( t \) is a term or a context and \( \sigma \) is a substitution, we use postfix notation \( t\sigma \) for the image of \( t \) under \( \sigma \). This is defined as follows: If \( \sigma x_i \mapsto s_i \), then \( x\sigma \equiv s_i \); if \( x \) is a variable not in the domain of \( \sigma \), then \( x\sigma \equiv x \); if \( f(t_1, \ldots, t_k) \) is a term then \( f(t_1, \ldots, t_k)\sigma \equiv f(t_1\sigma, \ldots, t_k\sigma) \). Also, \( \sigma \equiv \circ \). The image of an equation, rewrite rule, or formula under a substitution is defined similarly. If \( t \) is a term, \( t\sigma \) is called an instance of \( t \). Note that \( l[u]\sigma \equiv (l\sigma)[u\sigma] \). 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 \( \sigma \) a ground substitution for \( t \) and \( t\sigma \), a ground instance of \( t \). Similar terminology applies to instances of equations, rewrite rules, and formulas.

A term-rewriting or rewrite system \( R \) is a set of rewrite rules, \( l \to r \), where \( l \) and \( r \) are each terms, both of which may contain variables which refer to arbitrary terms. A rewrite system \( R \) defines a rewrite relation \( \to_R \) (or just \( \to \)) on terms, which is the smallest relation such that for all contexts \( t[\sigma] \), rules \( l \to r \in R \), and substitutions \( \sigma \), \( t[\sigma] \to_R t[r\sigma] \). That is, if \( t[\sigma] \) has a subterm \( l\sigma \) which is an instance of the left-hand side of a rule \( l \to r \), then the \( R \)-redex \( l\sigma \) in \( t \) is contracted by replacing that subterm with the corresponding instance \( r\sigma \) of the right-hand side of the rule, thereby rewriting \( t[\sigma] \) to \( t[r\sigma] \). A redex \( u \) is innermost for \( R \) if \( u \) is a \( R \)-redex but no proper subterms of \( u \) are \( R \)-redexes. An occurrence of a redex \( u \) in a term \( t \) is outermost for \( R \) if the occurrence of \( u \) is not a proper subterm of any other \( R \)-redexes of \( t \). We often use \( R \) as an abbreviation for the binary relation \( \to_R \). A rewrite rule \( l \to r \) is left-linear if \( l \) is linear, and right-linear if \( r \) is. A rewrite system is left- (right-) linear if all its rules are. If function symbol \( f \) is not the outermost function symbol of the left-hand side of any rule in \( R \), \( f \) is called a constructor for \( R \).

As is commonplace, the reflexive-transitive closure of a binary relation \( \to \) is indicated by \( \to^* \), its transitive closure is indicated by \( \to^+ \), and its reflexive closure by \( \to^- \). We write \( s \leftarrow r \) if \( r \to s \) and \( r \leftarrow s \) if either \( r \to s \) or \( r \leftarrow s \). Composition of two relations is indicated by \( \circ \). Thus \( r \leftarrow t \circ \to^* s \) if there is an element \( t \) such that \( r \leftarrow t \) and \( t \to^* s \).

An equation is a formula of the form \( r = s \) where \( r \) and \( s \) are terms. An equation system is any set of equations. If \( E \) is a set of equations let \( E_\to \) be \( \{r \to s : r = s \in E\} \). We write \( r \leftrightarrow_E s \) if \( s \to_E r \) or \( r \to_E s \) where \( u \to_E v \) is defined as \( u \to_E v \). We may write \( \leftrightarrow_E \) when we actually mean \( \leftrightarrow_E \), viewing the equations as unordered pairs—as long as no confusion is likely to arise. If \( E \) is a set of equations and \( > \) is an ordering on terms, we define the ordered rewriting relation of \( E \) as the pairs of terms \( \{(r, s) : r \leftrightarrow_E s \text{ and } r > s\} \). This relation may also be simply denoted by \( \to_E \) when the ordering in question is understood.

A derivation for a binary relation \( \to \) (for instance, a rewrite relation \( \to_R \)) is a
sequence of the form \( t_0 \rightarrow t_1 \rightarrow t_2 \ldots \). An element \( t \) is reducible (with respect to a given binary relation \( \rightarrow \)) if there is an element \( u \) such that \( t \rightarrow u \); otherwise, \( t \) is irreducible. We say that \( u \) is an \( \rightarrow \)-normal form of \( t \) if \( t \rightarrow^* u \) and \( u \) is irreducible via \( \rightarrow \). We write \( t \rightarrow^* u \) if \( t \rightarrow^* u \) and \( u \) is a normal form. We also write \( R \)-reducible instead of \( \rightarrow_R \)-reducible and \( R \)-normal form instead of \( \rightarrow_R \)-normal form, etcetera. The normalizability relation \( \rightarrow^! \) defines a partial function when normal forms are unique and a total function when it is always uniquely normalizing.

A binary relation \( \rightarrow \) is terminating (or strongly normalizing) if there are no infinite derivations \( t_0 \rightarrow t_1 \rightarrow t_2 \ldots \). It is terminating for a set \( T \) of elements if there are no infinite derivations with \( t_0 \in T \). A relation \( \rightarrow \) is confluent if there is an element \( v \) such that \( s \rightarrow^* v \) and \( t \rightarrow^* v \) whenever \( u \rightarrow^* s \) and \( u \rightarrow^* t \) for some elements \( s, t \), and \( u \). Confluence is equivalent to the Church-Rosser property that \( s \rightarrow^* v \) and \( t \rightarrow^* v \) for some \( v \) whenever \( s \leftrightarrow^* t \). By extension, the terms “terminating” and “confluent” are also applied to rewrite systems \( R \) whose rewrite relation \( \rightarrow_R \) has those qualities. We say that a relation \( \rightarrow \), or rewrite system \( R \), is convergent (or complete) if it is terminating and confluent.\(^2\) Convergent rewrite system are especially interesting, because all derivations lead to a unique normal form; such systems are used to decide equational theories (which describe varieties).

Suppose \( M \) is a first-order structure, with domain \( D \), assigning meaning (semantics) to each individual constant, function symbol, and variable in the vocabulary. The meaning \([t]_M \) of a term \( t \) is an element of \( D \) and is defined inductively: \([x]_M = x_M \) for variable \( x \) with meaning \( x_M \); \([f(t_1, \ldots, t_n)]_M = f_M([t_1]_M, \ldots, [t_n]_M)\), if the meaning of \( f \) with arity \( n \) is \( f_M \). If \( M_1 \) and \( M_2 \) are two structures and \( X \) is a set of variables, then \( M_1 \equiv M_2 \) (mod \( X \)) if \( M_1 \) and \( M_2 \) agree on all function and constant symbols and on all variables not in \( X \). If \( s \equiv t \) is an equation and all variables in \( s \) or \( t \) appear in a set \( X \) of variables, and \( M \) is a structure, then \( M \) satisfies \( s \equiv t \), written \( M \models s \equiv t \), if for all structures \( M' \) such that \( M \equiv M' \) (mod \( X \)), \([s]_{M'} = [t]_{M'}\). One says that equation \( s \equiv t \) is valid in \( M \), in this case. This corresponds to the intention that variables are implicitly universally quantified. A structure \( M \) is a model of \( E \), written \( M \models E \), if \( M \) satisfies each element of \( E \). If \( E \) and \( E' \) are two equational systems, we write \( E \models E' \) if all structures \( M \) that satisfy \( E \) also satisfy \( E' \), that is, all models of \( E \) are also models of \( E' \). In this case, if \( E \models E' \), we call \( E' \) a logical consequence of \( E \). The equational theory of \( E \) is the set of equations that are logical consequences of \( E \).

Define \( R_\equiv \) to be the set of equations \( l = r \) for each rule \( l \rightarrow r \) in a system \( R \). When \( R \) is convergent, \( R_\equiv \models s = t \) (\( R_\equiv \) logically implies \( s = t \)) if \( s \) and \( t \) have the same normal form with respect to \( R \)-rewriting. Thus we can use \( R \) for theorem proving in the equational theory \( R_\equiv \). If \( R \) is not convergent, we may want to complete it, that is, find another system \( S \) such that their theories are the same but \( S \) is convergent; then \( S \) may be used to decide the equational theory \( R_\equiv \) of \( R \).

General rewriting logics have been devised [Goguen, Kirchner and Meseguer 1987, Cirstea and Kirchner 1999]. Such logics can be used to express many other logics.

\(^2\)The term canonical is sometimes used as a synonym for “convergent”, but will not be so used here.
in terms of the rewriting relation of a rewrite system.

3. Normal Forms and Validity

Rewriting methods have turned out to be among the more successful approaches to equational theorem proving. One main question of interest is to determine when $E \models s = t$ for various equation sets $E$ and equations $s = t$. Intuitively, this means that the equation is necessarily true whenever all the equations in $E$ are true. In order to answer this question, many proof systems have been developed, and recently many have been adapted for computer implementation. Rewriting can be used for this purpose:

3.1. Theorem. Suppose $E$ is an equational system and $\rightarrow$ is a binary relation. Then the equational theory $E$ is decidable if the following conditions are satisfied:
1. If $s \rightarrow t$ then $E \models s = t$.
2. It is decidable whether a term is $\rightarrow$-reducible.
3. If $r$ is $\rightarrow$-reducible, then one can compute a term $s$ such that $r \rightarrow^* s$.
4. The relation $\rightarrow$ is terminating.
5. If $r$ and $s$ are irreducible for $\rightarrow$, then it is decidable whether $E \models r = s$.

When the conditions are satisfied, one can reduce two given terms $r$ and $s$ to normal form using $\rightarrow$ and test if they are $E$-equivalent. Presumably, testing if $r$ and $s$ are $E$-equivalent is easier when they are both irreducible. This theorem is used implicitly in the discussion of relativized (equational) rewriting in Section 7.

Rewrite systems are used in this way to check for validity. The simplest application of the above theorem is in the case when there is exactly one irreducible term in each $E$-equivalence class. Hence, one of the most essential properties a rewrite system can enjoy is unique normalization. In particular, convergent systems compute unique normal forms and serve as decision procedures for validity in the equational theory of $E$. For a convergent system $R$ to determine provability in its underlying equational theory $R_m$, the test for reducibility must be recursive. Then, to decide if $s \leftrightarrow_R^* t$, one can check if the $R$-normal forms of $s$ and $t$ are identical.

A rewrite ("valley") proof of an equation $s = t$ for rewrite system $R$ is a sequence $s_1, s_2, \ldots, s_n, t_m, t_{m-1}, \ldots, t_1$ where $s_1 \equiv s$ and $t_1 \equiv t$ and $s_n \equiv t_m$ and for all $i$, $1 \leq i < n$, $s_i \rightarrow_R s_{i+1}$ and for all $i$, $1 \leq i < m$, $t_i \rightarrow_R t_{i+1}$. This establishes the relation $s \rightarrow^* o \leftarrow^* t$ between $s$ and $t$, indicating that the same term can reached by rewriting $s$ and $t$ some number of times. The Church-Rosser property means that $\leftrightarrow^*$, as a relation, is equal to $\rightarrow^* o \leftarrow^*$; termination ensures that $\rightarrow^* o \leftarrow^* = \rightarrow' o \leftarrow'$; Church-Rosser implies that $\rightarrow'$ defines a function; recursiveness of reducibility makes that function computable. Though the Church-Rosser property is undecidable, Knuth and Bendix [1970] devised an effective superposition test, based on "critical overlaps", to decide whether a terminating system is convergent. But termination itself is undecidable. Termination is further discussed in Section 4 and the Church-Rosser property in Section 5.
The (ground) word problem is to decide the truth of ground equations in equational theories (whose classes of models are called varieties); the uniform word problem is to decide validity of (universally quantified) equations. Of course, not all word problems can be solved by rewriting: some theories are not finitely based, and some finitely-based equational theories are undecidable.

Many rewrite-system decision procedures are known; perhaps the first such procedure for a word problem was Trevor Evans’ for loops [Evans 1951]; see Example 1.5. In their seminal paper, Knuth and Bendix [1970] (building on the work of Evans) demonstrated how failure of the superposition test suggests additional rules that can be used to help complete a nonconvergent system. Completion utilizes an ordering on terms to provide guidance in the generation of new rules and to direct the simplification of equations. Ordered completion, first suggested in [Brown 1975, Lankford 1975], and later developed further in [Hsiang and Rusinowitch 1987, Bachmair, Dershowitz and Plaisted 1989, Bachmair and Dershowitz 1994], is a powerful extension of Knuth’s method. Ordered completion permits unorientable equations to be used together with rewrite rules in the completion process.

Under weaker conditions, one can still use rewrite systems for theorem proving. A binary relation $R$ is normalizing (or weakly terminating) if every term has an $R$-normal form, that is, for all $t$ there is an $s$ such that $r \vdash_R s$. A binary relation is uniquely normalizing if every term has exactly one normal form, though it need not necessarily be terminating. If a rewrite system $R$ is finite and uniquely normalizing, one can test whether $R \vdash s = t$ by enumerating (in some breadth-first fashion) all derivations from $s$ and $t$ until their normal forms $u$ and $v$ are found, that is, $s \rightarrow^*_R u$ and $t \rightarrow^*_R v$. Then $R \vdash s = t$ iff $u$ and $v$ are identical. But this approach is often impractical, and we do not pursue it further.

In addition to its use for generating convergent systems to serve as decision procedures for the given axioms, ordered completion may also be used as an equational theorem prover. Classical forward reasoning systems work from the axioms, “expanding” the set of established formulæ by inferring new ones from old ones. Completion may be viewed as an inference engine that also “contracts” formulæ by constantly rewriting them, making forward reasoning practical. The potentially infinite set of rules and equations generated by completion are used to simplify the two sides of the equation in question. Even if a convergent system is not generated, it may be possible to generate enough rewrite rules to prove the theorem in question. Rules are used in the direction of the arrow only, while equations are used in whichever direction reduces the term it is applied to in the given ordering. An identity is proved when both sides reduce via rules and equations to the identical term. This is the subject of Section 8.

4. Termination Properties

One of the most important properties of a rewrite system, especially in the context of automated deduction, is termination.
The rules for the Chameleon Puzzle (1.3) are not terminating; to wit $R \ Y \ Y \rightarrow \ G \ G \ Y \rightarrow \ G \ R \ R \rightarrow \ Y \ Y \ R$, which rearranges to $R \ Y \ Y$, from which point the same three steps may be repeated over and over again. On the other hand, each step in the Grecian Urn Puzzle (1.2) decreases the number of beans, except for the last two which shift beans around; so it always terminates if there is no infinite cycle of shifts. The Loop system also terminates, since it always shortens the length of the expression. Our Interpreter (1.6) is nonterminating.

The standard fundamental tool for termination proofs is the well-founded partial ordering. A partial ordering $\succ$ of a set $W$ is well-founded if there are no infinite descending chains $x_0 \succ x_1 \succ x_2 \succ \cdots$ of elements $x_i \in W$. A total well-founded ordering is a well-ordering. (As a consequence of Zorn’s Lemma, every well-founded ordering is contained—as a set of pairs—in some well-ordering.) To make convenient use of well-founded orderings, we need several additional properties.

4.1. DEFINITION (Monotonicity). A binary relation $\rightarrow$ on a set $T$ of terms satisfies the **monotonicity** property if $s \rightarrow t$ implies $u[s] \rightarrow u[t]$ for all terms $s$, $t$ in $T$ and all contexts $u$ over $T$.

We will use the term “monotonic” in this sense only.

4.2. DEFINITION (Stable Extension). The **stable extension** of a binary relation $\rightarrow$ on ground terms (over some vocabulary) to a relation on terms $T$ containing variables is defined so that $s \rightarrow t$ iff $s\sigma \rightarrow t\sigma$ for all $s$, $t \in T$ and all ground substitutions $\sigma$.

The stable extension of a well-founded ordering is also well-founded. In practice, this extension is approximated in implementations by weaker orderings on free terms.

Stable extensions enjoy the following slightly more general property:

4.3. DEFINITION (Full Invariance). A binary relation $\rightarrow$ on a set $T$ of free terms satisfies the **full invariance** property if for all terms $s$, $t \in T$ and all substitutions $\sigma$ over $T$, $s \rightarrow t$ implies $s\sigma \rightarrow t\sigma$.

Length-based comparison of terms is a monotonic but not fully invariant ordering.

The rewrite step $\rightarrow_R$ is by definition monotonic and fully invariant. So we are led to define:

4.4. DEFINITION (Rewrite Relation). A binary relation on terms is a **rewrite relation** if it is monotonic and fully invariant.

4.5. DEFINITION (Reduction Ordering). A well-founded partial ordering on terms is a **reduction ordering** if it is monotonic and fully invariant.
If $s > t$ under a reduction ordering $>$, then all variables in $t$ must appear also in $s$. For example, if $f(x) > g(x, y)$, then by definition we also have $f(x) > g(x, f(x))$, and by monotonicity $g(x, f(x)) > g(x, g(x, f(x)))$, etc., giving an infinite descending sequence of terms. It follows that:

4.6. Theorem ([Lankford 1977]). A rewrite system $R$ is terminating if for some reduction ordering $>$, $l > r$ for all rules $l \rightarrow r \in R$ (symbolically: $R \subseteq >$).

The Loop system can be proved terminating using the fully-invariant subterm ordering $>_\text{sub}$ defined by $r >_\text{sub} s$ if $s$ is a proper subterm of $r$.

In fact, whenever (finite or infinite) $R$ is terminating, there is such a reduction ordering, viz. the derivability relation $\rightarrow_R^\#$. Thus, reduction orderings are (in theory) necessary and sufficient to prove termination of any terminating system $R$. But to be useful, reduction orderings for proving termination should be computable, or at least given to computable approximation, so that one can test whether $l > r$ for each rule.

If $R$ is a finite rewrite system over a set of terms $\mathcal{T}$ and $t$ is a term in $\mathcal{T}$, define $R^\#(t)$ to be the length of the longest derivation beginning with $t$, and $\infty$ if there is an infinite derivation beginning with $t$. Now, $R$ is terminating if for all $t$ in $\mathcal{T}$, $R^\#(t)$ is finite. When $R^\#(t)$ is finite, the set of terms $u$ such that $t \rightarrow_R^* u$ is finite, by König’s Lemma.

4.7. Theorem ([Huet and Lankford 1978]). The following problem is semi-decidable but not decidable: Given a finite rewrite system $R$ and a term $t$, does $R$ terminate for $t$?

The construction is standard [Yasuhara 1971].

In general, termination of even one rule systems is undecidable [Dauchet 1992]. For systems in which right-hand sides are ground, termination is decidable [Huet and Lankford 1978, Dershowitz 1981]. The concept of fair termination was defined in [Porat and Francez 1985]; the idea is that one only considers derivations in which no redex remains forever in the derivation without being contracted. Fair termination is decidable for ground systems [Tison 1989].

Since termination of all derivations initiated by a given term is undecidable, and termination for all terms is not even partially decidable, all one can hope for is practically useful termination tools.

Quasi-orderings are often more convenient than partial orderings for termination arguments. We will say that a quasi-ordering is well-founded, or is a reduction ordering, whenever its strict part is.

4.8. Definition (Sequence Ordering). If $\preceq_i$ are quasi-orderings of sets of elements $S_i$, then the lexicographic extension $\preceq_{\text{lex}}$ of the $\preceq_i$ to arbitrary-length sequences in
\[ \bigcup_n (S_1 \times S_2 \times \cdots \times S_n) \text{ is defined as follows:}^3 \]

\[ \langle s_1, \ldots, s_n \rangle \preceq_{\text{lex}} \langle t_1, \ldots, t_n \rangle \]

\[ \langle s_1, \ldots, s_{m+1} \rangle \succ_{\text{lex}} \langle t_1, \ldots, t_{n+1} \rangle \]

\[ \langle s_1, \ldots, s_{n+1} \rangle \preceq_{\text{lex}} \langle t_1, \ldots, t_{n+1} \rangle \]

\[ \langle s_1, \ldots, s_m \rangle \preceq_{\text{lex}} \langle t_1, \ldots, t_m \rangle \]

\[ \langle s_1, \ldots, s_{m+1} \rangle \succ_{\text{lex}} \langle t_1, \ldots, t_{m+1} \rangle \]

where \( \preceq_{\text{lex}} \) is the union of the equivalence relation \( \preceq_{\text{lex}} \) and the partial ordering \( \succ_{\text{lex}} \).

It is not hard to see that the relation \( \preceq_{\text{lex}} \) is reflexive and transitive. When each of the element orderings is well-founded, the sequence ordering is also well-founded for bounded-length tuples (only).

Bags (a.k.a. multisets) are unordered collections of elements, in which multiplicity of elements matters, and for which we will use square brackets. Informally, a bag (e.g. \([2, 1, 1, 2, 3]\)) is a set in which an element can occur more than once. Formally, a bag \( B \) is a function from some underlying domain to the nonnegative integers. Thus, \( x \in B \) if \( B(x) > 0 \). A bag is finite if \( \{ x : B(x) > 0 \} \) is finite.

4.9. **Definition** (Bag Ordering [Dershowitz and Manna 1979]). The bag (multi-set) extension \( \succ_{\text{bag}} \) of a quasi-ordering \( \succ \) of elements is defined as follows:

\[ \bigcup \left[ s_1, s_2 \ldots, s_n \right] \preceq_{\text{bag}} \left[ t_1, \ldots, t_n \right] \text{ if } s \sim t \]

\[ \left[ s_1, \ldots, s_n, s \right] \preceq_{\text{bag}} \left[ t_1, \ldots, t_n, t \right] \]

\[ \left[ s_1, \ldots, s_m \right] \succ_{\text{bag}} \left[ t_1, \ldots, t_n \right] \text{ if } s \succ u_1, \ldots, u_k \]

\[ \left[ s_1, \ldots, s_m, s \right] \succ_{\text{bag}} \left[ t_1, \ldots, t_n, u_1, \ldots, u_k \right] \]

where \( s \succ u_1, \ldots, u_k \) (for \( k \geq 0 \)) means that \( s \) is greater than each of the \( u_i \) and \( \preceq_{\text{bag}} \) is the union of the equivalence relation \( \preceq_{\text{bag}} \) and the partial ordering \( \succ_{\text{bag}} \).

The idea of this ordering is that a bag becomes smaller if an element is replaced by any number of smaller elements. Thus \([3, 4, 4] \succ_{\text{bag}} [2, 2, 1, 2, 4, 4]\), since 3 has been replaced by a single 1 and four 2's. This operation can be repeated any number of times, still yielding a smaller bag; in fact, the relation \( \succ_{\text{bag}} \) can be defined in this way as the smallest transitive relation having this property [Dershowitz 1987]. This relation can be computed reasonably quickly. It can be shown that the relation \( \succ_{\text{bag}} \) is transitive [Dershowitz and Manna 1979].

For a totally ordered underlying set, the elements of bags may be sorted in non-ascending order, then compared lexicographically; thus, the bag of ordinals

---

\(^3\)An inference rule of the form \( A_1 A_2 \cdots A_m \) indicates that if \( A_1, A_2, \ldots, A_n \) have been derived, then we may also derive \( A \) by using this rule. Such rules can also be used in a backward direction, meaning that if we are trying to derive \( A \), or an instance of \( A \), we attempt to derive all of \( A_1, A_2, \ldots, A_n \) (or their instances).
[\alpha_0, \ldots, \alpha_n] is of order type \( \sum \omega^{\alpha_i} \), summed naturally [Dershowitz and Manna 1979]. The bag ordering extends this effect to element orderings that are partial. For comparisons with alternative orderings on bags, see [Jouannaud and Lescanne 1982, Martin 1989].

4.10. THEOREM ([Dershowitz and Manna 1979]). The bag (multiset) ordering is well-founded (for finite bags) iff the element ordering is.

4.11. DEFINITION (Simplification Ordering [Dershowitz 1982]). A monotonic quasi-ordering \( \succeq \) on terms is a (quasi-) simplification ordering for every function symbol \( f \) and index \( i \),

\[ f(\ldots, x_i, \ldots) \succeq x_i \]

If \( f \) is varyadic, we also require

\[ f(\ldots, x_i, \ldots) \succeq f(\ldots, x_{i-1}, x_{i+1}, \ldots) \]

A monotonic partial ordering \( > \) on terms is a strict simplification ordering if for every function symbol \( f \) and index \( i \),

\[ f(\ldots, x_i, \ldots) > x_i \]

If \( f \) is varyadic, we also require

\[ f(\ldots, x_i, \ldots) > f(\ldots, x_{i-1}, x_{i+1}, \ldots) \]

For fixed-arity symbols, if a partial ordering is a rewrite relation and includes the subterm relation \( >_{\text{sub}} \), then it is a fully-invariant simplification ordering. Simplification orderings are like the “divisibility order” of [Higman 1952], but apply to varyadic (variable arity) function symbols. We will say that a simplification ordering is well-founded if its strict counterpart is.

4.12. DEFINITION (Homeomorphic Embedding). The homeomorphic embedding relation \( \succeq_{\text{emb}} \) on a set \( T \) of fixed-arity terms is the derivability relation of the rewrite rules

\[ f(x_1, \ldots, x_i, \ldots, x_n) \rightarrow x_i \]

where \( x_1, \ldots, x_n \) are distinct variables, for every \( f \) in the vocabulary and for each \( i \). For variable-arity symbols \( f \), we also have rules

\[ f(\ldots, x_i, \ldots) \rightarrow f(\ldots, x_{i-1}, x_{i+1}, \ldots) \]

If \( t \preceq_{\text{emb}} s \), that is, if \( s \rightarrow^* t \) in this system, we say that \( t \) is embedded in \( s \). If \( t \) is embedded in \( s \) but they are not identical, then we write \( t <_{\text{emb}} s \). Homeomorphic embedding is a monotonic quasi-ordering.

Simplification orderings \( \succeq \) contain the homeomorphic embedding relation \( \succeq_{\text{emb}} \). Strict simplification orderings \( > \) contain the strict embedding \( >_{\text{sub}} \).
4.13. **Theorem** ([Dershowitz 1982]). For terms over a finite set of function symbols and constants, any (quasi-) simplification ordering is well-founded.

This follows directly from Kruskal’s Tree Theorem:

4.14. **Theorem** (Tree Theorem [Kruskal 1960]). Any infinite sequence \( t_1, t_2, \ldots \) of terms in a set \( T \) with finitely many function symbols and constants contains two terms \( t_j \) and \( t_k \) (\( j < k \)) such that \( t_j \preceq_{\text{emb}} t_k \).

Kruskal extended Higman’s work [Higman 1952] to varyadic function symbols; for fixed-arity terms, this is “Higman’s Lemma”.4

Since fully-invariant strict simplification orderings (for finite vocabularies) are reduction orderings, they can be used to show termination, as per Theorem 4.6.

4.15. **Definition** (Self-Embedding). A derivation \( t_1 \rightarrow_R t_2 \rightarrow_R \cdots \rightarrow_R t_j \rightarrow_R \cdots \rightarrow_R t_k \rightarrow_R \cdots \) of terms is self-embedding if \( t_j \preceq_{\text{emb}} t_k \) for some \( j < k \). A rewrite system is self-embedding if it allows a self-embedding derivation. A rewrite system \( R \) is self-embedding on \( t \) if there is a self-embedding \( R \)-derivation initiated by \( t \).

4.16. **Corollary** ([Dershowitz 1982]). If a finite rewrite system is nonterminating, then it must be self-embedding.

4.17. **Definition** (Simple Termination [Ferreira and Zantema 1993]). A rewrite system \( R \) is simply terminating if there is a fully-invariant strict simplification ordering \( \succ \) containing \( R \).

We have seen (Theorem 4.13) that a rewrite system is terminating in this case. There are, however, terminating systems that are not simply terminating, such as the lone rule \( f(f(x)) \rightarrow f(g(f(x))) \).

4.18. **Theorem** ([Dershowitz 1982]). Suppose \( R \) is a finite rewrite system and \( \succeq \) is a fully-invariant simplification ordering. If \( R \subset \succ \), then \( R \) is terminating.

**Proof.** Suppose \( s_1 \rightarrow_R s_2 \rightarrow_R \cdots s_n \rightarrow_R \cdots \). Then \( s_1 \succeq s_2 \succeq s_3 \cdots \) because the quasi-ordering \( \succeq \) is monotonic. Let \( B(s_i) \) be the bag of subterms of \( s_i \). One can show that \( B(s_1) \succ_{\text{bag}} B(s_2) \succ_{\text{bag}} B(s_3) \cdots \), since a subterm \( t \) of \( s_i \) has been replaced by a strictly smaller rewritten term \( t' \) to obtain \( s_{i+1} \), and all subterms of \( t' \) are smaller than or equivalent to \( s_i \) in the simplification ordering. Since \( R \) is finite, only finitely many function symbols appear in the derivation sequence. Since \( \succ \) is well-founded for terms over a finite vocabulary, \( \succ_{\text{bag}} \) is also. Therefore, the derivation must be finite and \( R \) terminates. 

---

4 Both Higman’s Lemma and Kruskal’s Theorem also apply to infinite vocabularies, with an underlying well-quasi-ordering of the symbols, extended to a homeomorphic embedding on terms. A beautiful, non-constructive proof may be found in [Nash-Williams 1963].
Instead of working directly with an ordering on terms, one can compare measures of terms in some well-founded ordering. For this purpose we have the following definition:

4.19. Definition (Termination Function). A (quasi-) termination function $\tau$ is a function from a set of terms $T$ to a set $W$, supplied with a well-founded (quasi-) ordering $\triangleright$ that has the monotonicity property, such that for all terms in $T$,

$$\tau(f(s_1, \ldots, s_n)) \triangleright \tau(s_1, \ldots, \tau(s_n))$$

For variable-arity symbols $f$, we also require

$$\tau(f(\ldots, x_i, \ldots)) \triangleright \tau(f(\ldots, x_{i-1}, x_{i+1}, \ldots))$$

4.20. Theorem. Suppose $R$ is a rewrite system over a set $T$ of terms and $\tau$ is a termination function, then $R$ is terminating if for all rules $r \rightarrow s$ in $R$ and for all substitutions $\sigma$ over $T$,

$$\tau(r\sigma) \triangleright \tau(s\sigma).$$

This is the method of [Manna and Ness 1970], adapted to quasi-orderings by analogy to Theorem 4.18.

Typically, termination functions $\tau : T \rightarrow W$ are defined as a set of homomorphisms $\tau_f : W^n \rightarrow W$, one for each $f \in \mathcal{F}$ of arity $n$:

$$\tau(f(s_1, \ldots, s_n)) = \tau_f(\tau(s_1), \ldots, \tau(s_n))$$

Each variable $x$ is mapped by $\tau$ to a distinct variable over $W$. When the termination function is defined by a convergent rewrite system, its necessary properties can be established using techniques similar to those of the next section; see [Bachmair and Dershowitz 1986, Bellegarde and Lescanne 1990].

As examples of termination functions, let $W$ be the natural numbers and obtain $\tau(t)$ equal to the size (number of symbols) of $t$ by defining $\tau_f(x_1 \ldots x_n) = 1 + x_1 + \ldots + x_n$ for all symbols $f \in \mathcal{F} \cup \mathcal{C}$. This permits us to prove termination if all rules of $R$ are of the form $r \rightarrow s$ where the size of all instances of $r$ is larger than the size of corresponding instances of $s$, as in the Loop example. We can let $W$ be the nonnegative reals and differentially weigh symbols by letting $\tau_f(x_1 \ldots x_n) = c_f + x_1 + \ldots + x_n$, where $c_f$ is a positive real number depending on $f$. Or we can obtain a quasi-termination function to the naturals by letting $\tau_f(x_1 \ldots x_n) = 1 + \max\{x_i\}$, which computes the depth of a term and shows termination when all rules are depth-reducing. Loops (Example 1.5) may be also be shown terminating by the depth measure.

More generally, we can obtain termination functions by letting $\tau_f$ be an arbitrary multivariate polynomial in $\vec{x}$ with integer coefficients, but then it is necessary to verify monotonicity for each such polynomial separately. Theorem proving techniques can be applied to this problem; implementations include [Ben Cherifa and Lescanne 1987, Steinbach 1994, Giese 1995]. One must also show that
for given rewrite systems.

for obtaining such polynomials are given in [Steinbach 1994].

embedding [Plaisted 1985].

ordering will work.

given system is terminating. (Recall that integers show a decrease with each rewrite [Lankford 1975]. This can automate

termination [Dershowitz 1979], not just to check that given polynomials over the natural numbers so that the ordering > on integers is well-founded. Of course, there are some systems for which no such polynomial ordering will work. It is known that if the termination of \( R \) can be shown using such polynomials, then the length of the reduction sequences from a term \( t \) is at most doubly-exponential in the size of \( t \) [Hofbauer and Lautemann 1989]. This limits the kind of functions that can be computed by systems whose termination can be shown with these orderings. Also, it is not easy to devise appropriate polynomials for given rewrite systems.

Applying Theorem 4.13, we can extend polynomial orderings to the real numbers. Though the range of the polynomials is not well-ordered, termination is guaranteed by the theorem; see [Dershowitz 1982]. An interesting sidelight is that it is then possible to use a decision procedure for quantified inequalities involving polynomials over the reals, such as Tarski's original method [Tarski 1951] or the “reasonably practical” cylindrical algebraic decomposition algorithm [Arnon, Collins and McCallum 1984], to decide if polynomials of given degree exist that prove termination [Dershowitz 1979], not just to check that given polynomials over the integers show a decrease with each rewrite [Lankford 1975]. This can automate the method, but at a considerable cost, since the best upper bounds for the \( \exists \forall \) fragment of reals (with addition, multiplication, and inequalities) are quite large (multiply-exponential in the number of variables) [Renegar 1992]. Some heuristics for obtaining such polynomials are given in [Steinbach 1994].

One can obtain evidence that a system \( R \) is terminating by demonstrating that \( R#(t) \) is finite for “convincingly” many terms \( t \), which is necessarily the case if \( R \) is terminating. (Recall that \( R#(t) \) is the length of the longest derivation beginning with \( t \).) Indeed, if \( R \) is terminating, then \( s \to t \) implies \( R#(s) > R#(t) \). However, \( R# \) need not be monotonic. In addition, it cannot be defined as a homomorphism, since \( R#(f(t_1, \ldots, t_n)) \) depends not only on \( R#(t_i) \), but also on the structure of the \( t_i \).

We can also secure evidence that a system \( R \) is not simply terminating, as follows:

4.21. THEOREM. Given a finite rewrite system \( R \) and a term \( t \), it is decidable whether \( R \) is self-embedding on \( t \).

PROOF. If there is a self-embedding of \( R \), one can be found by enumerating derivations from \( t \) until a self-embedding is found. If no such self-embedding exists, then by Theorem 4.16 no derivation from \( t \) can be infinite, and thus there are only finitely many such derivations, by König’s Lemma. These can be listed exhaustively, demonstrating that \( R \) is not self-embedding on \( t \). In either case, it is decidable whether a given system \( R \) is self-embedding on \( t \). \( \Box \)

It is undecidable if there exists some term for which a given system is self-embedding [Plaisted 1985]. If \( R \) is self-embedding for some \( t \), then \( R \) cannot be
simply terminating. On the other hand, if \( R \) is simply terminating, then this can often be demonstrated, since many computable simplification orderings are known (as we will see below). Combining these approaches, one may be able to determine automatically whether a system is simply terminating.

There are also techniques which can show nontermination of rewrite systems; for example, a rewrite system \( R \) is nonterminating if there is a term \( r \), context \( u \), and substitution \( \sigma \) such that \( r \xrightarrow{\sigma} u[r] \). By using methods for proving termination together with this technique for establishing nontermination, one may be able to decide automatically for many naturally arising systems whether they are terminating.

The recursive path ordering [Dershowitz 1982] and its many variants have proved to be very useful for proving termination of rewrite systems. These orderings can, for example, easily handle the distributive rule \( x \times (y + z) \rightarrow x \times y + x \times z \). In addition, for many common systems, it is easy to find a recursive path ordering that suffices for termination proofs, in contrast to polynomial orderings. These orderings are defined in terms of a precedence on function symbols, which is a quasi-ordering on the symbols.

There are two main versions of the recursive path ordering, one based on bags of subterms (the multiset path ordering) and the other on sequences (the lexicographic path ordering). We begin with the sequence version:

4.22. Definition (Lexicographic Path Ordering [Kamin and Lévy 1980]). The lexicographic path ordering \( \succeq_{lpo} \) induced by a quasi-ordering \( \succeq \) on function symbols is defined as follows:

\[
\begin{align*}
\cdashline{3-4}
\text{if } f \succeq g & \quad \text{then } f(s_1, \ldots, s_m) \succeq_{lpo} g(t_1, \ldots, t_n) \\
\text{if } f(s_1, \ldots, s_m) \succeq_{lpo} g(t_1, \ldots, t_n) & \quad \text{if } f \simeq g \quad \text{then } \langle s_1, \ldots, s_m \rangle \simeq_{lpo} \langle t_1, \ldots, t_n \rangle \\
\text{if } f \simeq g & \quad \text{then } f(s_1, \ldots, s_m) \simeq_{lpo} g(t_1, \ldots, t_n) \\
\text{if } f(s_1, \ldots, s_m) \simeq_{lpo} g(t_1, \ldots, t_n) & \quad \text{if } f \succeq g \quad \text{then } f(s_1, \ldots, s_m) \succeq_{lpo} g(t_1, \ldots, t_n)
\end{align*}
\]

where \( \succeq_{lpo} \) is the union of the equivalence relation \( \simeq_{lpo} \) and the partial ordering \( \succeq_{lpo} \) and \( \succeq_{lex} \) and \( \simeq_{lex} \) are the lexicographic extensions of \( \succeq_{lpo} \) and \( \simeq_{lpo} \), respectively (see Definition 4.8).

Recall that \( f \succeq g \) iff \( f \succeq g \) but not \( g \succeq f \) and \( f \simeq g \) iff \( f \succeq g \) and \( g \succeq f \).

The lexicographic path ordering is a simplification ordering for systems having fixed-arity function symbols [Kamin and Lévy 1980]. If the precedence \( \succeq \) is total, so is \( \succeq_{lpo} \).

This ordering has the useful property that \( f(f(x, y), z) \succeq_{lpo} f(x, f(y, z)) \); informally, the reason is that—which although the terms have the same size—the first subterm \( f(x, y) \) of \( f(f(x, y), z) \) is always larger than the first subterm \( x \) of \( f(x, f(y, z)) \). In this ordering, one can easily prove termination of Ackermann’s function, for instance.
4.23. Example (Distributivity). Suppose $\times \succ +$ in the precedence on function symbols. Then we can show that $x \times (y + z) \succ_{\text{opo}} x \times y + x \times z$ (in the stable extension of the ordering), as follows:

\[
\begin{array}{c}
\frac{x \approx_{\text{ipo}} x}{x \times (y + z) \succ_{\text{lex}} (x, y)} \frac{y \approx_{\text{ipo}} y}{x \times (y + z) \succ_{\text{ipo}} y} \frac{z \approx_{\text{ipo}} z}{x \times (y + z) \succ_{\text{ipo}} z}
\end{array}
\]

\[
\begin{array}{c}
(\times, (y + z)) \succ_{\text{ipo}} (x, y) \quad (\times, (y + z)) \succ_{\text{ipo}} (x, z)
\end{array}
\]

\[
\begin{array}{c}
x \times (y + z) \succ_{\text{ipo}} x \times y + x \times z
\end{array}
\]

4.24. Definition (Multiset Path Ordering [Dershowitz 1982]). The multiset path ordering $\succ_{\text{mpo}}$, induced by a quasi-ordering $\succ$ on function symbols, is defined as follows:

\[
\begin{array}{c}
s_i \approx_{\text{mpo}} t \quad f \succ g
\end{array}
\]

\[
\begin{array}{c}
(f(s_1, \ldots, s_m) \succ_{\text{mpo}} g(t_1, \ldots, t_n))
\end{array}
\]

\[
\begin{array}{c}
(f(s_1, \ldots, s_m) \approx_{\text{bag}} [t_1, \ldots, t_n] \quad f \approx g
\end{array}
\]

\[
\begin{array}{c}
(s_1, \ldots, s_m) \approx_{\text{mpo}} g(t_1, \ldots, t_n)
\end{array}
\]

where $\approx_{\text{mpo}}$ is the union of the equivalence relation $\approx_{\text{mpo}}$ and the partial ordering $\succ_{\text{mpo}}$ and $\approx_{\text{bag}}$ is the bag extension of $\approx_{\text{mpo}}$, as in Definition 4.9.

The multiset path ordering is a simplification ordering [Dershowitz 1982]. When the precedence is total, the ordering is total up to permutations of arguments.

Suppose we want to show that $f(g(a, b), d) \succ_{\text{mpo}} f(g(b, a), c)$ if $d \succ c$. (Recall that the precedence on function symbols is a quasi-ordering.) Using the above inference rules, we have that $g(a, b) \approx_{\text{mpo}} g(b, a)$ hence $[g(a, b), d] \succ_{\text{mul}} [g(b, a), c]$ so $f(g(a, b), d) \succ_{\text{mpo}} f(g(b, a), c)$.

4.25. Example (Permuted Distributivity). Suppose $\times \succ +$ in the precedence. Then we can show that $x \times (y + z) \succ_{\text{mpo}} x \times y + x \times z$ (in the stable extension of the ordering), as follows: we have:

\[
\begin{array}{c}
x \approx_{\text{mpo}} x \quad y \approx_{\text{mpo}} y \quad z \approx_{\text{mpo}} z
\end{array}
\]

\[
\begin{array}{c}
x \times (y + z) \approx_{\text{mpo}} x \times (y + z) \approx_{\text{mpo}} x \times (y + z)
\end{array}
\]

\[
\begin{array}{c}
x \times (y + z) \approx_{\text{bag}} [x, z] \quad x \times (y + z) \approx_{\text{bag}} [x, y]
\end{array}
\]

\[
\begin{array}{c}
x \times (y + z) \approx_{\text{mpo}} x \times (y + z) \times x \times (y + z)
\end{array}
\]

To prove that Insertion Sort (Example 1.4) terminates using $\succ_{\text{mpo}}$, note that five of the rules show a decrease for $\succ_{\text{mpo}}$, by virtue of the subterm case of the definition. For the recursive rules of $\text{max}$ and $\text{min}$, we can use the precedence $\max, \min \succ_s$. For the base case of $\text{insert}$, we let $\text{insert} \succ:_s$ in the precedence. For the recursive rule of $\text{sort}$, we let $\text{sort} \succ \text{insert}$. Finally, for the recursive rule of $\text{insert}$, we make $\text{insert} \succ \max, \min$ in the precedence.
The multiset path ordering can be used to show that Hercules (Example 1.1) is invincible, considering Hydra as a term, nodes as a varyadic function symbol, and head as some constant symbol. We need to show that after each chop and regrowth, Hydra is smaller than before in $\succ_{mpo}$. Removing a head makes a branch smaller in $\succ_{mpo}$. Replacing a branch with any number of smaller branches is a decrease in the multiset path ordering. Thus any sequence of chops and regrowths terminates, which means Hercules wins as long as he keeps chopping, regardless of what Hydra does to retaliate.

An advantage of path orderings is that the precedence is often quite natural. For example, suppose, for some function $f$, $f(s)$ is defined in terms of $f(t_i)$ for $t_i$ simpler than $s$ (say, $t_i$ are subterms of $s$). In other words, we have rules like $f(s) \rightarrow u[f(t_1), \ldots, f(t_n)]$ where $u$ is some multi-hole context containing smaller recursive calls $f(t_i)$. We want to show that this rule is decreasing in the multiset path ordering. Suppose $u$ can be expressed as a composition of functions that have previously been defined. Then we can choose the precedence so that $f$ is greater than any previously defined symbol. Since $s \succ t_i$ for all $i$, $f(s) \succ f(t_i)$ for all $i$. Also, $f(s) \succ g(f(t_1), \ldots, f(t_n))$ if $f(s) \succ f(t_i)$ for all $i$. Using this rule repeatedly, we obtain that $f(s) \succ u[f(t_1), \ldots, f(t_n)]$, and the rule $f(s) \rightarrow u[f(t_1), \ldots, f(t_n)]$ is decreasing.

4.26. Example (Factorial). As an example, consider the following straightforward rewrite system to compute the factorial function:

$$
egin{align*}
0 + x & \rightarrow x \\
0 \times x & \rightarrow 0 \\
fact(0) & \rightarrow s(0) \\
S(x) + y & \rightarrow s(x + y) \\
S(x) \times y & \rightarrow y + (x \times y) \\
fact(s(x)) & \rightarrow s(x) \times fact(x)
\end{align*}
$$

For this system, we choose the precedence $\succ_{mpo}$ for all rules $l \rightarrow r$. This is natural, since factorial is defined in terms of multiplication, multiplication in terms of addition, and addition in terms of the constructors. Using this precedence with the multiset path ordering, we have $l \succ r$ for all rules $l \rightarrow r$. This can be seen by considering the “dominant” subterm on both sides of each rule, by which we mean the subterm with the largest outermost function symbol. For the rule $fact(s(x)) \rightarrow s(x) \times fact(x)$, the dominant term on the left is $fact(s(x))$ and on the right is $fact(x)$. Since $s(x) \succ x$, $fact(s(x)) \succ fact(x)$ so the left-hand side is larger in the ordering. In general, there may not be a single dominant term, in which case the ordering has to be examined more carefully. One can show that $l \succ r$ if some dominant term in $l$ is larger than all the dominant terms in $r$.

For any primitive recursive function, there is a rewrite system $R$ computing it whose termination can be shown using a multiset path ordering [Plaisted 1978]. Primitive recursive definitions are encoded in the obvious way (as in Example 4.26) and the precedence is just the hierarchy of definitions. Moreover, Hofbauer [1992] showed conversely that if the termination of $R$ can be shown using a multiset path ordering, then $R$ computes a primitive recursive function (that is, the
function from a term to its \(R\)-normal form is primitive recursive) and Weiermann [1995] showed that lexicographic path orderings imply multiply recursive derivation lengths. Adding certain restrictions to the definition of the path orderings guarantees polynomial computations [Cichon and Marion 1999].

Total recursive functions can also be expressed as terminating rewrite systems. A function \(f\), defined by minimization of a predicate \(t\), may be encoded via rules

\[
\begin{align*}
f(\ldots x_i \ldots) & \rightarrow \mu_t(0, t(0, \ldots x_i \ldots, \ldots x_i \ldots)) \\
\mu_t(n, s(z), \ldots x_i \ldots) & \rightarrow n \\
\mu_t(n, 0, \ldots x_i \ldots) & \rightarrow \mu_t(s(n), t(s(n), \ldots x_i \ldots, \ldots x_i \ldots))
\end{align*}
\]

where \(t(n, \ldots x_i \ldots)\) is a predicate, with the added proviso that \(t(k, \ldots x_i \ldots)\) implies \(t(j, \ldots x_i \ldots)\) for all \(j > k\). (A nonzero value for \(t(n, \ldots x_i \ldots)\) is interpreted as true and a zero value is interpreted as false.) The normal form of a term \(f(\ldots a_i \ldots)\) is the minimum \(n\) such that \(t(n, \ldots a_i \ldots) = 0\). We assume that \(t\) can also be evaluated by a collection of rewrite rules.

The multiset and lexicographic path orderings can be directly combined using the notion of “status” [Lescanne 1990]. The idea is that for some function symbols \(f\), when \(f(s_1, \ldots, s_m)\) and \(f(t_1, \ldots, t_n)\) are compared, the subterms are compared recursively using the bag ordering, while for other function symbols, subterms are compared using the lexicographic ordering (with the subterms arranged from left-to-right or right-to-left or in any fixed order). The name “recursive path ordering” will be used to refer to this status-based combination \(\succ_{\text{rpo}}\).

A number of relationships between termination orderings and large ordinals have been found; this is only natural since any well-ordering corresponds to some ordinal. It is nice that the recursive path ordering (for total precedence) and other term orderings provide intuitive and useful descriptions of large ordinals. For some relevant discussions, see [Dershowitz 1987, Gallier 1991].

It is useful to be able to define termination orderings that are partial orderings and to be able to extend them piecemeal to more powerful orderings.\(^5\) Therefore, an important issue is incrementality, which means that a stronger precedence makes a stronger ordering: An ordering \(\succ_o\) based on a precedence \(\succ\) has the incrementality property if whenever a precedence \(\succ'\) extends \(\succ\), the induced ordering \(\succ'_o\) extends \(\succ_o\). The path orderings have this property, which allows one to successively extend the path ordering, as needed to orient more and more rules. This is one reason why the use of partial orderings is more flexible than the direct use of ordinal notations.

There are many other termination orderings that are similar to the above ones, such as the path of subterms ordering [Plaisted 1978] and the recursive decomposition ordering [Jouannaud, Lescanne and Reinig 1982]. These agree when the precedence is total [Rusinowitch 1987] and they all enjoy the incrementality property. This suggests the possibility of computing a “maximal” ordering for this class, which is as powerful as all of them combined.

\(^5\)One ordering \(\succ'\) extends another \(\succ\) if \(x \succ y\) implies \(x \succ' y\), that is, if \(\succ \subseteq \succ'\), with orderings viewed as sets of pairs.
Define the maximal multiset path ordering for a given precedence \( \succeq \), as follows:

\[
\preceq_{\text{mpo}}^{\sup} = \bigcap (\circlearrowright_{\preceq} \preceq_{\text{mpo}})
\]

Only total extensions \( \succeq' \) of the precedence are considered. See [Detlefs and Forgaard 1985]. Computing the maximal ordering in this way may be expensive, since there may be many total extensions. By using the inference rules defining the path ordering in a goal-directed manner, it is possible to construct a reasonably efficient decision procedure for term inequalities in this ordering. For example, suppose the precedence is \( f > b \) and \( a > g \). The terms \( f(a) \) and \( g(b) \) are incomparable in the multiset path ordering. Nevertheless, we can show that in any total extension \( \succeq' \) of the precedence, \( f(a) \succeq_{\text{mpo}} g(b) \). If \( f \geq a \) then \( f > g \), and \( f \) is the maximal symbol of the two terms and \( f(a) \succeq_{\text{mpo}} g(b) \). If \( f \geq a \) then \( a \) is the maximal symbol of the two terms, and \( f(a) \succeq_{\text{mpo}} g(b) \). Since one or the other must hold in any total extension of the precedence, we have \( f(a) \succeq_{\text{mpo}}^{\sup} g(b) \).

A more efficient way of determining if \( s \succeq_{\text{mpo}}^{\sup} t \) is to construct a constraint involving the precedence \( \succeq \) and function symbols such that \( s \succeq_{\text{mpo}} t \) iff the constraint \( B(\succeq) \) holds. This can be derived systematically from the rules for the multiset path ordering. To decide whether \( s \succeq_{\text{mpo}}^{\sup} t \), we need to show that all extensions of \( \succeq \) to a total precedence \( \succeq' \) satisfy \( B \). This avoids repeated computations on the term structures. The precedence \( \succeq \) can itself be defined by a conjunction of inequalities of the form \( f_1 > g_1, f_2 \geq g_2, \) etc. Call this conjunction of inequalities on function symbols \( C \) and let \( Q \) be the axioms of total quasi-orderings, namely, reflexivity, transitivity, and totality. We need to show that \( C \land Q \implies B \). This is a straightforward theorem proving problem, and can be approached by a number of methods that are reasonably efficient on small formulae. In addition, one can use theorem proving to help compute the stable extension.

Determining if a precedence exists that makes two ground terms comparable in the multiset path ordering is \( \text{NP-complete} \) [Krishnamoorthy and Narendran 1985], but an inequality on ground terms can be decided in quadratic time, using a dynamic programming algorithm. The first-order theory of these orderings is undecidable [Comon and Treinen 1997]. The existential fragment, needed for completion (see Section 6), of the recursive path ordering—at least for total precedences—was shown decidable in [Comon 1990, Jouannaud and Okada 1991], and \( \text{NP-complete} \) in [Nieuwenhuis 1993, Narendran, Rusinowitch and Verma 1998].

Another important class of orderings, the \textit{numeric path orderings}, uses homomorphic interpretations, perhaps in conjunction with a precedence. The \textit{Knuth-Bendix ordering} is an example of such a hybrid ordering. For the Knuth-Bendix ordering with variables, an algorithm to decide inequalities was given in [Dick, Kalms and Martin 1990]. In [Korovin and Voronkov 2001] a polynomial-time algorithm was given to decide whether there is a fully-invariant extension of a given Knuth-Bendix ordering that orients a given rewriting system. If such an ordering exists, the algorithm computes its parameters. The existential fragment is \( \text{NP-complete} \) [Korovin and Voronkov 2000].
Polynomial interpretations, followed by a lexicographic comparison of subterms, were suggested in [Lankford 1979]. Of course, one can use classes of interpretations other than polynomials, though it may be harder to decide inequalities. More advanced uses of semantics in path orderings have been defined; see [Kamin and Lévy 1980, Dershowitz and Hoot 1995, Zantema 1995, Genet and Gnadig 1997, Borralleras, Ferreira and Rubio 2000].

The method of dependency pairs [Arts and Giesl 2000] limits the pairs that need to be shown decreasing by a reduction ordering to establish termination. Observe that if a system is nonterminating, then there must be an infinite derivation with at least one redex at the top of a term. Also if a system is nonterminating, then there’s an infinite derivation in which all proper subterms of every redex initiate only finite derivations. Thus, to show impossibility of any infinite rewrite derivation it suffices to show the existence of some well-founded (not necessarily monotonic) order > such that lσ > σ for nonvariable subterms s of the right side of a rule l → r, and that lσ > t for all t derivable from σ without any top-level rewrite. One way to establish that this termination condition holds is to show that u ≥' v whenever u rewrites to v (using a quasi-simplification ordering), but that lσ > σ for each nonvariable subterm s of the right-hand side using another, related (non-monotonic) ordering for which u ≥' v implies f(...u...) ≥ f(...v...), for all f. Moreover, one can weed out sterile, finite derivations by using data-flow techniques. For example, terms s that are headed by a constructor should be minimal in the ordering.

For right-linear systems one need only show that there are no infinite forward closures [Dershowitz 1981]; the same is true for systems with no “critical pairs” (defined in the next section) [Geupel 1989]; for left-linear systems, it is enough to consider overlap closures [Guttag, Kapur and Musser 1983, Geupel 1989].

As a comparative example, consider the first four rules of Example 4.26:

\[
\begin{align*}
0 + x & \rightarrow x & 0 \times x & \rightarrow 0 \\
\text{s}(x) + y & \rightarrow \text{s}(x + y) & \text{s}(x) \times y & \rightarrow y + (x \times y)
\end{align*}
\]

We can use the following termination function τ₀ : \(T \rightarrow N\):

\[
\begin{align*}
\tau₀(0) & = 1 & \tau₀(\text{s}(x)) & = \tau₀(x) + 2 \\
\tau₀(x + y) & = 2\tau₀(x) + \tau₀(y) & \tau₀(x \times y) & = \tau₀(y) \cdot 2^{\tau₀(x)}
\end{align*}
\]

Or, to avoid exponentials, we could use the lexicographic combination \(\langle τ₁(t), τ₂(t)\rangle\) of two simpler interpretations:

\[
\begin{align*}
τ₁(0) & = 1 & τ₂(0) & = 0 \\
τ₁(\text{s}(x)) & = τ₁(x) + 2 & τ₂(\text{s}(x)) & = τ₂(x) + 2 \\
τ₁(x + y) & = τ₁(x) + τ₁(y) & τ₂(x + y) & = 2τ₂(x) + τ₂(y) \\
τ₁(x \times y) & = τ₁(x) \cdot τ₁(y) & τ₂(x \times y) & = 0
\end{align*}
\]

Rather than pairs of interpretations, one could—interchangeably—interpret every term as a pair of numbers, and define homomorphisms for each function symbol
that extracted the appropriate components from the interpretations of subterms and then put them together again [Zantema 1994].

For the dependency pair method, one can use the “natural” interpretation:

\[
\begin{align*}
\tau'(0) &= 0 \\
\tau'(x + y) &= \tau'(x) + \tau'(y) \\
\tau'(x \times y) &= \tau'(x) \cdot \tau'(y)
\end{align*}
\]

to show that \( u \rightarrow t \) implies \( u \geq t \). Then for \( > \) use the termination function:

\[
\begin{align*}
\tau(0) &= \langle 0, 0 \rangle \\
\tau(s(x)) &= \langle 0, 0 \rangle \\
\tau(x + y) &= \langle 1, \tau'(x) \rangle \\
\tau(x \times y) &= \langle 2, \tau'(x) \rangle
\end{align*}
\]

which expresses the fact that the recursion is on the first argument and that multiplication is defined in terms of addition. We have \( s(x) + y > s(x + y), x + y \) and \( s(x) \times y > y + x \times y, x \times y \). The top two rules can be ignored.

Of course, either the multiset or the lexicographic path ordering can be used, simply with the precedence \( \times > + > s \). If however the fourth rule were changed to read \( s(x) \times y \rightarrow \langle y \times x \rangle + y \), then only the multiset version would work.

5. Church-Rosser Properties

A key property a rewrite system can enjoy is confluence, because it reduces validity testing to rewriting. To begin with:

5.1. Theorem (Birkhoff’s Theorem [1935]). For any equational system \( E \) and terms \( t \) and \( u \), \( E \models t = u \) iff \( t \leftrightarrow^*_E u \).

In other words, \( t = u \) holds in all models of the identities \( E \) iff there is a finite sequence \( v_1, v_2, \ldots, v_n \) of terms such that \( t \equiv v_1 \) and \( u \equiv v_n \) and for each \( i \), \( v_{i+1} \) is obtained from \( v_i \) by replacing a subterm \( r \) of \( v_i \) by a term \( s \), where the equation \( r = s \) or the equation \( s = r \) is an instance of an equation in \( E \). This gives an inefficient method for deriving logical consequences of sets of equations. Paramodulation improves on the naïve search for proofs; see [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

Let \( R \) be a rewrite system \( \{ r_1 \rightarrow s_1, r_2 \rightarrow s_2, \ldots \} \) and recall that \( \equiv_R \) is the associated equational system \( \{ r_1 = s_1, r_2 = s_2, \ldots \} \). We write \( t \equiv_R u \) iff \( \equiv_R \models t = u \), that is, the equation \( t = u \) is a logical consequence of the associated equational system. The relation \( =_R \) is thus the smallest congruence relation generated by \( R \), in algebraic terms. The relation \( =_R \) is defined semantically, and the relation \( \rightarrow^* \) is defined syntactically. We would like to find relationships between these two concepts to be able to compute properties of \( =_R \) and to find complete restrictions of the inference rules suggested by Birkhoff’s Theorem. As we will see, when \( R \) has certain properties, some of them decidable, then \( t =_R u \) iff some normal form of \( t \) is identical to some normal form of \( u \).
For any binary relation \( \rightarrow \), we say that \( s \) derives \( t \) when \( s \rightarrow^* t \) and that \( s \) and \( t \) are convertible when \( s \leftrightarrow^* t \). If there is an element \( u \) mutually derivable from \( s \) and \( t \) (\( s \rightarrow^* u \) and \( t \rightarrow^* u \)), we write \( s \Downarrow t \) and say that they are joinable, and that \( s = t \) has a rewrite proof. We write \( s \Uparrow t \), and say that \( s \) and \( t \) are meetable, if there is an \( r \) such that \( r \rightarrow^* s \) and \( r \rightarrow^* t \).

5.2. Definition (Church-Rosser). A binary relation is Church-Rosser if any two elements are joinable whenever they are convertible. Symbolically: \( \leftrightarrow^* \subseteq \Downarrow \).

5.3. Definition (Confluence). A binary relation \( \rightarrow \) is confluent if any two elements are joinable whenever they are meetable. Symbolically: \( \Uparrow \subseteq \Downarrow \).

The meaning of this is that diverging derivations can always be “brought together”.

5.4. Theorem. A binary relation has the Church-Rosser property iff it is confluent.

Proof. Let \( \Uparrow^n \) be the \( n \)-fold composition of \( \Uparrow \) and \( \Uparrow^* \) its reflexive-transitive closure. We show that \( \Uparrow \subseteq \Downarrow \) (confluence) implies \( \Uparrow^* \subseteq \Downarrow \) (Church-Rosser, since \( \leftrightarrow^* = \Uparrow^* \)).

Trivially, \( \Uparrow^0 \subseteq \Downarrow \). For \( n > 0 \), we have

\[
\Uparrow^n = \Uparrow^{n-1} \circ \Uparrow \subseteq \Uparrow^{n-1} \circ \Downarrow
\]

by confluence. By definition (whether \( n = 1 \) or not),

\[
\Uparrow^{n-1} \circ \Downarrow \subseteq \rightarrow^* \circ \Uparrow^{n-1} \circ \leftarrow^*
\]

and by induction

\[
\rightarrow^* \circ \Uparrow^{n-1} \circ \leftarrow^* \subseteq \rightarrow^* \circ \Downarrow \circ \leftarrow^* = \Downarrow
\]

\(\square\)

Since \( s \leftrightarrow_R^n t \) iff \( s =_R t \), this theorem connects the equational theory of \( R \) with rewriting. In order to decide if \( s =_R t \) it is only necessary to see if \( s \) and \( t \) have a common normal form.

Often, we are only interested in confluence for ground (variable-free) terms.

5.5. Definition (Ground Confluence). A rewrite system \( R \) is ground confluent if for all ground terms \( r \), if \( r \rightarrow_R^* s \) and \( r \rightarrow_R^* t \) then \( s \Downarrow_R t \).

In other words, the rewrite relation, restricted to ground terms, is confluent.

Top-down and bottom-up tree automata execute a special form of ground rewriting and have been successfully used for proving decision properties in the ground case [Thomas 1990]. Confluence of ground systems is decidable [Dauchet, Tison, Heuillard and Lescanne 1987, Oyamaguchi 1987]. The unique normal form property [Verma, Rusinowitch and Lugiez 2001] is also polynomial for ground systems. But ground confluence of non-ground systems is undecidable [Kapur, Narendran and Otto 1990]. The latter property is useful for the so-called “inductionless induction” method invented by Musser [1980]; See [Comon 2001] (Chapter 14 of this Handbook).
5.6. Definition (Convergence). Terminating confluent relations are called convergent.

By extension, a rewrite system is called convergent if its rewrite relation is. Many such systems are known.

Termination means that a rewriting process, applied to a term, will eventually stop, no matter how the rules are applied. If $R$ is terminating, we can always find a normal form of a term by any rewrite sequence continued long enough. However there can be more than one normal form. But a convergent relation $R$ defines unique normal forms, and it can be viewed as a function $R(x)$ from elements $x$ to their normal forms. A convergent rewrite system gives a decision procedure for its equational theory, since for terms $r$ and $s$, $r =_R s$ iff $r \leftrightarrow^*_R s$ (by Birkhoff’s theorem) iff $r \downarrow s$ (by confluence) iff $R(r) = R(s)$ (by termination). The latter is a directed form of theorems proving for such an equational theory.

The straightforward encoding of primitive recursive functions as rewrite systems using the successor notation for natural numbers is convergent.

The next property we define is interesting because it permits a proof of confluence without assuming termination:

5.7. Definition (Strong Confluence). A binary relation $R$ is strongly confluent if for all $r$, $s$, and $t$, $r \rightarrow s$ and $r \rightarrow t$ imply that $s \leftrightarrow t$, or $s$ and $t$ are identical, or there is a term $u$ such that $s \rightarrow u$ and $t \rightarrow u$.

5.8. Theorem ([Newman 1942]). Strongly confluent binary relations are confluent.

The proof is by induction on the length of rewrite sequences. We invite the reader to construct the proof, which is straightforward.

Huet [1980] defines a weaker version of “strong confluence” which also allows for $s \rightarrow^* t$ together with $t \rightarrow^* s$, and which still gives confluence.

Rewrite systems are typically not strongly confluent, while ordinary confluence, as stated, looks like a difficult property to demonstrate. However, we will see that when $R$ is terminating and finite, confluence is decidable.

5.9. Definition. Binary relations $\rightarrow_R$ and $\rightarrow_S$ (sub-) commute if $\leftarrow_R \circ \rightarrow_S \subseteq \rightarrow_S \circ \leftarrow_R$. Recall that $\rightarrow^*$ is the reflexive closure of $\rightarrow$, allowing for at most one step.

5.10. Lemma (Hindley-Rosen Lemma [Hindley 1964, Rosen 1973]). If two relations are each strongly confluent and the two commute, then their union is confluent.

The following widely-applicable criterion for confluence extends the Hindley-Rosen method by dividing a relation $\rightarrow$ into a family of subrelations $\rightarrow_i$. For any set $K$ of indices $i$, we define $\rightarrow_K = \bigcup_{i \in K} \rightarrow_i$. 
5.11. THEOREM ([van Oostrom 1994]). Let $K$ be a well-founded set of indices, divided into two not-necessarily disjoint sets $R$ and $S$. Suppose the relation $\rightarrow_i \circ \rightarrow_j$ is contained in the relation $\rightarrow'_i \circ \rightarrow'_j \circ \rightarrow'_{K'} \circ \rightarrow'_{K'} \circ \rightarrow'_{K'} \circ \rightarrow'_{K'}$, for all $i \in R$ and $j \in S$, where $I' = \{ k \in S | k < i \}$, $J' = \{ k \in R | k < j \}$, and $K' = I' \cup J'$. Then $\rightarrow_R$ and $\rightarrow_S$ commute.

The proof uses a bag ordering that ignores “noise” steps with indices that are smaller than subsequent steps.

5.12. DEFINITION (Local Confluence). A binary relation $\rightarrow$ is locally confluent (weakly Church-Rosser) if for all terms $r$, $s$, and $t$, if $s \rightarrow t$ whenever $r \rightarrow s$ and $r \rightarrow t$.

The following result connects local and global confluence:

5.13. LEMMA (Diamond Lemma [Newman 1942]). A terminating binary relation is Church-Rosser iff it is locally confluent.

PROOF. Clearly Church-Rosser implies local confluence, even without termination. Suppose $s \leftrightarrow^* t$ for a locally confluent and terminating relation $\rightarrow$. Then there is some sequence $r_1, r_2, \ldots, r_n$ of elements such that $s \equiv r_1$ and $t \equiv r_n$ and for all $i$, $r_i \rightarrow r_{i+1}$. For such a conversion, consider the bag $S = [r_1, \ldots, r_n]$, ordered by the well-founded bag extension of the well-founded ordering $\rightarrow^+$ (the relation is terminating). If for no element $r_i$ do we have $r_{i-1} \rightarrow r_i \rightarrow r_{i+1}$ then $s \downarrow t$ immediately. Suppose for some $r_i$ we have $r_{i-1} \rightarrow r_i \rightarrow r_{i+1}$. Since the relation is locally confluent, $r_{i-1} \downarrow r_{i+1}$. Thus there is another conversion between $s$ and $t$ in which $r_i$ is replaced by all the elements participating in the derivation of $r_{i-1} \downarrow r_{i+1}$. These elements are all smaller than $r_i$ in the element ordering. Let $T$ be the bag for this new conversion. Since $r_i$ in $S$ has been replaced by smaller elements, $S \succcurlyeq_{bag} T$. By induction on the well-founded ordering $\succcurlyeq_{bag}$, the derivation for $T$ can be brought to the desired form, that is, $s \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots \leftarrow t_2 \leftarrow t_1 \leftarrow t$, so $s \downarrow t$.

In order to show that a rewrite system is locally confluent, it is necessary to consider critical pairs, the computation of which requires most general unifiers. A substitution $\alpha$ is a unifier of two terms $s$ and $t$ if $s \equiv \alpha$ is identical to $t \alpha$. In this case we say that $s$ and $t$ are unifiable. We say that a substitution $\alpha$ is as general as a substitution $\beta$ relative to a pair $s = t$ of terms if there is a substitution $\gamma$ such that $s \alpha \gamma \equiv s \beta$ and $t \alpha \gamma \equiv t \beta$. A substitution $\alpha$ is a most general unifier of terms $s$ and $t$, denoted $\text{mgu}(s, t)$, if $\alpha$ is a unifier of $s$ and $t$ and $\alpha$ is as general as any other unifier of $s$ and $t$.

5.14. DEFINITION (Critical Pair [Knuth and Bendix 1970]). If $s \rightarrow t$ and $l \rightarrow r$ are two (not necessarily distinct) rewrite rules (but with variables renamed so that

If $s$ and $t$ are unifiable then most general unifiers exist and are easy to compute. Unification is the subject of Chapter 8 of this Handbook [Baader and Snyder 2001].
they are distinct) and \( \mu \) is a most general unifier of \( l \) and a nonvariable subterm \( s' \) of \( s \), then the equation \( sp[l[\mu]] = tp[\mu] \), where \( rp[\mu] \) has replaced \( s' \mu \) (= \( l \mu \)) in \( sp[\mu] \) is a critical pair of those rules. We also require that either the two rules are distinct or that \( s' \) is a proper subterm of \( s \) to prohibit the trivial critical pair of a rule with itself. A critical pair \( u_1 = u_2 \) is joinable if \( u_1 \) and \( u_2 \) are joinable. A critical pair is an overlay if \( s' \) is \( s \) (and the two rules \( s \rightarrow t \) and \( t \rightarrow r \) are distinct). A rewrite system is non-overlapping if there are no critical pairs between its rules (disallowing the trivial one between a rule and itself).

The idea is that the term \( sp[l[\mu]] \) can rewrite either to \( tp[\mu] \), applying the first rule at the top, or can rewrite to \( sp[r[\mu]] \), applying the second rule to the redex \( l[\mu] \). So if \( u_1 = u_2 \) is a critical pair, then \( u_1 \leftarrow \sigma \rightarrow u_2 \) is a minimalist prototypical non-rewrite proof. A finite rewrite system can only have a finite number of critical pairs.

Insertion Sort has one trivially joinable critical pair, \( 0 = 0 \), formed from the base cases of the \texttt{max} and \texttt{min} functions.

5.15. Lemma (Critical Pair Lemma [Knuth and Bendix 1970, Huet 1980]). A rewrite system is locally confluent iff all its critical pairs are joinable.

5.16. Example (Fragment of Group Theory). Each of the rules

\[
\begin{align*}
0 + x & \rightarrow x & x + 0 & \rightarrow x \\
(-x) + x & \rightarrow 0 & x + (-x) & \rightarrow 0 \\
-0 & \rightarrow 0 & (-x) & \rightarrow x \\
(-x) + (x + y) & \rightarrow y & x + ((-x) + y) & \rightarrow y
\end{align*}
\]

follows from some combination of the three axioms: \( x + 0 \rightarrow x \), \( 0 + x \rightarrow x \), and \( (-x) + (x + y) \rightarrow y \). This system has numerous critical pairs, all of which are joinable. For example, the rules \( x + (-x) \rightarrow 0 \) and \( x + ((-x) + y) \rightarrow y \) form a critical pair \( x + 0 = (-x) \), both sides of which reduce, via other rules, to \( x \).

Proof. One direction is trivial. For the other, suppose \( u \) is a term and \( u \rightarrow_R s \) and also \( u \rightarrow_R s' \) for rewrite system \( R \). There must be contexts \( l \) and \( l' \rightarrow r' \) of \( R \) and substitutions \( \sigma \) and \( \sigma' \) such that \( u \equiv t[l[\sigma]] \equiv c[l'[\sigma']] \), \( s \equiv t[r[\sigma]] \), and \( s' \equiv t[l'[\sigma']] \). We need to show that \( s \downarrow_R s' \).

If the redexes \( l[\sigma] \) and \( l'[\sigma'] \) are disjoint in \( u \), then \( u \equiv t[l[\sigma], l'[\sigma']] \equiv t[r[\sigma], l'[\sigma']] \) and \( s \equiv t[l[\sigma], r'[\sigma']] \), for some context \( l \). Then immediately \( s \downarrow_R s' \), since \( s, s' \rightarrow_R t[l[r[\sigma], r'[\sigma']] \).

Another possibility is that one redex is “inside a variable” of another; that is, \( u \equiv c[l[\sigma]] \), with \( \sigma \) instantiating some variable \( x \) to a term \( t[l'[\sigma']] \) containing an instance of \( l' \). In this case, we can view \( l[\sigma] \) as \( l[\tau][l'[\sigma']] \), where \( \tau \) is the same as \( \sigma \), except that instead of \( x \mapsto l'[\tau][l'[\sigma']] \) it maps \( x \) to \( l[\tau][l'[\sigma']] \), with that hole filled by \( l'[\sigma'] \). So \( u \equiv c[l[\tau][l'[\sigma']]] \), \( c' \equiv c[l[\tau]] \), \( s \equiv c[l[\tau]] \equiv c[l[\tau]][l'[\sigma'], \ldots, l'[\sigma']] \), with one occurrence of \( l'[\sigma'] \) filling each hole left by \( \tau \) for an \( x \) that occurs in \( r \), and \( s' \equiv c'[l'[\sigma']][c[l[\tau]][r'[\sigma']]] \rightarrow_R c[l[\tau]][r'[\sigma'], \ldots, r'[\sigma']] \), while \( s \equiv c[l[\tau]][l'[\sigma'], \ldots, l'[\sigma']] \rightarrow_R c[l[\tau]][r'[\sigma'], \ldots, r'[\sigma']] \) by successive rewrites. Thus \( s \downarrow_R s' \).
The only other possibility is that the redexes overlap in a nonvariable subterm. Then \( u \equiv c(\sigma) \), with \( l \equiv d(\ell) \), and \( t \sigma \equiv l' \sigma' \). In this case, the equation \( s = s' \) must contain an instance of a critical pair of \( R \), that is, \( s = s' \equiv c(\sigma \rho) = c(d\rho[r''\sigma']) \equiv c(\rho \tau) = c(\tau \sigma) \), for some critical pair \( p = q \) or \( q = p \) (which is \( r \mu = d\rho[r''\mu] \) for most general unifier \( \mu \) of \( l' \) and \( t \)), and substitution \( \tau \) (such that \( \mu \tau = \sigma \)). Since all critical pairs are joinable, \( \rho \tau \downarrow_R \rho \tau \) and, hence, \( s \downarrow_R s' \).

5.17. Corollary. Suppose \( R \) is a terminating rewrite system. Then \( R \) is convergent iff for all critical pairs \( s = t \) of \( R \), arbitrarily computed normal forms \( s' \) of \( s \) and \( t' \) of \( t \) are identical.

Proof. The system \( R \) is locally confluent by the Critical Pair Lemma 5.15. Since \( R \) is terminating, it is confluent by the Diamond Lemma 5.13.

The implication of this result is that if \( R \) is a (finite) terminating rewrite system, then its local confluence (and confluence) is decidable, since given a finite \( R \), there are only finitely many critical pairs, and they can be computed (in polynomial time). However, often we are interested in systems that are not known to terminate and other methods are needed to establish confluence. Such methods are interesting, even though it is not possible to decide their equational theory by rewriting, because confluence still enables one to prove validity of equations between those terms having normal forms.

5.18. Definition (Encompassment [Huet 1981]). A term \( s \) encompasses a term \( t \) if a subterm of \( s \) is an instance of \( t \). We write \( s \triangleright t \) if \( s \) encompasses \( t \), but not vice-versa.

5.19. Definition (Reduced System [Butler and Lankford 1980]). A system \( R \) is reduced (interreduced) if, for each rule \( l \rightarrow r \) in \( R \), the right-hand side \( r \) is irreducible (unrewritable) and if \( l \triangleright l' \), then \( l' \) is irreducible (proper subterms of \( l \) are in normal form, as is any term more general than \( l \)).

The group theory fragment (Example 5.16) is reduced, but the Interpreter (1.6) isn’t.

5.20. Definition (Reduced Convergent System). A rewrite system is reduced convergent if it is confluent, terminating, and reduced.

It turns out that for a given reduction ordering \( > \) and equational theory \( E \), there is a unique reduced convergent system.

5.21. Theorem (Uniqueness [Butler and Lankford 1980, Métivier 1983]). Two reduced convergent (not necessarily finite) rewrite systems with the same equational theory (that is, their convertibility relations are the same) are identical (up to renaming variants), if their union is terminating.
See [Dershowitz, Marcus and Tarlecki 1988] for details.

A rewrite system is locally confluent if (but not only if) no left-hand side unifies with a nonvariable subterm (except itself) of any left-hand side, taking into account that variables appearing in two rules (or in two instances of the same rule) are always treated as distinct. To get confluence for nonterminating systems we impose an additional requirement of left-linearity:

5.22. Definition (Orthogonality). A left-linear rewrite system is (weakly) orthogonal if all critical pairs are trivial (both sides identical).

In particular, all critical pairs are trivial if all of the following hold:
1. no left-hand side is just a variable;
2. there are no variables on the right side of a rule that do not appear also on the left;
3. no left-hand side unifies with a (not necessarily proper) non-variable subterm of another left side (renamed apart);
4. no left-hand side unifies with a proper non-variable subterm of a renamed version of itself.

Orthogonality is usually defined in this particular sense, but the weaker notion suffices for most purposes.

The importance of orthogonal systems stems from the following:

5.23. Theorem ([Rosen 1973]). Every orthogonal system is confluent.

In particular, our erstwhile interpreter (1.6) is confluent.

Proof. The idea is to show that a parallel rewriting relation \( \rightarrow^\parallel_R \) associated with \( R \) is strongly confluent (since the system may be assumed nonterminating). Parallel rewriting is rewriting at one or more disjoint redexes at the same time. We need to consider parallel rewriting because if \( s \) rewrites to \( t \) and \( s \) rewrites to \( t' \), at positions that are not disjoint, then a subterm of \( s \) may appear many times in \( t \) or \( t' \), and all of these occurrences may have to be rewritten in parallel to obtain a \( v \) to which both \( t \) and \( t' \) rewrite in one (parallel) step. For example, if \( s \equiv f(a) \) and \( R \) is \{ \( a \rightarrow b \), \( f(x) \rightarrow g(f(x), f(x)) \) \} then \( f(a) \) rewrites to both \( f(b) \) and \( g(f(a), f(a)) \). Both of these terms parallel-rewrite to \( g(f(b), f(b)) \), and this requires parallel rewriting of the two occurrences of \( a \) in \( g(f(a), f(a)) \). Similar techniques are used to show the confluence of lambda calculus and combinatory logic, which do not terminate. \( \square \)

The following well-known example [Klop 1980] shows up the essentiality of left-linearity:

\[
\begin{align*}
d(x, x) & \rightarrow e \\
c(x) & \rightarrow d(x, c(x)) \\
a & \rightarrow c(a)
\end{align*}
\]
The system is non-overlapping because the outermost function symbols of each left-hand side are distinct, and there are no non-variable proper subterms. Note that \( c(a) \rightarrow^* c \) and \( c(a) \rightarrow^* c(e) \), but we do not have \( c \downarrow c(e) \).

Even though non-left-linearity removes the confluence guarantee, Chew [1981] suggested that some limited confluence properties still apply to some non-overlapping non-left-linear systems, in terms of the uniqueness of normal forms. For example, his methods imply that the above system has unique normal forms, despite its lack of confluence.

5.24. Example (Combinatory Logic). Combinatory Logic is a prime example of a (nonterminating) orthogonal system:

\[
\begin{align*}
I \circ x & \rightarrow x \\
(K \circ x) \circ y & \rightarrow x \\
((S \circ x) \circ y) \circ z & \rightarrow (x \circ z) \circ (y \circ z)
\end{align*}
\]

The combinators \( K \) and \( S \) were dubbed “kestrel” and “starling” by Smullyan [Smullyan 1985]; \( I \) is the identity combinator; \( \circ \) is composition. This system can be used to implement any recursive function.

5.25. Example (Cartesian Closed Categories). The following non-orthogonal, non-confluent system [Huet 1985] is used in the compilation of functional languages (juxtaposition and pairing \( \langle \circ, \circ \rangle \) are binary operators):

\[
\begin{align*}
Ix & \rightarrow x & (xy)z & \rightarrow x(yz) \\
xI & \rightarrow x & \langle x, y \rangle z & \rightarrow \langle xz, yz \rangle \\
F(x, y) & \rightarrow x & E(Cx, y) & \rightarrow x(I, y) \\
S(x, y) & \rightarrow y & (Cx)y & \rightarrow C(x(yF, S))
\end{align*}
\]

The combinators \( E \) and \( C \) stand for “evaluation” and “Currying”, respectively; \( I \) is the identity morphism; \( F \) and \( S \) project the components of pairs.

There are redex-choosing strategies for orthogonal systems that compute the unique (but not necessarily existent) normal form of a term; see Section 10. There are also some results about termination of orthogonal systems; for example, it is known that an orthogonal system is weakly innermost normalizing (every term has a normal form obtainable by some innermost rewriting sequence) if and only if it is terminating [Gramlich 1995].

Since the factorial example (4.26) is orthogonal, innermost termination implies (strong) termination. This allows one to prove termination by a straightforward natural interpretation and ordinary induction on the value of the first argument.

For orthogonal systems that are non-erasing (the left and right-hand sides of rules have the same variables appearing), a system is weakly normalizing (every term has a normal form) if and only if it is terminating [O’Donnell 1977], as is the case for
the λ calculus. Left-linearity is not actually required for these results [Dershowitz and Hoot 1995, Gramlich 1995].

See [Raoult and Vuillemin 1980, Naoi and Inagaki 1989] for studies of denotational semantics and confluent systems. Though most research in term rewriting has concentrated on confluent systems, there is some interest in systems that model nondeterminism [Kaplan 1988]. For such systems, we may introduce a nondeterministic choice operator with rules \([x][y] \rightarrow x\) and \([x][y] \rightarrow y\).

### 6. Completion

The impractical, “British Museum” approach to constructing a convergent system for a given equational theory \(E\) would be to generate all possible sets of equational consequences of \(E\), orient as many as possible according to some given reduction ordering, and check each subset of the rules for completeness (all \(E\) follows) and confluence (all critical pairs between the resultant rules resolve). As a practical matter, a method, called completion, is used, which converts unresolved critical pairs into rewrite rules. Completion uses a reduction ordering to orient equations, rewriting to simplify rules and equations, and the encompassment ordering to determine which of two rules is more general, and hence preferred. Knuth and Bendix [1970] used their completion program (written in Fortran) to construct convergent systems for free groups, loops (Example 1.5 shown earlier was found manually by Evans [1951]), left and right groups, and central groupoids.

In general, inference systems have rules for combining theorems in a database of proved theorems to obtain new theorems which can be added to the database. The generic version of that process is

\[
\text{Expand: } \quad \frac{E}{E, C} \quad \text{if } E \vdash C
\]

meaning that \(E\) is a collection of theorems that have been proved and are in the database and \(C\) is one or more consequences of \(E\) that can be added to the database.

Completion procedures, which are inference systems, incorporate another schema for deleting redundant entries in the database, ones that cannot contribute to minimal proofs. For that purpose, one employs a proof ordering \(\gg\) in conjunction with the inference rule:

\[
\text{Contract: } \quad \frac{E, C}{E} \quad \text{if for every proof } p \text{ of } E \cup C \vdash e, \text{ there is a proof } q \text{ of } E \vdash e, \text{ such that } p \gg q
\]

For such a theorem-proving method to be complete, every non-minimal proof must contain a subproof that can be reduced with the help of consequences that will eventually be added by the expansion rule.

To get the flavor of the general approach, consider first how rewriting can be used to transform a (finite) set \(E\) of ground equations into a simple decision procedure for \(E\) [Lankford 1975]. This transformation is expressed as a set of rules that use a total
simplification ordering to replace equations with simpler ones, without changing the theory:

\[
\frac{u = u, \ E}{E} \quad \frac{e[l], \ l = r, \ E}{e[r], \ l = r, \ E} \quad \text{if } l > r
\]

The left rule deletes trivial equations and is the simplest contraction rule. The other simplifies an existing equation \( e \) using another equation, it being understood that \( e \) is distinct from \( l = r \). This simplification is a combination of an expansion step that adds \( e[r] \) and a contraction that removes \( e[l] \).

In a completion procedure, these two rules are applied repeatedly as long as possible, assuming totality of the simplification ordering. Regardless of the order in which things are done, this inference process terminates. Done right, it completes a ground system \( E \) in time proportional to \( n \log n \), where \( n \) is the number of symbols in \( E \) [Snyder 1989]. This process is also known as congruence closure (see, for example, [Nelson and Oppen 1980]). The final system of equations, call it \( R \), reduces any term \( t \) to normal form in no more steps than symbols in \( t \). An equation \( s = t \) holds in \( E \) iff \( R \) reduces \( s \) to \( t \) to a trivial equation of the form \( u = u \).

For example, if terms are compared by length, then these rules can have the following effect:

\[
ss0 = 0, \ ss0 = 0, \ sss0 = s0 \quad \vdash \quad sss0 = 0, \ ss0 = 0, \ sss0 = s0
\]

\[
\vdash \quad ss0 = 0, \ ss0 = s0
\]

\[
\vdash \quad ss0 = 0, \ 0 = s0
\]

\[
\vdash \quad s0 = 0
\]

We are using \( E \vdash E' \) to indicate one application of an inference rule. The result is the one-rule rewrite system, \( s0 \rightarrow 0 \), which reduces all terms \( s'0 \) to normal form \( 0 \).

Turning to the general case, when the axioms in \( E \) have variables, the question is how one constructs an equivalent convergent system \( R \), which can be used to check validity by the same method, viz. reducing to normal form. In the version of completion we present, rules and equations are used for simplification. A term \( u[\sigma] \) containing an instance of \( l \) of an equation \( l = r \) or \( r = l \) may be rewritten, via ordered rewriting, to \( u[r\sigma] \) whenever \( u[l\sigma] > u[r\sigma] \) in a given reduction ordering \( > \). If reducibility is recursive, a set of equations \( E \) for which ordered rewriting \( \rightarrow E \) is confluent computes unique normal forms. So, to prove an identity \( s = t \), one Skolemizes its variables (treating its variables as constants), and uses the evolving \( \rightarrow E \) to reduce both its sides. A trivial equation \( s' = t' \) is obtained eventually iff \( E \vdash s = t \).

6.1. Example (Groupoid). Consider the following system for a subvariety of entropic groupoids [Hsiang and Rusinowitch 1987]:

\[
(xy)x \rightarrow x \quad \quad x(yz) \rightarrow xz
\]

\[
(xy)z \leftrightarrow (xw)z \quad ((xy)w)z \rightarrow xz
\]
We are using the symbol $\rightarrow$ for equations such that all instances of the left-hand side are greater than corresponding instances of the right-hand side, and $\leftrightarrow$, when the direction of application depends on the instance. Thus, the double-headed rule is used to rewrite any product of the form $(xy)z$ to the same term with the occurrence of $y$ replaced by a sufficiently small term in the ordering $>$, as though it were

$$y > w \mid (xy)z \rightarrow (xw)z$$

(In this example, it matters little which reduction ordering is used.) To prove validity of the identity $(xy)(wz) = (xw)(yz)$, both sides are rewritten. Applying the upper-right rule to both sides, we get $(xy)z = (xw)z$. Suppose $a$ is smaller than any other term. Then the identity reduces to the trivial equation $(xa)z = (xa)z$.

By adding to the vocabulary a new constant $a$ smaller than any term in the ordering $>$, one can reformulate the system as

$$(xa)x \rightarrow x \quad x(yz) \rightarrow xz$$

$$y \neq a \mid (xy)z \rightarrow (xa)z \quad ((xa)a)z \rightarrow xz$$

The condition $y \neq a$ insures that $(xy)z > (xa)z$. There is no need to consider other instances of the original rules, since terms to which they would apply can first be rewritten by the conditional rule.

Ordered rewriting always terminates, since the rewrite relation is contained in a reduction ordering. For confluence, we require a broader notion of critical pair than 5.14:

6.2. Definition (Ordered Critical Pair [Lankford 1975]). Given a reduction ordering $>$, if $s = t$ (or $t = s$) and $l = r$ (or $r = l$) are two (not necessarily distinct) equations (but with variables made distinct) and $\mu$ is a most general unifier of $l$ and a nonvariable subterm $s'$ of $s$, and for some substitution $\sigma$, we have $s\mu \not\leq t\mu$ and $s\sigma \not\leq s\mu[r\mu]\sigma$ in the ordering, then the equation $s\mu[r\mu] = t\mu$ (with $r\mu$ replacing $s'\mu$ in $s\mu$) is a (ordered) critical pair.

Let $\text{cp}(E)$ denote the set of all (ordered) critical pairs between equations $l = r$ in a set $E$. When $l > r$, then by definition its instances are also ordered thus. So overlapping $r$ on either side of another equation does not contribute a critical pair to $\text{cp}(E)$. Also, if $R$ is a set of rewrite rules, then $\text{cp}(E \cup R) = \text{cp}(E \cup R_\leq)$, permitting critical pairs between rules and equations.

6.3. Theorem (Ordered Critical Pair [Lankford 1975]). Let the ordered rewrite relation $\rightarrow_E$ be defined by an equational system $E$ and reduction ordering $>$. If all ground instances $l\sigma = r\sigma$ of critical pairs $l = r \in \text{cp}(E)$ have ordered rewriting proofs, $l\sigma \downarrow_E r\sigma$, then $\rightarrow_E$ is ground confluent.

Proof. As in Lemma 5.15, for any peak $s \leftarrow E \circ \rightarrow_E t$ between ground terms $s, t$, there either exists a rewrite proof $s \downarrow_E t$ or a critical-pair proof $s \leftrightarrow_{\text{cp}(E)} t$. 

Rewriting 35
In the latter event, \( s \equiv u[l] \sigma \) and \( t \equiv u[r] \sigma \) for some critical pair \( l = r \), and by assumption \( u[l] \sigma \downarrow_E u[r] \sigma \). So the relation \( \rightarrow_E \) is locally confluent, and by the Diamond Lemma the rewrite relation \( \rightarrow_E \) is confluent for ground terms.

Though there are only a finite number of critical pairs for finitely many equations, the question is how, in general, one can check whether all instances have rewrite proofs. In the previous example (6.1), the critical pair \( x = (xw)x \) (one of several between the first two rules) always reduces by another application of the first rule to \( x = x \). On the other hand, the pair \( ((xx')y)z = ((xx)w)z \) can be rewritten to \( ((xx')w)z = ((xx')w)z \) by the third rule, but only because we must have \( x > x' \) and \( y > w \) for the critical pair to arise in the first place. In this case, it is actually no problem at all, since the third rule can be used instead to show that both sides have normal form \( xz \).

In general, confluence of ordered rewriting is decidable for the wide class of precedence-based path orderings described in Section 4 [Comon, Narendran, Nieuwenhuis and Rusinowitch 1998].

Completion is used to construct convergent systems for a given set of equational axioms. The procedure maintains a set \( E \) of equations and a set \( R \) of rules, oriented according to a given reduction ordering \( > \). Equations are only used to rewrite when they cause a decrease under \( > \). These sets are manipulated by the following inference rules:

**Deduce:** Add a critical pair formed from left-sides of rules in \( R \) and both sides of equations in \( E \):

\[
\frac{E, R}{E, R, s = t} \quad \text{if } s = t \in cp(E \cup R)
\]

**Orient:** Orient an equation \( l = r \) (or \( r = l \)) for which \( l > r \):

\[
\frac{E, R, l = r}{E, R, l \rightarrow r} \quad \text{if } l > r
\]

**Delete:** Remove an equation whose sides are identical:

\[
\frac{E, R, r = r}{E, R}
\]

**Simplify:** Use a rule to rewrite either side of an equation (all equations are treated as unordered pairs) or use either a rule or equation to rewrite the right side of a rule (all rules are treated as ordered pairs):

\[
\frac{E, R, l = r}{E, R, l = r'} \quad \text{if } r \rightarrow_R r'
\]

\[
\frac{E, R, l \rightarrow r}{E, R, l \rightarrow r'} \quad \text{if } r \rightarrow_{E \cup R} r'
\]

**Collapse:** Use equations or rules to rewrite a less general term:
Rewriting

We begin with a finite set $E$ of equations and no rules in $R$. The set of equations is expanded by deduction of critical pairs (the rule deduce). Simplify, in essence, first expands the set of rules to include even though total precedences yield more powerful termination orderings.

A reduction ordering is called a complete simplification ordering if it can be extended to a complete simplification ordering.

We begin with a finite set $E$ of equations and no rules in $R$. The set of equations is expanded by deduction of critical pairs (the rule deduce). Simplify, in essence, first expands the set of rules to include $l \rightarrow r'$, since $l = r = r'$ and $l > r > r'$, and then contracts the rules by deleting $l \rightarrow r$ which, given $l \rightarrow r'$ and the fact that $r \rightarrow r'$, is redundant.

There is room for flexibility in how the inference rules are applied. One simple version of completion mixes the above inference steps according to the following strategy:

**Completion** = $(((\text{Simplify} + \text{Delete})^*; (\text{Orient}; \text{Collapse}^*))^*; \text{Deduce})^*$

In words: Simplify and delete equations as much as possible before orienting. Then use the newly oriented equation to collapse left-hand sides of all nonreduced existing rules; then, go back and simplify over again. When there are no equations left to orient, generate one new critical pair, and repeat the whole process.

Different versions of completion differ in which equations they orient first and in how they keep track of critical pairs that still need to be deduced.

For guaranteed success of completion, the ordering should be total on ground terms.

6.4. Definition (Complete Simplification Ordering [Hsiang and Rusinowitch 1987]).

A reduction ordering is called a complete simplification ordering if it is a total ordering of the ground terms (over some vocabulary). A reduction ordering is completable if it can be extended to a complete simplification ordering.

Completable simplification orderings of necessity have the subterm property, which states that terms are larger than their proper subterms. Examples include the empty ordering, the lexicographic path ordering with a partial precedence, and the numeric path ordering. Orderings, such as the recursive path ordering, enjoying the incremental property can be gradually extended to cover new rules as they are generated. This is why precedences that are partial orderings are of importance even though total precedences yield more powerful termination orderings.

An implementation of completion is fair if it does not altogether avoid processing any relevant, nonredundant critical pair. Running completion with a fair strategy can have one of three outcomes: It might converge on a finite system of only rules that is a decision procedure for the initial set of equations; it might reach a point in which all (ordered) critical pairs have been considered and all have rewrite proofs—using rules and equations; or it might loop and generate an infinite number of rules and equations. The uniqueness of reduced convergent systems (Theorem 5.21) implies that the choice of ordering determines the final result of completion.
6.5. Definition (Fairness [Bachmair and Dershowitz 1994]). An inference sequence $E_0 \vdash E_1 \vdash \cdots$ is fair with respect to completion if
\[
\mathsf{cp}(E_\infty) \subseteq E_0 \cup E_1 \cup \cdots
\]
where $E_\infty = \lim \inf_j E_j (= \cup_{i \geq 0} \cap_{j \geq 1} E_j)$ is the set of persisting equations.

The following can be shown by induction with respect to a suitable proof ordering $\gg$:

6.6. Theorem (Completeness of Completion [Bachmair and Dershowitz 1994]). For any fair completion strategy and complete simplification ordering, ordered rewriting with (finite or infinite) $E_\infty$ is convergent.

This means that eventually both sides of any identity in the theory of the initial set of equations $E_0$ will become joinable. Thus, fair completion is a complete equational theorem prover.

6.7. Example (Abelian Groups I). Completion, given
\[
\begin{align*}
x \cdot 1 &= x & x \cdot y &= y \cdot x \\
x \cdot (y \cdot z) &= (x \cdot y) \cdot z & x \cdot x^- &= 1
\end{align*}
\]
and a lexicographic path ordering (in which $- > > 1$ and left arguments are looked at first) will generate the following set of rules which constitute a decision procedure for free Abelian (commutative) groups under ordered rewriting:

\[
\begin{align*}
1^- & \rightarrow 1 & x \cdot y & \leftrightarrow y \cdot x \\
x \cdot 1 & \rightarrow x & x \cdot (y \cdot z) & \leftrightarrow y \cdot (x \cdot z) \\
1 \cdot x & \rightarrow x & (x \cdot y) \cdot z & \rightarrow x \cdot (y \cdot z) \\
(1^-) & \rightarrow x & x \cdot (x^- \cdot z) & \rightarrow z \\
x \cdot x^- & \rightarrow 1 & (y \cdot x^-)^- & \rightarrow x^- \cdot y^-
\end{align*}
\]

Those equations used in both directions have a two-headed arrow. To decide validity of an equation $s = t$, the variables in $s$ and $t$ are replaced by distinct new (Skolem) constants, obtaining a ground instance $s' = t'$ of the equation $s = t$.

The lexicographic path ordering is extended to a total ordering that includes any constants appearing in $s'$ or $t'$. The double-headed rules are then used only in a direction that reduces in this ordering. The equation is valid iff both $s'$ and $t'$ have the same normal form. The reason is that ordered rewriting using the above rules causes ground terms to associate to the left and sorts any ground terms so that small terms occur leftmost and constants and their inverses appear together.

6.8. Theorem ([Bachmair et al. 1989]). Suppose $R$ is a finite convergent system for axioms $E$ and $>$ is a completable simplification ordering for which all rules in $R$ decrease. Any fair completion strategy for $>$ will generate a finite convergent system for $E$ (not necessarily identical to $R$).
See also [Devié 1990, Bachmair and Dershowitz 1994].

If $R$ is a reduced convergent system and the strategy performs all compositions and collapses, then, by Theorem 5.21, completion will actually produce $R$.

By adding the following rule, a system will always be found if one exists for any given reduction ordering, even if the ordering is not completable to a well-ordering of ground terms:

**Double:** Create one rule to take two steps:

$$
\frac{E, R, s \to t[u], l \to r}{E, R, s \to t[u], l \to r, s\mu \to t[r]\mu} \quad \mu = \text{mgu}(l, u), \text{ nonvariable } u
$$

6.9. **Theorem** ([Bofill, Godoy, Nieuwenhuis and Rubio 1999]). Suppose $R$ is a finite convergent system for axioms $E$ and $>$ is a reduction ordering for which all rules in $R$ decrease. With any fair strategy, using the rules deduce, double, orient, delete, the set of persisting rules will subsume $R$ (that is, every equation in $R$ will reduce to triviality).

The efficiency of completion depends on the number of critical pairs deduced. Simplification (rewriting to normal form), as employed in the above procedure, is one very effective mechanism for eliminating superfluous equations: if an equation or rule can be rewritten to something previously generated, then it is not needed.

We have not specified how to choose which critical pair to consider at any given time in completion, except that it should be done fairly. Generating them in age-order is one possibility. Another idea is to overlap the two smallest equations that have not been tried together so far. This is good because it tends to produce small rules, and small rules tend to be more useful since they are likely to apply more often. It might also be useful to look for equations with relatively many (linear) variables. Using small equations is fair since there are only finitely many equations of a given size or smaller. Lescanne [1984] suggested running completion for a while, then filtering out the most interesting equations (typically small equations), adding them to the original set $R$, and repeating the process. “Filter and reuse” tends to focus attention on the more interesting equations, and can produce good results.

In order to perform completion, it is necessary to choose a reduction ordering. Since such orderings abound, this can be difficult. In [Plaisted 1986], a method is proposed to circumvent this problem. There, nondeterministic versions of completion are given; the nondeterminism has to do with how a critical pair $s = t$ is converted into a rule, whether $s \to t$ or $t \to s$ is chosen. The idea is to give a simple criterion for nontermination; for any system $R$ satisfying this criterion, there is no simplification ordering $>$ such that for all rules $l \to r$ in $R$, $l > r$. When this happens, some nondeterministic choice has to be re-done, and completion is again attempted. This method is guaranteed to be able to generate any system that could be generated using a simplification ordering to orient the critical pairs. Another approach is to test whether a member of a particular class of simplification orderings orients a given set of rules, using algorithms for existential fragments of fully-invariant orderings, mentioned in Section 4.
Various techniques have been used in practice to check for redundancy of equations, so as to limit the critical pairs that are required for completeness of completion. Suppose the proof ordering \( \succ \) has the property that a proof decreases by replacing a subproof with one involving terms that are all smaller vis-à-vis the reduction ordering \( \succ \) supplied to completion. Then a critical pair \( s = t \) is redundant if there exists an equational proof \( s \leftrightarrow u_1 \leftrightarrow \cdots \leftrightarrow u_n \leftrightarrow t \), such that term \( u \) from which the critical pair was formed is greater than each of the \( u_i \). For the Group Fragment (5.16), the critical pair \((-0) + y = y\), obtained by rewriting \((-0) + (0+y)\) with \((-x) + (x+y) \rightarrow y\) and \(0 + x \rightarrow x\), is redundant since \((-0) + (0+y)\) is greater than each of the terms in the alternative proof \((-0) + y \rightarrow 0 + y \rightarrow y\). In particular, a critical pair can be ignored if the variable part of either of the rules involved becomes reducible. Some such redundancy criteria have been suggested in [Winkler and Buchberger 1983, Bachmair and Dershowitz 1988, Kapur, Musser and Narendran 1988, Zhang and Kapur 1990]. These criteria can save some work, but experimental results suggest that most do not improve running times significantly, except in the associative-commutative case of Section 7, where it can make a dramatic difference.

We illustrate another phenomenon with the following single axiom for group theory [Higman and Neumann 1952]:

\[
x/((((x/x)/y)/z)/(((x/x)/x)/z)) \rightarrow y
\]

Here \(x/y\) is analogous to \(xy^{-1}\). During completion, the unorientable equation \(x/x = y/y\) is obtained, but it is clear that the value of \(x/x\) does not depend on \(x\), since \(x\) does not appear on the right-hand side. So, rather than using this equation wildly, we can introduce a new constant \(e\) and the definitional rule \(x/x \rightarrow e\). The same situation occurs when completing the Loop axioms. In general, if an equation \(r = s\) is derived, and at least one variable appears on only one side of this equation, then we can add rules \(r \rightarrow f(x_1 \ldots x_n)\) and \(s \rightarrow f(x_1 \ldots x_n)\) where the \(x_i\) are all the variables in common to \(r\) and \(s\) and \(f\) is a new function symbol. Knuth and Bendix [1970] suggested treating unorientable equations by introducing such new operators, but the technique often degenerates by coming up with infinitely many new operators. In some cases when completion does not terminate, patterns in the infinite set of rules generated may be observed and used to generate “meta-rules” that encode these patterns. This may permit construction of a convergent “meta-term” rewriting system; see [Kirchner and Hermann 1990].

In principle, any theory with decidable word problem can be solved by rewriting with an ordered system for some conservative extension of the theory [Dershowitz, Marcus and Tarlecki 1988]. This is not, however, true for ordinary rewriting [Kapur and Narendran 1985b].

Completion can be applied to program generation [Dershowitz and Reddy 1993]. This entails obtaining a program in the form of a set of equations by completing an axiomatic specification.
7. Relativized Rewriting

Many axioms are difficult to handle by rewriting. One example is the commutative axiom \( x + y = y + x \) which is nonterminating no matter how it is oriented as a rewrite rule. We can use ordered rewriting, as in Section 6, and restrict application of the rule to commute only in the direction that decreases the term in some given ordering. For example, we might allow \( 2 + 1 \) to be replaced by \( 1 + 2 \), but not vice versa. However, if an operator like \( + \) is associative and commutative, then there are many equivalent ways to represent terms like \( a + b + c + d \), which imposes a burden in storage and time on completion.

In the previous section, we saw (Example 6.7) how to use ordered rewriting to decide validity in free Abelian groups. An alternative approach is to apply commutativity only to enable the application of other rules. For example, we would apply a rule \( x \cdot 1 \rightarrow x \) to \( a \cdot 1 \), as well as to \( 1 \cdot a \). This “relativized” rewriting permits one to treat such axioms in a special way, without explicitly representing many equivalent forms of a term. The cost is a more complicated rewriting relation, more difficult termination proofs, and a more complicated completion procedure.

The general idea is to let individual terms stand for their \( E \)-equivalence classes, for some equational theory \( E \). For example, if \( E \) includes associative and commutative (AC) axioms for \( * \), then the terms \( (a \cdot b) \cdot a^{-1} \), \( a^{-1} \cdot (a \cdot b) \), \( a \cdot (b \cdot a^{-1}) \), etc., will all be rewritable by \( x \cdot x^{-1} \rightarrow 1 \). Usually some representation of the whole equivalence class is used; thus it is not necessary to store all the different terms in the class.

To define a rewriting relation on \( E \)-equivalence classes, recall that the relation \( s =_E t \) \((E \models s = t)\) is an equivalence relation. If \( s \) is a term, let \([s]_E\) be its \( E \)-equivalence class, containing all terms \( E \)-equivalent to \( s \). The simplest approach would be to say that \([s]_E \rightarrow [t]_E\) if \( s \rightarrow t \). Retracting this back to individual terms, we say that \( u \rightarrow_{R/E} v \) if there are terms \( s \) and \( t \) such that \( u =_E s \), \( s \rightarrow_{R} t \), and \( v =_E t \). This relation \( R/E \) is called a class rewriting system; however, \( R/E \) rewriting is difficult to compute, even when equivalence classes are finite, since it requires searching through all of \([u]_E\) for potential redexes. Note that \( E \)-equivalent rules are redundant, in this approach.

A computationally simpler idea is to consider the equivalence classes of instances of left-hand sides and rewrite \( u \rightarrow v \) if \( u \) has a subterm \( s \) such that \( s =_E \ell \sigma \) and \( l \rightarrow r \in R \) and \( v \equiv u \) with \( s \) replaced by \( r \sigma \). In this case, we write \( u \rightarrow_{E \setminus R} v \) and call the relation \( E \setminus R \) the extended rewrite system for \( R \) modulo \( E \). In the associative-commutative case, this means that \( u[s] \) rewrites to \( u[r \sigma] \) whenever \( s \) is equal under AC to an instance \( l \sigma \) of the left-hand side of some rule \( l \rightarrow r \) (that is, \( s \) and \( l \sigma \) may have arguments to AC symbols permuted). Thus, AC-matching must be used to detect applicability of extended AC rewriting rules. (Matching a left-hand side \( l \) to a term \( s \) is essentially unification of the two terms in the theory of \( E \), with the variables of \( s \) regarded as new constants that may not be instantiated.) Consider the systems \( R/E \) and \( E \setminus R \), where \( R \) has \( a \cdot b \rightarrow d \) and \( E \) is associativity and commutativity of the operator. Then \( (a \cdot c) \cdot b \rightarrow_{R/E} c \cdot d \), since \( (a \cdot c) \cdot b =_E c \cdot (a \cdot b) \). However, it is not true that \( (a \cdot c) \cdot b \rightarrow_{E \setminus R} c \cdot d \) since
there is no subterm of \((a * c) * b\) that is \(E\)-equivalent to \(a * b\). On the other hand, 
\((b * a) * c \rightarrow_{\text{E}/\text{R}} d * c\), since \(b * a =_E a * b\).

The extended rewrite relation only requires using the equational theory on the 
chosen redex \(s\) instead of the whole term to “semantically” match \(s\) with the left-hand 
side of some rule. Such \(E\)-matching is often easy enough computationally 
to make \(E\)-rewriting much more efficient than \(R/E\) rewriting, but semantic 
matching is undecidable in general [Heilbrunner and Hölldobler 1987]. Construction 
of convergent systems using this rewriting relation requires semantic unification 
algorithms (see [Baader and Snyder 2001], Chapter 8 of this Handbook). This, 
very successful approach (to the problem that Knuth left open) was initiated by 
Lankford and Ballantyne [1977] and Peterson and Stickel [1981] and generalized 
in [Jouannaud and Kirchner 1986, Bachmair and Dershowitz 1989]. It applies to 
theories with permutative axioms like commutativity, \(AC\), or \(AC\) with idempotence 
and/or identity.

It is impossible for class rewriting to be confluent in the traditional sense, if \(E\) 
equivalence classes are nontrivial, since any term \(E\)-equivalent to a normal form 
has \(R^*\) as a redex, while \(R^*\) is a normal form. Instead, to capture the property that class 
rewriting is confluent when considered as a rewrite relation on equivalence classes, 
we sat that \(R/E\) is (class) confluent if for any term \(t\), if \(t \rightarrow_{\text{R}/\text{E}}^* v\) and \(t \rightarrow_{\text{R}/\text{E}}^* v\) 
then there are \(u\) and \(v\) such that \(u \rightarrow_{\text{R}/\text{E}}^* u\), \(v \rightarrow_{\text{R}/\text{E}}^* v\), and \(u =_E v\). If \(R/E\) 
is class confluent and terminating then a term may have more than one normal form, 
but all of them will be \(E\)-equivalent. Furthermore, if \(R/E\) is class confluent 
and terminating, then we can reduce any \(R =_E E\)-equivalent terms to \(E\)-equivalent 
terms by rewriting. Then an \(E\)-equivalence procedure can be used, if there is one.

Though \(E\)-rewriting is much more efficient than \(R/E\) rewriting, \(\rightarrow_{\text{R}/\text{E}}^*\) has 
better logical properties for deciding \(R/E\) equivalence. So the theory of relativized 
rewriting is largely concerned with finding connections between these two rewriting 
relations.

7.1. Definition (Church-Rosser Modulo [Peterson and Stickel 1981]). The rewrite 
relation \(E\)-\(R\) is Church-Rosser modulo \(E\) if any two \(R =_E E\)-equivalent terms 
can be \(E\)\(R\) rewritten to \(E\)-equivalent terms.

This is not the same as saying that \(R/E\) is Church-Rosser, considered as a rewrite 
system on \(E\)-equivalence classes.

It follows that

7.2. Theorem ([Peterson and Stickel 1981]). Let \(R\) be a rewrite system and \(E\) an 
extensional system. If all of the following hold:

1. \(R\) is finite,
2. \(\rightarrow_{\text{R}/\text{E}}^*\) is terminating,
3. \(R\) is Church-Rosser modulo \(E\),
4. \(E\)-matching is solvable,

then equivalence in the combined theory of \(R =_E E\) is decidable, by normalizing with 
\(\rightarrow_{\text{E}/\text{R}}^*\).
Note that $\rightarrow_{E \setminus R}$ is a subset of $\rightarrow_{R \setminus E}$, so if $R \setminus E$ is terminating, so is $E \setminus R$. But Church-Rosser modulo $E$ is not a local property; as stated, it's not obvious that the property is decidable.

Suppose $R$ is a binary relation, $E$ is an equivalence relation, and $T$ is a binary relation (think of $\rightarrow_{E \setminus R}$) lying between $R$ and $R/E$, that is, $R \subseteq T \subseteq R/E$. We say that $T$ is Church-Rosser modulo $E$ with $R$ if any two $R \cup E$-equivalent elements can be rewritten by $T$ into $E$-equivalent elements.

7.3. DEFINITION (Local Coherence [Jouannaud and Kirchner 1986]). A binary relation $\rightarrow_T$ is locally coherent with a binary relation $R$ modulo a binary relation $E$ if $\rightarrow_T \circ \leftrightarrow_E$ is contained in $\rightarrow_T \circ \leftrightarrow_E \circ \rightarrow_T$.

The idea of coherence is that there should be some similarity in the way different elements of an $E$-equivalence class rewrite.

7.4. LEMMA (Coherence [Jouannaud and Kirchner 1986]). Let $\rightarrow_T$ be a relation such that $\rightarrow_R \subseteq \rightarrow_T \subseteq \rightarrow_{R \setminus E}$. Suppose $\rightarrow_{R \setminus E}$ is terminating. Then $\rightarrow_T$ is Church-Rosser modulo $E$ with $R$ iff $\rightarrow_T$ is locally coherent modulo $E$ with both $\rightarrow_R$ and $\leftrightarrow_E$.

This lemma reduces the desired Church-Rosser property to local properties that can be tested, analogous to the Diamond Lemma (5.13). It forms the basis of methods for completing $R$ and $E$ to obtain systems that are Church-Rosser modulo $E$.

Another approach is to add rules to $R$ to obtain a logically equivalent system $S/E$; that is, $R \cup E$ and $S \cup E$ have the same logical consequences (that is, are equivalent), but $E \setminus S$ rewriting is the same as $R/E$ rewriting. Therefore we can use the computationally simpler $E \setminus S$ rewriting to decide the equality theory of $R/E$. This is done for associative-commutative operators by Peterson and Stickel [1981]. In this case, confluence can be decided by methods simpler than those above. Consider the special case of rewriting relative to the associative and commutative axioms $E = \{ f(x, y) = f(y, x), f(f(x, y), z) = f(x, f(y, z)) \}$ for a function symbol $f$. In this case, one can flatten the term structure so that $E \setminus R$ rewriting can be used rather than $R/E$ rewriting, that is, a term $f(s_1, f(s_2, \ldots, f(s_{n-1}, s_n) \ldots))$, where none of the $s_i$ have $f$ as outermost function symbol, is represented as $f(s_1, s_2, \ldots, s_n)$, where $f$ is now a varyadic symbol, taking a variable number of arguments. For such flattened terms, all permutations of arguments of $f$ are identified. This means that each $AC$-equivalence class is represented by one flat term. Since now all members of a given $AC$-equivalence class have the same term structure, $AC \setminus R$ rewriting is easier. However, the subterm structure has been changed; $f(s_1, s_2)$ is a subterm of $f(f(s_1, s_2), s_3)$ but there is no corresponding subterm of $f(s_1, s_2, s_3)$. Thus, $AC \setminus R$ rewriting does not simulate $R/AC \setminus R$ rewriting on the original system. For example, consider $R = \{ a * b \rightarrow d \}$ with AC axioms for $\ast$. Suppose $s \equiv (a * b) * c$ and $t$ is $d * c$. Then $s \rightarrow_{R/AC} t$; in fact, $s \rightarrow_{AC \setminus R} t$. However, if we flatten the terms, then $s$ becomes $\ast(a, b, c)$ and $s$ no longer rewrites to $t$ since the subterm $\ast(a, b)$ has disappeared. To overcome this, we must add extensions to rewrite rules to simulate their
effect on flattened terms. The extension of the rule \(a\ast b \rightarrow d\) is \(x\ast a\ast b \rightarrow x\ast d\), where \(x\) is a new variable. With this extended rule, we do have that \(a\ast b \ast c \rightarrow_{AC}\ d\ast c\).

The general idea, then, is to flatten terms, and extend \(R\) by adding extensions of rewrite rules to it. Then, extended rewriting on flattened terms using the extended \(R\) is equivalent to class rewriting on the original \(R\). Formally, suppose \(s\) and \(t\) are terms and \(s'\) and \(t'\) are their flattened forms. Suppose \(R\) is a rewrite system and \(S\) is \(R\) with the extensions added. Suppose \(E\) is associativity and commutativity. Then \(s \rightarrow_{R/E} t\) iff \(s' \rightarrow_{S/E} t'\). The extended \(R\) is obtained by adding, for each rule of the form \(f(r_1, \ldots, r_n) \rightarrow s\) where \(f\) is \(AC\), an extended rule of the form \(f(x, r_1, \ldots, r_n) \rightarrow f(x, s)\), where \(x\) is a new variable. The original rule is also retained. This idea does not always work on other equational theories, however. Note that some kind of \(AC\) matching is needed for extended rewriting. This can be fairly expensive, since there are so many permutations to consider, but it is fairly straightforward to implement. Completion relative to \(AC\) can be done with the flattened representation [Peterson and Stickel 1981], using \(AC\) matching and unification. Unification is needed to form critical pairs \(\mathbf{cp}_{AC}(E)\) modulo \(AC\). This means that we look at the finite set of most general substitutions that allow one side of an equation to be \(AC\)-equivalent to a nonvariable subterm of one side of another equation, and get a critical pair for each such ambiguously rewritable term.

Specifically, the completion procedure of Section 6 may be modified in the following ways to handle \(AC\) symbols:

1. \(AC\)-unification is used to generate critical pairs instead of ordinary unification:

\[
\frac{E, R}{E, R, s = t} \quad \text{if } s = t \in \mathbf{cp}_{AC}(E \cup R)
\]

2. An additional expansion operation is needed whenever a new equation \(f(s, t) = r\) is formed, where \(f\) is an \(AC\) symbol:

\[
\frac{E, R, f(s, t) = r}{E, R, f(s, t) = r, f(s, f(t, z)) = f(r, z)} \quad \text{if } f \text{ is AC, } f(s, t) \not\in r
\]

for some new variable \(z\). This extension rule ensures that \(f(s, t) = r\) can be used even when rearrangement is needed to get an instance of its left-hand side.

3. The ordering must be such that \(s > t\) only if \(s' > t'\) for all \(AC\) variants \(s'\) of \(s\) and \(t'\) of \(t\). This ensures that each new \(AC\)-rule reduces the term it is applied to:

\[
\frac{E, R, l = r}{E, R, l \rightarrow r} \quad \text{if } l > r
\]

4. Any equation between \(AC\)-variants is deleted:

\[
\frac{E, R, s = t}{E, R} \quad \text{if } s =_{AC} t
\]

5. \(AC\)-rewriting is used for simplification:
Recall that the Chameleons (Example 1.3) do not terminate. But if we turn any one of the rules around, they do! Though the result is not confluent, AC completion can be used to generate a confluent system of rules and equations: Start off with the rules as unoriented equations, and use an ordering with $R > Y > G$. The oriented equations are:

\[
\begin{align*}
R Y & \rightarrow G G \\
R R & \rightarrow G Y \\
R G & \rightarrow Y Y 
\end{align*}
\]

Notice how the second rule acts contrary to the “real” chameleons. The flattened, extended rules are:

\[
\begin{align*}
Y R z & \rightarrow G G z \\
R R z & \rightarrow G Y z \\
G R z & \rightarrow Y Y z 
\end{align*}
\]

and their commutative variants. The $GR$ and extended $Y R$ rules produce a critical pair $Y Y Y = G G G$, which gets oriented from left to right: $Y Y Y \rightarrow G G G$. The complete system reduces the initial state and the monochrome states to distinct normal forms. (Which?) Since the system is Church-Rosser, there is no way to get from the initial arrangement of chameleons to one in which they are of uniform color, no matter which way any of the rules are used. \(^7\)

7.5. Example (Abelian Groups II). The AC-completion procedure, given

\[
x \cdot 1 = x \quad x \cdot x^{-} = 1
\]

where $\cdot$ is AC, and a polynomial ordering in which $\tau(x \cdot y) = \tau(x) + \tau(y) + 1$, $\tau(x^{-}) = 2\tau(x)$, and $\tau(1) = 1$, generates the following decision procedure for Abelian groups:

\[
\begin{align*}
1^{-} & \rightarrow 1 \\
(x^{-}) & \rightarrow x \\
(y \cdot x)^{-} & \rightarrow x^{-} \cdot y^{-} \\
x \cdot x^{-} & \rightarrow 1 \\
x \cdot (x^{-} \cdot z) & \rightarrow z
\end{align*}
\]

\(^7\)The rewriting solution to the Chameleon puzzle is due to Claude Marché.
The last rule is a composed version of the extension \( x \cdot (x^- \cdot z) \rightarrow 1 \cdot z \) of \( x \cdot x^- \rightarrow 1 \). This extended rule, together with \( (y_0 \cdot x_0)^- \rightarrow x_0 \cdot y_0 \), forms a critical pair

\[
(x_2 \cdot z_1)^- (x_2 \cdot (x_1 \cdot x_2)^- \cdot z_2)^- = (z_1 \cdot z_2)^-
\]

via AC-unifier \( \{ x \mapsto x_1 \cdot x_2, z \mapsto z_1 \cdot z_2, y_0 \mapsto x_1 \cdot z_1, x_0 \mapsto x_2 \cdot x^- \cdot z_2 \} \). Both sides reduce to \( z_2^+ \cdot z_1^- \). With this system, both sides of any identity reduce by AC-rewriting to AC-equivalent terms.

The Grecian Urn (1.2) may also be viewed as an AC-rewriting system. Beans are rearranged until the pair to be removed are adjacent. With extended rules this system is confluent and terminating. For instance, \( B W \rightarrow W \) and the extended rule \( W W z \rightarrow B z \), rewrite a \( B W W \) bean arrangement to \( W W \) and \( B B \), respectively, both of which rewrite to \( B \). This means that the normal form is independent of the order in which rules are applied. If there are an even number of white beans, they can be paired and reduced to black, and then all the black beans reduce to one; if there are an odd number, the leftover white bean swallows all remaining blacks. Thus rewriting solves the problem since the color of the last bean is determined by the normal form of the rewriting relation, which does not depend on which rewrite rules are chosen.

7.6. Example (Ring Idempotents). The standard ring axioms plus

\[
\begin{align*}
aa &= a & \bb &= b & \cc &= c \\
(a + b + c)(a + b + c) &= a + b + c
\end{align*}
\]

can be completed, with an appropriate ordering, to a convergent AC system that includes the equations

\[
\begin{align*}
ba &= -ab - bc - cb - ac - ca \\
bca &= abc + acb + cab + cac + cbc + ab + ab + ac + ac + cb + cb + bc + ca
\end{align*}
\]

The normal forms of this system include (besides inverses and sums) all monomials not containing \( aa, bb, cc, ba, \) or \( bca \) (the order of factors matters since multiplication is not commutative).

Lankford and Ballantyne [1977] extended completion to handle the most important nonterminating axioms and found the above associative-commutative systems for free Abelian groups; Peterson and Stickel [1981] used a similar process to derive convergent systems for free commutative rings with unit and distributive lattices (and tried Boolean algebra without success). Hullot [1980] used these procedures to derive systems for various quasigroups, associative and non-associative rings (but could not handle anticommutative or Lie rings), left and right \( A \)-modules, \( A \)-bimodules, \( A \)-rings, and \( A \)-algebras.

An alternative approach in which \( E \) is only used to check “semantic equality” of subterms matching different occurrences of the same non-left-linear rule variable
was proposed by Pedersen [1985a]. In general, (infinitely) many more rules may be needed to complete the system. The more general issue of deduction modulo a non-equational theory \( E \) is raised in [Dowes, Hardin and Kirchner 1994].

Termination for relativized rewrite systems is tricky to test for; one needs special termination orderings. The problem is that \( E \)-equivalent terms are identified when doing relativized rewriting, so that it pays to make all \( E \)-equivalent terms equivalent in the quasi-ordering. This is problematic for the recursive path ordering and similar orderings. For example, we can represent associative-commutative equivalence classes by flattened terms, as mentioned above. However, applying the multiset path ordering to such terms violates monotonicity. For example, suppose \( \times > + \) and \( \times \) is associative-commutative. Then \( x \times (y + z) > x \times y + x \times z \). By monotonicity, we should have \( u \times x \times (y + z) > u \times (x \times y + x \times z) \). But, in fact, the term on the right is larger in the multiset path ordering. A number of attempts have been made to overcome this, starting with the “associative path ordering” of Dershowitz, Hsiang, Josephson and Plaisted [1983]. This ordering applied to transformed terms, in which big operators like \( \times \) were pushed below small operators like \( + \). More recently, better orderings have been devised [Bachmair and Plaisted 1985, Rubio and Nieuwenhuis 1993, Kapur and Sivakumar 1997, Rubio 1999].

8. Equational Theorem Proving

Huet [1981] showed how the completion procedure serves as a semidecision procedure for validity in equational theories, as long as no unorientable simplified critical pair is generated. Lankford [1975] proposed that completion-like methods be incorporated in resolution-based theorem provers for the first-order predicate calculus, with paramodulation used for unorientable equations.

Indeed, it follows from the completeness of the completion process (Theorem 6.6) that a rewrite proof between two ground terms will eventually be generated by ordered completion from a theory iff the terms are equal in the theory. Ordered completion uses paramodulation to avoid the possibility of failure inherent in the original [Knuth and Bendix 1970, Huet 1981] procedures. Thus, the uniform word problem in arbitrary equational theories can always be semidecided by running ordered completion using a completable simplification ordering:

**8.1. Theorem ([Hsiang and Rusinowitch 1991, Bachmair and Dershowitz 1994]).** Suppose \( s = t \) is a theorem of \( E \). For any fair strategy using the completion rules, starting with \( E \), and a completable simplification ordering, eventually \( s \) and \( t \) will rewrite to the identical term using the generated rules and equations.

With an empty ordering, completion amounts to ordinary paramodulation of unit equations; with more of an ordering, completion can be more effective—by reducing the number of allowed inferences and increasing the amount of simplification that may be performed without loss of completeness. The Gröbner basis approach to properties of polynomial ideals and solving word problems [Becker and Weispfenning 1993] is similar to completion.
8.2. Example (Distributive Lattices). With axioms:

\[
\begin{align*}
  x \land x & = x & x \lor x & = x \\
  x \land y & = y \land x & x \lor y & = y \lor x \\
  (x \land y) \land z & = x \land (y \land z) & (x \lor y) \lor z & = x \lor (y \lor z) \\
  (x \lor y) \land x & = x & (x \land y) \lor x & = x \\
  x \lor (y \land z) & = (x \land y) \lor (x \lor z)
\end{align*}
\]

and a lexicographic path ordering that makes meet bigger than join, completion eventually generates:

\[x \land (y \lor z) \rightarrow (x \lor y) \land (x \lor z)\]

Simplifying by ordered rewriting (ordered simplification) has been implemented (e.g. [McCune 1989]). Its advantage is that it is not necessary to use a special unification algorithm for associative and commutative functions. A disadvantage is that it is often necessary to keep more terms than if an associative-commutative unification algorithm were used. The idea of ordered simplification was taken further, to “constrained completion”, in [Kirchner, Kirchner and Rusinowitch 1990, Martin and Nipkow 1990, Peterson 1990]. The idea is to constrain rewrite rules by inequalities between (substitution instances of) variables. Thus we can have a rule \(x \cdot y \rightarrow y \cdot x\) with the constraint that \(x > y\) in the given ordering. They present methods for showing that such constrained rewriting systems are ground confluent.

8.3. Example (Bags). The following system normalizes any ground term over the equational theory containing the axioms of associativity and commutativity by performing ordered simplification with the lexicographic path ordering:

\[
\begin{align*}
  (xy)z & \rightarrow x(yz) \\
  xy & \leftrightarrow yx \\
  x(yz) & \leftrightarrow y(xz)
\end{align*}
\]

Using this system, the term \((bc)a\) will rewrite to \(b(ca)\) using \((xy)z \rightarrow x(yz)\), then to \(b(ac)\) using ordered simplification and the equation \(xy = yx\), then to \(a(bc)\) using the equation \(x(yz) = y(xz)\) and ordered simplification.

Other systems like this that are convergent with respect to ordered simplification are given in [Martin and Nipkow 1990].

Lescanne [1984] experimented with various presentations of groups: applying Knuth’s idea of handling unorientable equations by introducing new operators to Higman and Neumann’s one-equation presentation, completion came up with definitions of identity and inverse. Pedersen [1984a, 1985a] used this technique, plus a special way of handling permutative axioms, to generate systems for some entropic groupoids, and also solved some one relation monoid word problems [Pedersen 1989]. Christian [1989] was partially successful in his work on Burnside groups and Grau’s...
ternary Boolean algebra. Foret [1988] found rewrite-based decision procedures for several systems (K, Q, T, S5) of propositional modal logic. Edelson [1990] was able to rediscover syntactic proofs for properties of regular rings.

8.4. Example (Groupoid II). The entropic groupoid [Pedersen 1984b]

\[(xy)(zw) = (xz)(yw) \quad (xy)x = x\]

has the following decision procedure for ordered simplification, where \(\ast\) is defined by the equation \(x \ast z = (xy)z\), cancelling \(y\) (on the basis of the inferred equation \((xy)z = (xy')z)\):

\[
\begin{align*}
  x \ast x & \rightarrow x \\
  (xy)z & \rightarrow x \ast z \\
  x(yz) & \rightarrow xz \\
  (x \ast y)z & \rightarrow x \ast z \\
  (xy) \ast z & \rightarrow xz \\
  (xy)(zw) & \leftrightarrow (xz)(yw)
\end{align*}
\]

An absorbing rule is one in which every nonvariable subterm on the left contains all the variables [Pedersen 1985b], ensuring that its critical pairs with ground equations are still ground. With associative-commutative systems this is essentially never the case, but sometimes the nonground pairs are certain to reduce to ground ones. Ballantyne and Lankford [1981] accordingly gave rewriting-based procedures for the uniform word problem in finitely-generated commutative semigroups; with Butler [1984], they did the same for finitely-generated Abelian groups; and Lankford [1980] used this method to show decidability of the uniform word problem in finitely-presented J-algebras. Pedersen [1985b] gave sufficient syntactic conditions for decidability of the uniform word problem for some absorbing systems, including finitely-presented loops, using a completion procedure that adds new symbols as needed. Even when the uniform word problem is undecidable, as for non-Abelian groups, this method frequently finds decision procedures for specific word problems. In this vein, Le Chenadec [1985] did substantial work on finitely-presented groups from topology and geometry, including the Coxeter groups (showing termination was sometimes difficult).

Burris and Lawrence [1991] presented systems for finite fields, rings with \(x^n = x\), and such rings with \(n\) prime and \(nx = 0\) (studied also by Nipkow [1990]). Kapur and Zhang [1989] used an enhanced associative-commutative completion procedure (which avoids many redundant critical pairs) to prove commutativity for rings with \(x^n = x\), for many specific \(n\). Anatharaman and Hsiang [1990] used a combination of ordered and associative-commutative completion to derive purely syntactic proofs of the Moufang identities for alternative rings. Pedersen [1985a] used a variant of associative-commutative completion to generate an infinite system that “computes” Whitman normal form for lattices. Finally, the Robbins algebra conjecture has recently been proved using rewriting techniques [McCune 1997]. The 12-step equational proof followed after 50,000 equations were generated and 6,000
simplification steps were performed; almost 90% of the computer’s effort went into simplification. Simplification was critical for the success of the automated proof.

As first suggested in [Hsiang and Dershowitz 1983], one can apply completion to full first-order theorem proving, by using Boolean rings [Zhegalkin 1927, Stone 1936] to represent formulae:

\[
\begin{align*}
x \cdot T & \rightarrow x \\
x \cdot F & \rightarrow F \\
(x \oplus y) \cdot z & \rightarrow (x \cdot z) \oplus (y \cdot z) \\
x \cdot x & \rightarrow x \\
x \cdot x \cdot y & \rightarrow x \cdot y \\
x \oplus x & \rightarrow F \\
x \oplus x \oplus y & \rightarrow y
\end{align*}
\]

where \( \cdot \) is “conjunction”, \( \oplus \) is “exclusive or”, and both are associative and commutative. With this system, all propositional tautologies reduce to \( T \) and contradictions to \( F \). To prove validity of a formula, one Skolemizes its negation (and renames variables) to obtain a universally quantified formula, and expresses it using the above connectives. Then, were the original formula true, there would be an equational proof of the contradiction \( T = F \). Completion can be used to discover such a proof.

As a very simple example, consider the theorem

\[
[\exists xp(x) \land \forall x (p(x) \Rightarrow p(f(x)))] \Rightarrow \exists xp(f(x))
\]

Its Skolemized negation is equivalent to the following equations:

\[
\begin{align*}
p(f(f(x))) &= F \\
p(a) &= T \\
p(x) \lor p(f(x)) &= p(f(x))
\end{align*}
\]

where \( a \) is a Skolem constant. These equations entail the contradiction:

\[
T = T \lor [p(f(a)) \lor p(f(f(a)))]
\]

\[
= p(a) \lor [p(f(a)) \lor p(f(f(a)))]
\]

\[
= [p(a) \lor p(f(a))] \lor p(f(f(a))]
\]

\[
= p(f(a)) \lor p(f(f(a))]
\]

\[
= p(f(f(a))) = F
\]

This method can be refined [Hsiang 1985] to ignore many critical pairs, but one must take care to ensure completeness. Further work along these lines includes [Bachmair and Dershowitz 1987, Zhang 1994, Kapur and Narendran 1985a], see also [Bachmair and Ganzinger 2001] (Chapter 77 of this Handbook). For the use of Boolean rings in the propositional case, see [Hsiang and Huang 1996, Dershowitz, Hsiang and Shieng 2000].

There is a subtlety in the notion of logical consequence which should be mentioned here. A first-order structure \( M \) may include elements that are inaccessible, and cannot be represented by any ground term. For example, if \( E \) includes the constant 0 and unary function symbol \( s \), then the ground terms are...
\{0, s(0), s(s(0)), \ldots\}. However, the domain of the structure \( M \) may include elements \( d \) that cannot be represented by any of these elements; formally, it may be that \( d \neq 0^M, d \neq s(0)^M, d \neq s(s(0))^M, \ldots \). This means that the notion of logical consequence may not agree with naïve intuition, where we wish to infer those equations \( s = t \) such that whenever the variables of \( s \) and \( t \) are replaced by ground terms in a systematic way, the result is a consequence of \( E \). We are often interested in those equations \( s = t \) such that all ground instances are logical consequences of \( E \). For example, suppose \( E \) contains two axioms: \( x + 0 = x \), and \( x + s(y) = s(x + y) \). These equations completely define the usual addition operation on nonnegative integers, regarding \( s \) as the successor function. We expect addition to be commutative, which means that we expect to have \( E \models u + v = v + u \). Indeed, any ground instance of this equation containing only the function symbols \( s \) and \( 0 \) is a logical consequence of \( E \). However, the equation \( u + v = v + u \) itself is not a logical consequence of \( E \), since it is possible to construct a structure \( M \), containing inaccessible elements, in which \( E \) is valid but the equation \( u + v = v + u \) is not. The inductive theory of a set \( E \) of equations is the set of equations \( s = t \) such that all ground instances \( s \sigma = t \sigma \) are logical consequences of \( E \). These two concepts are related as follows: Consider the term algebra \( T(\Sigma, V) \) where \( V \) is a countable number of variables. Then \( T(\Sigma, V)/=_{E} \models s = t \) iff \( s = t \) is a logical consequence of \( E \). And \( T(\Sigma)/=_{E} \models s = t \) iff \( s = t \) is in the inductive theory of \( E \). (\( T(\Sigma, V)/=_{E} \) is a free algebra in the class of models of \( E \) and \( T(\Sigma)/=_{E} \) is the initial algebra in this class.)

In fact, by Gödel’s First Incompleteness Theorem, there can be no method of deriving all the quantified formulæ of arithmetic (addition, multiplication, and inequality) that are true of the nonnegative integers. This means that every set of axioms for the integers has a nonstandard model, containing inaccessible elements, in which some inductive theorem is false in the model. Another way of looking at this is that it is impossible in a finite axiom system to fully characterize the integers, or the set of finite terms. Nevertheless, there are incomplete methods for deriving and verifying equations in an inductive theory, which involve some form of mathematical induction. A modified completion is an integral part of many of the inductionless induction theorem proving methods [Comon 2001], see Chapter 14 of this Handbook.

Oftentimes one is interested in the satisfiability of equations. We may want to know if \( E \models (\exists x) x = t \), that is, from \( E \) does it follow that there is an \( x \) such that \( s = t \) ? This topic is covered in [Baader and Snyder 2001] (Chapter 8 of this Handbook), including the rewriting-based approach called “narrowing”.

9. Conditional Rewriting

Sometimes equational systems are not expressive enough. For example, we may want to state that \( s = t \) if some condition \( C \) is true, as in

\[
\frac{x}{y} = \frac{x - y}{y} + 1 \quad \text{only if } y \neq 0
\]
In the many cases where specifications are naturally conditional, we cannot use conventional term-rewriting systems. Instead, we use conditional rewrite systems, in which the rewrite rules have conditions attached, which must hold true for the rewrite to transpire. A rule \( l \rightarrow r \) with a condition \( C \) is written \( C \mid l \rightarrow r \), as in

\[
y \neq 0 \mid \frac{x}{y} \rightarrow \frac{x - y}{y} + 1
\]

where \( C \) is viewed as a “guard” for the application of the rewrite. Conditions \( C \) may take several different forms; for example, they may be logical formulae, or equations, or inequations. If \( C \) is a logical formula, the meaning is that a term encompassing \( l \) rewrites to the corresponding term with the appropriate instance of \( r \) in place of \( l \) only if \( C \) is true. That leaves the question of how to determine whether \( C \) is true.

In this chapter, we will assume that \( C \) is a conjunction of equations, omitting a condition when it is the empty conjunction. For example,

\[
\begin{align*}
(x > y) = T & \mid \max(x, y) \rightarrow x \\
(x > y) = F & \mid \max(x, y) \rightarrow y \\
& \mid s(x) > s(y) \rightarrow x > y \\
& \mid s(x) > 0 \rightarrow T \\
& \mid 0 > x \rightarrow F
\end{align*}
\]

define maximum and greater-than for tally numbers \( s'(0) \). The equations in the conditions can be evaluated recursively by conditional rewriting, in a manner similar to the goal-subgoal structure of Prolog, making conditional rewrite systems an attractive combination of logic and functional programming paradigms.

A conditional equation (or equational Horn clause) is an implication of the form

\[
u_1 = v_1 \land \cdots \land u_n = v_n \Rightarrow l = r
\]

In a rule of form \( C \mid l \rightarrow r \), we call \( C \) the premiss and \( l \rightarrow r \) the conclusion. The semantics is that \( C \Rightarrow l = r \). In general, \( C \) may be any formula, and may contain the equality predicate, or the rewrite relation \( \rightarrow \), or its variants. If \( C \) is a conjunction of equations, we call it a semi-equational rule; if \( C \) is a conjunction of statements of the form \( s_i \downarrow t_i \), we call it a join rule.

A (standard or join) conditional rewrite system is a collection of rules of the form

\[
u_1 \downarrow v_1 \land \cdots \land u_n \downarrow v_n \mid l \rightarrow r
\]

intending that terms \( u[\sigma] \), containing an instance of left-hand side \( l \), rewrite to \( u[\tau\sigma] \) only when all the conditions \( u_i[\sigma] \downarrow v_i[\sigma] \) hold for that substitution \( \sigma \). The most popular operational semantics for such a system require both sides of each condition to rewrite to the same normal form before an instance of \( l \) may be rewritten.
9.1. Example (Conditional Append). To feel the flavor of this version of rewriting, consider the system

\[
\begin{align*}
\text{null}(e) & \rightarrow T \quad \text{head}(x:y) \rightarrow x \\
\text{null}(x:y) & \rightarrow F \quad \text{tail}(x:y) \rightarrow y \\
\text{append}(e,y) & \rightarrow y \\
\text{null}(x) \downarrow F \mid \text{append}(x,y) & \rightarrow \text{head}(x) : \text{append}(\text{tail}(x),y)
\end{align*}
\]

and its derivation

\[
\text{append}(a :(b: \epsilon), c: \epsilon) \rightarrow^* a :(b : (c : \epsilon))
\]

Confluence of conditional systems in general is undecidable [Brand, Darringer and Joyner 1979, Kaplan 1987], unlike the unconditional case. Other approaches to conditional confluence include using oriented conditions, an analogue to orthogonality (Theorem 10.10 below), or restricted-depth proofs of conditions [Dershowitz, Okada and Sivakumar 1987, Giovannetti and Moiso 1987].

We say a conditional system \( R \) is Noetherian if there are no infinite sequences \( t_1, t_2, t_3, \ldots \) of terms such that \( t_1 \rightarrow_R t_2 \rightarrow_R t_3 \ldots \). This corresponds to the concept of termination for unconditional systems, but each rewrite step involves the recursive evaluation of conditions. That is, to determine whether \( s \rightarrow_R t \), it may be necessary to determine whether \( u \rightarrow_R v \) for \( u \) and \( v \) obtained from the condition of some rule of \( R \). It is possible for a system to be Noetherian even when such recursive evaluation of conditions does not terminate; in such a case, the computation might not be terminating. In fact, it may happen that the rewrite relation \( s \rightarrow_R t \) is undecidable.

Semi-equational systems are confluent if they are Noetherian and all critical pairs are joinable [Dershowitz and Plaisted 1988]. In contradistinction with the unconditional case, only under certain circumstances are Noetherian join systems confluent—even if all their critical pairs are joinable [Dershowitz et al. 1987]. But we first need a suitable notion of critical pair:

9.2. Definition (Conditional Critical Pair). Let \( E \) be a conditional equational system and \( \geq \) be a reduction ordering. The conditional equation \( c\mu \land p\mu \Rightarrow s\mu[r\mu] = t\mu \) is a conditional critical pair of \( E \) if \( c \Rightarrow l = r \) and \( p \Rightarrow s = t \) are conditional equations of \( E \), \( l \) unifies via most general unifier \( \mu \) with a nonvariable subterm of \( s \), \( l\mu \not\subset c\mu \), \( s\mu \not\subset p\mu \), \( l\mu \not\subset r\mu \), \( s\mu \not\subset t\mu \), and \( c\mu \land p\mu \) is satisfiable in \( E \).

A conditional critical pair \( p \Rightarrow s = t \) is joinable if \( s\sigma \) and \( t\sigma \) are joinable for all \( \sigma \) satisfying \( p \).

By \( l\mu \not\subset c\mu \), we mean that both sides of each condition in \( c\mu \) are not always greater than \( l\mu \). We use \( \text{cp}(E) \) to denote the set of conditional critical pairs so defined.

If \( c|l \rightarrow r \) and \( p|s \rightarrow t \) are two different rules and \( l \) and \( s \) unify with most general unifier \( \sigma \), then the critical pair \( p\sigma \land c\sigma \Rightarrow r\sigma = t\sigma \) is an overlay for those rules.
9.3. THEOREM ([Dershowitz, Okada and Sivakumar 1988]). A standard Noetherian conditional rewrite system is confluent if no left-hand side unifies with a nonvariable proper subterm of a left-hand side and every overlay critical pair is joinable.

Such systems with only top-level critical pairs are called overlaying. See also [Gramlich and Wirth 1996].

9.4. DEFINITION (Decreasing System). A conditional rewrite system \( R \) is decreasing if there exists a well-founded ordering \( > \) containing the rewrite relation \( \rightarrow_R \) and which satisfies two additional requirements:

1. \( > \) has the subterm property \( f(\ldots,s,\ldots) > s \), and
2. \( s\sigma > c\sigma \) for each rule \( c \vdash l \rightarrow r \) in \( R \) and substitution \( \sigma \).

Decreasing systems exactly capture the finiteness of recursive evaluation of terms; they refine the suggestion in [Kaplan 1987]. If \( R \) is decreasing, then the rewriting relation is decidable, since evaluating the conditions involves terms smaller than the redex. It’s possible for a conditional rewrite relation to be Noetherian without being decreasing. In practice, simplification orderings \( > \) may be used to show the decreasing property. This definition implies that all variables in the premiss appear also in the left-hand side. The notion needs to be extended, therefore, to cover systems (important in logic programming) with variables in conditions that do not also appear in the left-hand side. See [Bertling and Ganzinger 1989, Hanus 1995, Marchiori 1996, Ohlebusch 1999].

Standard join systems are locally confluent if they are decreasing and all critical pairs are joinable.

9.5. THEOREM ([Dershowitz, Okada and Sivakumar 1988]). A decreasing system is confluent (hence, convergent) if there is a rewrite proof of \( s\sigma = t\sigma \) for each critical pair \( c \Rightarrow s = t \) and substitution \( \sigma \) such that \( c\sigma \) holds.

Conditional Append (9.1) and the Stack system (1.7) are decreasing. Both have one trivially joinable critical pair for which there is no satisfying substitution.

Unlike the situation for completion of ordinary rewrite systems, there is no general-purpose mechanism for obtaining confluent conditional systems. There are some completion-like procedures for conditional equations [Kaplan 1987, Ganzinger 1991, Konnalis and Rusinowitch 1988], which could in some instances provide decision procedures for conditional equational theories (quasi-varieties), but, in practice, they do not work well and further research is required.

The free (initial) algebra for \( n \)-generators can be computed by completing the axioms and looking at the normal forms of bigger and bigger words until they stabilize. Pedersen [1988] did this for bands with up to three generators. Conditional rewriting, using conditional equations, can sometimes be used to capture an infinite number of unconditional rules in one conditional one; this approach was taken by Siekmann and Szabo [1982] for bands. For a conditional-rewriting approach to associative-commutative-idempotent systems, see [Baird, Peterson and Wilkerson 1989].
In order to obtain decision procedures employing conditional rewriting, we need to consider logical strengths of systems. The reason for this is as follow: Suppose we are given an equational system $E$ (with equational conditions). We want to obtain an equivalent convergent system which can be used to decide the equality theory of $E$. One way to do this is to convert $E$ into a join system $R$, and then complete $R$ to obtain $R'$ which is decreasing, Noetherian, and confluent. Then we know that $R \models s = t$ iff $R' \models s = t$ by the way completion is done, and $R' \models s = t$ iff $s$ and $t$ have the same $R'$-normal form, since $R'$ is Noetherian and confluent. However, we do not know that $R' \models s = t$ iff $E \models s = t$, which is what we really want. The reason is that we changed the conditions in $E$ to join assertions to obtain $R$. Therefore, the following theorem is useful:

9.6. Theorem ([Dershowitz, Okada, and Sivakumar 1988]). Let $R$ be a confluent and Noetherian standard conditional system, $R'$, the corresponding semi-equational systems (with conditions changed to equations), and $E$ the corresponding equational system (with right-hand sides changed to equations). Then the following are equivalent:

1. $E \models u = v$
2. $u$ and $v$ have the same $R$-normal form
3. $u$ and $v$ have the same $R'$-normal form.

If the conditions of the theorem hold, then reduction via $R$ can be used to decide the theory of $E$. Note that typically $E$, when converted to $R$, will not be confluent and Noetherian. Therefore, we have to complete $R$ in a way that does not change the corresponding equational theory $E$.

Completion for conditional systems is trickier than for unconditional systems. The inference rules we present may be classified into three expansion rules and four contraction rules. Contraction rules significantly reduce space requirements, but make proofs of completeness much more subtle. As with ordinary completion, a reduction ordering $\succ$ is used to control simplification (demodulation).

Superposition (that is, oriented paramodulation of positive equational literals) is performed only at nonvariable positions:

$$\text{Superpose:} \quad \frac{E}{E', e} \quad \text{if } e \in \text{cp}(E)$$

Only positive equations are used in this rule, and only in a decreasing direction. Either side of an equation may be used for superposition, but only if, in the context of the paramodulation, it is potentially the largest term involved. Note that the two conditional equations may actually be the same (except for renaming).

We need, additionally, a rule that paramodulates into maximal negative literals:

$$\text{Narrow:} \quad \frac{E, \: q \land s[l'] = t \Rightarrow u = v}{E, \: q \land s[l'] = t \Rightarrow u = v, \: p \mu \land q \mu \land s[\mu] = t' \mu \Rightarrow u \mu = v \mu} \quad \text{if } \begin{cases} p \Rightarrow l = r \in E, \\ l' \text{ is not a variable} \\ \mu = \text{mgu}(l, l') \\ s \mu \neq p \mu, q \mu, t \mu, s[\mu] \end{cases}$$
The condition $sv \neq pv$ means that $sv$ is not smaller than any side of any equation in $pv$. Whenever this or subsequent rules refer to a conditional equation like $q \land s = t \Rightarrow u = v$, the intent is that $s = t$ is any one of the conditions and $u$ is either side of the implied equation.

The last expansion rule in effect resolves a maximal negative literal with reflexivity of equals ($x = x$):

\[
\text{Reflect:} \quad \frac{E, \ q \land s = t \Rightarrow u = v}{E, \ q \land s = t \Rightarrow u = v, \ \sigma = \text{mgu}(s,t) \quad \text{if} \ \sigma \neq q\sigma, \ q\sigma \Rightarrow u\sigma = v\sigma}
\]

The contraction rules all simplify the set of conditional equations:

\[
\begin{align*}
E, \ q \Rightarrow u = u & \quad E, \ q \land s = s \Rightarrow u = v \\
E, \ q \Rightarrow u = v & \quad E, \ q \land s = s \Rightarrow u = v \\
E[\sigma] \Rightarrow u = v & \Rightarrow l = r \in E, \ l\sigma > r\sigma, p\sigma, \ E \models p\sigma \\
E[\sigma] \Rightarrow u = v & \Rightarrow l = r \in E, \ l\sigma > r\sigma, p\sigma, \ E \models p\sigma \\
E, \ q \Rightarrow u[l\sigma] = v & \Rightarrow v > u[l\sigma] \lor [(u[l\sigma] = v) \land (l = r)]
\end{align*}
\]

The first contraction rule deletes trivial conditional equations. The second allows for deletion of conditions that are trivially true. The last two use decreasing instances to simplify other clauses. One rule simplifies conditions; the other (compose) applies to the equation part. In both, the original clause is replaced by a version that is logically equivalent, assuming the rest of $E$. By $u = v \Rightarrow l = r$ we mean that the larger of $u$ and $v$, say $u$, strictly encompasses the larger side of $l = r$, say $l$, or that $u = l$ but $v$ is strictly greater than $r$ under $\Rightarrow$. This allows the larger side of an equation to be simplified by a more general equation, and the smaller side to be rewritten in any case.

As before, we use the notation $E \vdash E'$ to denote one inference step, applying any of the seven rules to a set $E$ of conditional equations to obtain a new set $E'$. The inference rules are evidently sound, in that the class of provable theorems is unchanged by an inference step.

Let $\rightarrow$ be any complete simplification ordering extending the given partial ordering $\gg$. A proof of an equation $s = t$ between ground terms (any variables in $s$ and $t$ may be treated as Skolem constants) is a sequence:

\[
s = t_1 \leftrightarrow t_2 \leftrightarrow \cdots \leftrightarrow t_{m+1} = t
\]

of $m + 1$ terms ($m \geq 0$), each step $t_k \leftrightarrow t_{k+1}$ of which is either trivial ($t_{k+1} = t_k$), or else is justified by a conditional equation $e_k$ in $E$, applied at some position in $t_k$. 


a substitution $\sigma_k$ for variables in the equation, and subproofs $P_k$ (of the same form) for each conditions $u_k \sigma_k = v_k \sigma_k$ of the applied instance $v_k \sigma_k$. Steps employing an unconditional equation do not have subproofs as part of their justification. By the completeness of positive-unit resolution for Horn clauses, any equation $s = t$ that is valid for a set $E$ of conditional equations is amenable to such an equational proof.

The above inference rules are designed to allow any equational proof to be transformed into normal form. A strategy based on these rules is complete if we can show that, with enough inferences, any theorem has a normal-form proof. For the unit strategy, a normal-form proof is a valley proof having no peaks $s \leftarrow u \rightarrow t$, no steps $s \rightarrow t$ with (valley) subproofs, and no trivial steps. Normal-form proofs may be thought of as “direct” proofs; in a refutational framework the existence of such a proof for $s = t$ means that demodulation of $s$ and $t$ using positive unit equations suffices to derive a contradiction between the Skolemized negation $s' \neq t'$ of the given theorem and $x = x$.

We must demonstrate that for any proof $s \leftrightarrow t_{E_0}$, there eventually exists an unconditional valley proof $s \downarrow_{E_0} t$. In the unit strategy, only expansions involving an unconditional equation are necessary. Specifically, both equations used by superpose are unconditional and the positive literal used in narrow is a unit. Were it not for contraction rules, it would be relatively easy to show that narrow and reflect eventually provide an unconditional proof of $s = t$, and that superpose eventually turns that into a valley.

Conditional inference is fair if all persistent superpositions of unit clauses, narrowings via unit clauses, and reflections have been considered:

9.7. Definition (Unit Strategy [Dershowitz 1991a]). An inference sequence $E_0 \vdash E_1 \vdash \cdots$ is fair with respect to the unit strategy if

$$\text{cp}^1(E_\infty) \subseteq E_0 \cup E_1 \cup \cdots$$

where $E_\infty$ is the set of persisting conditional equations and $\text{cp}^1(E_\infty)$ is the set of conditional equations that may be inferred from persisting equations by one application of deduce for unconditional systems, narrow with $p$ empty, or reflect.

9.8. Theorem ([Dershowitz 1991a]). If an inference sequence $E_0 \vdash E_1 \vdash \cdots$ is fair for the unit strategy, then for any proof of $s = t$ in $E_0$, there is a normal-form proof of $s = t$ in $E_\infty$.

Proof. For the proof ordering, the term ordering $>$ is first extended to the transitive closure of it together with the proper subterm ordering $>_\text{sub}$, which is still well-founded [Dershowitz and Jouannaud 1990]. This in turn is extended to equations by considering the equation as a bag of two terms, and using the bag extension of this ordering. An equation is greater than a term iff one of its sides is. Conjunctions of equations are compared as bags of these bags, and a conjunction is larger than a term if one of its conjuncts is. Proofs are measured in the following way: Consider a step $s = w[\sigma] \leftrightarrow w[r \sigma] = t$ in a ground proof or its subproofs, using a conditional
equation \( q \Rightarrow t = r \), and \( s \) is the larger of \( s \) and \( t \) (in the complete simplification ordering extending \( > \)). To each such step, we assign the weight \([q\sigma, s, t\sigma], q \Rightarrow t = r\).

Steps are compared in the lexicographic ordering of these pairs. The first components of pairs are compared in the bag extension of the ordering on conjunctions and terms described above. (Note that \( s \) is always greater or equal to \( t\sigma \), and that for decreasing instances it is also greater than \( q\sigma \).) Second components are compared using the extension \( > \) of the encompassment ordering described earlier. Proofs are compared in the well-founded bag extension of the lexicographic ordering on steps.

We use \( \triangleright \) to denote this well-founded proof ordering.

One needs to show that inferences never increase the complexity of proofs and, furthermore, that there are always inferences that can decrease the complexity of nonnormal proofs. Then, by induction with respect to \( \triangleright \), the eventual existence of a normal-form proof follows: If \( E \triangleright E' \), then for any proof \( P \) in \( E \) of an equation \( s = t \), there exists a proof \( P' \) in \( E' \) of \( s = t \), such that \( P \triangleright P' \) or \( P = P' \). This is established by consideration of the effects of each contracting inference rule that deletes or replaces equations, since for expansion rules, \( E \subseteq E' \), and we can take \( P' = P \). The conditions imposed on compose are essential for showing a decrease in \( \triangleright \). Furthermore, if \( P \) is a non-normal-form proof in \( E \), then there exists a proof \( P' \) in \( E \cup \text{cp}^1(E) \) such that \( P \triangleright P' \).

If \( s = t \) is provable in \( E_0 \), then it has a proof \( P \) in the limit \( E_\infty \). If \( P \) is nonnormal, then it admits a smaller proof \( P' \) using (in addition to \( E_\infty \)) a finite number of equations in \( \text{cp}^1(E_\infty) \). By fairness, each of those equations appeared at least once along the way. Subsequent inferences can only decrease the complexity of the proof of such an equation once it appears in a set \( E_i \) (and has a one-step proof). Thus, each equation needed in \( P' \) has a proof of no greater complexity in \( E_\infty \) itself, and hence (by the bag nature of the proof measure), there is a proof of \( s = t \) in \( E_\infty \) that is strictly smaller than \( P \). Since the ordering on proofs is well-founded, by induction there must be a normal proof in \( E_\infty \).

In the above method, only unconditional equations are used for superposition and narrowing. This may be good for theorem proving purposes, but is useless from the point of view of completion. An alternative is to design an inference system that distinguishes between decreasing and nondecreasing non-unit clauses [Ganzinger 1991, Dershowitz 1991b]. The required inferences (using superpose and narrow) are a stringent restriction of paramodulation.

For the decreasing method, we redefine a normal-form proof of \( s = t \) to be a valley proof \( s \downarrow t \) in which each subproof is also in normal form and each term in a subproof is smaller than the larger of \( s \) and \( t \) [Dershowitz and Okada 1988]. Any non-normal-form proof has a peak made from decreasing instances with normal-form subproofs, or has a nondecreasing step with normal-form subproofs, or has a trivial step. Theorem 9.5 can be adapted to ground confuence of decreasing systems. Superposition is needed between decreasing conditional rules. As before, we must perform enough expansions with persistent conditional equations for there to always be a normal-form proof in the limit.
9.9. Definition (Decreasing Strategy [Dershowitz 1991a]). An inference sequence $E_0 \vdash E_1 \vdash \cdots$ is fair with respect to the decreasing strategy if

$$cp^+(E_\infty) \subseteq E_0 \cup E_1 \cup \cdots$$

where $cp^+(E_\infty)$ is the set of conditional equations that may be inferred from persisting equations by one application of an expansion rule superpose, narrow, or reflect.

Decreasingness is essentially the same condition as imposed on conditional rewrite rules by the completion-like procedures of [Kaplan 1987, Ganzinger 1991]. In these methods, superposition is used when the left-hand side is larger than the conditions; narrowing, when a condition dominates the left-hand side. As theorem provers, however, they were refutationally incomplete, since they made no provision for “unorientable” equations $s = t$ such that $s \not \approx t$ and $t \not \approx s$.

9.10. Theorem ([Dershowitz 1991a]). If an inference sequence is fair for the decreasing strategy, then for any proof of $s = t$ in the initial set $E_0$ of conditional equations, there is a normal-form proof of $s = t$ in the limit $E_\infty$.

For extensions of the ideas in this chapter to full first-order theorem proving, see [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook).

10. Programming

Rewrite systems are readily used as a programming language. If one requires of the programmer that all programs be terminating, then rewriting may be used as is to compute normal forms. With ground confluence, one is assured of their uniqueness.

Modularity is critical in the programming context. The idea of modularity is to infer properties of a combination of two rewrite systems from properties of their parts:

10.1. Theorem ([Toyama 1987]). The union of two confluent rewrite systems sharing no function symbols or constants is also confluent.

An example showing that confluence is not preserved when a constructor is shared is:

$$f(x, x) \rightarrow a \quad e \rightarrow c(e)$$
$$f(x, c(x)) \rightarrow b$$

In the combined nonterminating, non-left-linear system, $f(e, e)$ reduces both to $a$ and $b$.

10.2. Theorem ([Toyama, Klop and Barendregt 1995]). The union of two convergent left-linear rewrite systems sharing no function symbols or constants is also convergent.
For a proof, see [Marchiori 1995].

These results unfortunately do not carry over to the prevalent situation of shared constructors. One result that does is:

10.3. **Theorem** ([Gramlich 1995, Dershowitz 1995]). The union of two convergent rewrite systems sharing only constructor symbols and all of whose critical pairs are overlays is convergent.

This is because innermost termination of such systems implies termination, while innermost termination is preserved by such unions ([Kurihara and Kaji 1990]).

10.4. **Definition.** We say that two rewrite systems $R$ and $S$ are *mutually-orthogonal* if there are no non-trivial critical pairs between rules of the different systems.

As a corollary of Theorem 5.15, we have:

10.5. **Theorem.** The union of two mutually-orthogonal rewrite systems is confluent if it is terminating.

Analogous to Theorem 5.23, we have:

10.6. **Theorem** ([Raoult and Vuillemin 1980]). The union of two left-linear confluent mutually-orthogonal rewrite systems is confluent.

The related study of properties of combinations of algebraic rewriting with versions of the lambda calculus began with [Breazu-Tannen and Gallier 1989].

Many programs (interpreters, for example) do not always terminate. Still, we would want to compute normal forms whenever they exist. Confluent systems have at most one normal form per input term, and orthogonal systems are confluent. The left-linearity restriction for orthogonal systems is reasonable in the programming context, since the formal parameters of procedure definitions are distinct. It is also convenient for efficiency of pattern matching. To check if a term $f(s, t)$ is an instance of a left-hand side $f(x, x)$, it is necessary to check that $s$ and $t$ are identical, which can require time proportional to the size of $s$ or $t$. (Of course, there are also cases where it is very convenient to use non-left-linear rules.)

To find the unique normal form for orthogonal systems, when it exists, one can use the following strategy for choosing the redex at which to apply a rule:

10.7. **Definition** (*Outermost Rewriting*). A rewriting step $s \rightarrow t$ is *outermost* with respect to some rewrite system if no rule applies at a symbol closer to the root symbol (in the tree representation of terms).

10.8. **Theorem** (*Outermost Normalization* [O'Donnell 1977]). For any orthogonal system, if no outermost step is perpetually ignored, the normal form—if there is one—will be reached.
Outermost rewriting of expressions is similarly used to compute normal forms in combinatory logic and head normal forms in the lambda calculus.

In this way, orthogonal systems provide a simple, pattern-directed (first-order) functional programming language, in which the orthogonal conditional operator

\[
\begin{align*}
if (T, x, y) & \rightarrow x \\
if (F, x, y) & \rightarrow y
\end{align*}
\]

can also conveniently be incorporated. Various strategies have been developed for efficient computation in special cases. Moreover, orthogonal systems lend themselves easily to parallel evaluation schemes.

Huet and Lévy [1991] developed a theory of “needed redexes” and optimal derivations for orthogonal systems. The need for a redex is, however, undecidable, except in special cases [Hoffmann and O’Donnell 1982, Huet and Lévy 1991]. Chew [1980] used congruence-closure techniques to cache results of prior sequences of orthogonal rewrites, and improve performance; this idea was extended to a class of non-orthogonal convergent systems in [Verma 1995].

Since programs are often nonterminating, techniques for showing confluence of nonterminating conditional rewrite systems are useful:

10.9. Definition (Conditional Orthogonality [Bergstra and Klop 1986]). A conditional rewrite system is orthogonal if

1. every variable occurring on the right side or in a condition also appears on the left,
2. each variable occurs at most once in a left-hand side of a rule,
3. one side of each condition is a ground normal form,
4. no left-hand side unifies with a renamed nonvariable subterm of any other left-hand side or with a proper subterm of itself, and
5. no left-hand side is just a variable.

10.10. Theorem ([Bergstra and Klop 1986]). Every orthogonal conditional rewrite system is confluent.

This definition of orthogonality could be weakened to allow overlaps when the conjunction of the conditions of the overlapping rules cannot be satisfied by the rules of the system. This is the case with the Conditional Append example, since only the last two rules overlap, but \(\text{null}(c)\) can never be \(F\).

As indicated earlier, there are various methods of defining the semantics of conditional rewrite systems. For example, if we have arbitrary conditions as in

\[
\begin{align*}
p(c) & \quad a \rightarrow b \\
\neg p(c) & \quad a \rightarrow b
\end{align*}
\]

can we rewrite \(a\) to \(b\)? We might say yes, since either \(p(c)\) is true or \(\neg p(c)\) is. We might say no, since neither condition can be proved. For discussions of logic-based semantics and alternative operational semantics for conditional systems, see
Conditional equations provide a natural bridge between functional programming, based on equational semantics, and logic-programming, based on Horn clauses. Note that the above rules can be expressed as

\[ p(c) = T \quad | \quad a \rightarrow b \]
\[ \neg p(c) = T \quad | \quad a \rightarrow b \]

In this way, we can convert conditions involving arbitrary formulae to conditions involving equations. However, the law of the excluded middle no longer holds; we do not have \( x = T \) or \( x = F \) for all \( x \). This changes the semantics, of course. Interpreting definite Horn clauses \( p \lor \lnot q_1 \lor \cdots \lor \lnot q_n \) as conditional rewrite rules, \( q_1 \perp T \land \cdots \land q_n \perp T \mid p \rightarrow T \), gives a system satisfying the constraints of Theorem 9.3, because predicate symbols are never nested in the “head” \( p \) of a clause. Furthermore, all critical pairs are joinable, since all right-hand sides are just \( T \).

However, logic programming permits variables to be bound by unification, whereas conditional rewriting typically uses matching instead, which is more restrictive. To simulate a language like Prolog, something like “conditional narrowing” is needed. See [Dershowitz and Plaisted 1988] for one approach to conditional narrowing. (See [Baader and Snyder 2001, page xxx] in Chapter 8 of this Handbook, for the definition of narrowing and related equation-solving methods.) Solving existential queries for conditional equations corresponds to the logic-programming capability of resolution-based languages like Prolog. Goals of the form \( s =^? t \) can be solved by a linear restriction of paramodulation akin to narrowing (for unconditional equations) and to the selected linear strategy for Horn-clause logic. If \( s \) and \( t \) are unifiable, then the goal is satisfied by any instance of their most general unifier. Alternatively, if there is a (renamed) conditional rule \( C \mid l \rightarrow r \) such that \( l \) unifies with a nonvariable (selected) subterm of \( s \) via most general unifier \( \mu \), then the conditions in \( C\mu \) are solved, say via substitution \( \rho \), and the new goal becomes \( s\mu\rho =^? t\mu\rho \).

Suppose we wish to solve

\[ \text{append}(x, y) =^? x \]

using Conditional Append (9.1). To apply the conditional rule, we need first to solve \( \text{null}(x) =^? F \) using the (renamed) rule \( \text{null}(u : v) \rightarrow F \), thereby narrowing the original goal to

\[ \text{head}((u : v), \text{append}(\text{tail}(u : v), y)) =^? u : v \]

Straightforward rewriting reduces this to

\[ u : \text{append}(v, y) =^? u : v \]
to which the first rule for append applies (letting v be ε), giving a new goal u : y =? u : v. Since the two terms are now unifiable, this process has produced the solution x ː→ u : ε and y, v ː→ ε.

For ground confluent conditional systems, any equationally satisfiable goal can be solved by the method outlined above. Some recent proposals for logic programming languages, incorporating equality, adopt such an operational mechanism. The idea of adding rewrite-based equation solving to rewriting to provide a functional-logic language originated with [Dershowitz 1985, Fribourg 1985, Goguen and Meseguer 1986, Dershowitz and Plaisted 1988]. A number of experimental languages combine narrowing with outermost (“lazy”) evaluation to add goal solving capabilities within functional languages. See [Reddy 1986, Hanus 1994].

Simplification via terminating rules is a very powerful feature, particularly when defined function symbols are allowed to be arbitrarily nested in left-hand sides (which is not permitted with orthogonal rules). Assuming ground convergence, any strategy can be used for simplification, and completeness of the goal-solving process is preserved. One way negation can be handled is by incorporating negative information in the form of rewrite rules which are then used to simplify subgoals to F. Combined with eager simplification, this approach has the advantage of allowing unsatisfiable goals to be pruned, thereby avoiding some potentially infinite paths. Various techniques are also available to help avoid some superfluous paths that cannot lead to solutions.

The semantics of rewriting with infinite structures was explored in [Dershowitz, Kaplan and Plaisted 1991, Kennaway, Klop, Sleep and de Vries 1995].

Acknowledgments

The authors gratefully acknowledge the contributions of Jürgen Giesl, Bernhard Gramlich, Mitch Harris, Konstantin Korovin, Vincent van Oostrum, and Rakesh Verma to the preparation of this survey. The National Science Foundation partially supported the writing of this survey under grant NSF CCR-9972118.

Bibliography


Bachmann L. and Dershowitz N. [1989], 'Completion for rewriting modulo a congruence', Theoretical Computer Science 67(2 & 3).


Brand D., Darringer J. A. and Joyner, W. J. J. [1979], Completeness of conditional reductions, in 'Proceedings of the Fourth Workshop on Automated Deduction', Austin, TX.


Butler G. AND Lankford D. S. [1980], Experiments with computer implementations of procedures which often derive decision algorithms for the word problem in abstract algebras, Memo MTP-7, Department of Mathematics, Louisiana Tech. University, Ruston, LA.


Dauchet M. [1992], 'Simulation of Turing machines by a regular rewrite rule', Theoretical Computer Science 103(2), 409–420.


Dershowitz N. [1982], 'Orderings for term-rewriting systems', Theoretical Computer Science 17(3), 279–301.


GEUPEL O. [1989], Overlap closures and termination of term rewriting systems, Report MIP-8922, Universität Passau, Passau, West Germany.


GRAMLICH B. [1995], ‘Abstract relations between restricted termination and confluence properties of rewrite systems’, Fundamenta Informaticae 24, 3–23.


Kamin S. and Lévy J.-J. [1980], Two generalizations of the recursive path ordering, Unpublished note, Department of Computer Science, University of Illinois, Urbana, IL.
KOROVIN K. AND VORONKOV A. [2000], Knuth-Bendix constraint solving is NP-complete, Preprint CSPP-8, Department of Computer Science, University of Manchester.
KRUSKAL J. [1960], 'Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture', Transactions of the American Mathematical Society 95, 210–225.

LANKFORD D. S. [1975], Canonical inference, Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX.

LANKFORD D. S. [1977], Some approaches to equality for computational logic: A survey and assessment, Memo ATP-36, Automatic Theorem Proving Project, University of Texas, Austin, TX.

LANKFORD D. S. [1979], On proving term rewriting systems are Noetherian, Memo MTP-3, Department of Mathematics, Louisiana Tech. University, Ruston, LA.

LANKFORD D. S. AND BALLANTYNE A. M. [1977], Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions, Memo ATP-39, Department of Mathematics and Computer Sciences, University of Texas, Austin, TX.


LESCANNE P. [1990], On the recursive decomposition ordering with lexicographical status and other related orderings, J. Automated Reasoning 6, 39–49.


MÉTIVIER Y. [1983], 'About the rewriting systems produced by the Knuth-Bendix completion algorithm', Information Processing Letters 16(1), 31–34.


OYAMAGUCHI M. [1987], ‘The Church-Rosser property for ground term rewriting systems is decidable’, Theoretical Computer Science 49(1), 43–79.

PETERSEN J. [1984a], Confluence Methods and the Word Problem in Universal Algebra, PhD thesis, Emory University.


PETERSEN J. F. [1984b], Confluence methods and the word problem in universal algebra, PhD thesis, Emory University.


PLAISTED D. [1978], A recursively defined ordering for proving termination of term rewriting systems, technical report R-78-943, University of Illinois at Urbana-Champaign, Urbana, IL.


Weiermann A. [1995], 'Termination proofs via the lexicographic path ordering yields multiply recursive derivation lengths', Theoretical Computer Science 139, 355-362.
Winkler F. and Buchberger B. [1983], A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm, in 'Proceedings of the Colloquium on Algebra, Combinatorics and Logic in Computer Science'.
Zantema H. [1995], 'Termination of term rewriting by semantic labelling', Fundamenta Informaticae 24, 89-105.
Index

Abelian groups, 38, 45, 49
absorbing rule, 49
Ackermann’s function, 19
append, 52, 62
arity, 7
associative-commutative rewriting, 43

bag, 14, 48
bag extension, 14
Boolean rings, 50

canonical system, 9
Cartesian closed categories, 32
Chameleon Island, 4, 12, 45
Church-Rosser modulo, 42
Church-Rosser property, 9, 25, 26, 42
class confluence, 42
class rewriting, 41
closure, 8
combinatory logic, 31, 32, 61
commutative axiom, 41
commute, 27
complete simplification ordering, 37
completeness, 9
completion, 9, 33, 36, 54
  conditional .................. 55
  constrained .................. 48
  ordered ..................... 11
composition, 8
conditional completion, 55
conditional critical pair, 53
conditional equation, 52
conditional rewrite systems, 52
conditional rewriting, 51
confluence, 9, 25-27
  ground ..................... 26, 27
  local .......................... 28
  weak .......................... 28
congruence closure, 34
consequence
  logical .......................... 9
constraints, 23
constructor, 8
correct, 7
contract, 8
congruence, 9
convergent system, 27
convertible, 26
Coxeter groups, 49
critical pair, 28, 29
  conditional .................. 53
  ordered ..................... 35
Critical Pair Lemma, 29
decision procedures, 27
decreasing strategy, 58
decreasing system, 54
deduction modulo, 47
dependency pairs, 24
derivation, 8
derivation length, 13
derives, 26
Diamond Lemma, 28
distributivity, 20
embedding
  homeomorphic .................. 15
encompassment, 30
equation, 8
equational system, 8
equivalence classes, 41
extended rewriting, 41
extensions, 43
factorial, 21, 32
fair termination, 13
fairness, 37, 57
fields, 49
flattening, 43
Fortran, 33
forward closures, 24
Gödel, 51
Grecian urn, 4, 12, 46
Gröbner basis, 47
ground confluence, 26
ground instance, 8
ground systems, 26
group theory, 29, 40
groupoid, 34
groups, 49
Hercules, 21
Hindley-Rosen Lemma, 27
hole, 7
homeomorphic embedding, 15
Horn clause, equational, 52
Hydra, 3
incrementality, 22
induction, 26
inductionless induction, 51
innermost, 8
insertion sort, 4, 20
instance, 8
interpreter, 5, 12, 31
invariance
   full .................................. 12
   irreducible, 9
J-algebras, 49
join systems, 52
joinable, 26
Knuth-Bendix ordering, 23
König's lemma, 13, 18
lambda calculus, 31, 61
lattices, 48, 49
left linearity, 8, 60
lexicographic path ordering, 19
linear, 7
local confluence, 28
logics
   rewriting .............................. 9
   loops, 5, 11, 33, 49
maximal ordering, 23
model, 9
modularity, 7, 59
monotonicity, 12
most general unifier, 28
Moufang identities, 49
multiset, 14
multiset path ordering, 20
Newman's Lemma, 28
Noetherian, 53
non-erasing systems, 32
nondeterminism, 33
normal form, 9, 10
normal form proof, 57
normalization
   unique ............................. 11
   normalizing, 11
numeric path ordering, 23
ordered rewriting, 8
ordered simplification, 48
ordering
   incremental .......................... 22
   Knuth-Bendix .......................... 23
   lexicographic path ................... 19
   numeric path .......................... 23
   precedence ........................... 19
   quasi-simplification .................. 15
   recursive decomposition ............ 22
   recursive path ....................... 19
   reduction ............................ 13
   simplification ....................... 15
   stable ............................... 12
   subterm .............................. 13
   well-founded ........................ 12
ordinal notations, 22
ordinals, 22
orthogonality, 31, 32, 60, 61
outermost, 8
outermost normalization, 60
outermost rewriting, 60
overlap closures, 24
overlapping, 29
overlay, 29, 53, 54
paramodulation, 25
partial ordering, 7
positions, 7
precedence, 19
primitive recursive function, 21
program synthesis, 40
Prolog, 52, 62
quasi-ordering, 7, 13
quasi-simplification ordering, 15
quasi-varieties, 54
recursive decomposition ordering, 22
recursive function, 27
recursive path ordering, 19, 22
reduced convergent system, 30
reduced system, 30
reducible, 9
reduction ordering, 13
redundancy, 40
relativized rewriting, 41
relativized termination, 47
rewrite proof, 10, 26
rewrite relation, 8, 12
rewrite rule, 7, 8
rewrite system, 8
completeness .................. 9
conditional .................. 52
convergent .................. 9
ground ...................... 26, 34
higher-order .................. 7
orthogonal .................. 5
string ....................... 6
rewrite systems
conditional .................. 52
rewriting, 8
conditional .................. 51
right linearity, 8
rings, 46, 49
Robbins algebra, 49
satisfiability, 9
self-embedding, 16, 18
semantic matching, 42
semantic unification, 42
semantics, 9, 24, 61
semi-equational systems, 52
semi-Thue systems, 6
sequence ordering, 13
sequentiality, 7
simplification ordering, 15
strict .......................... 15
Skolemization, 34, 50, 56, 57
stable extension, 12
stack, 6
standard systems, 52
status, 22
string rewriting, 6
strong confluence, 27
strong normalization, 9
substitution, 8
ground ....................... 8
subterm, 7
subterm property, 37
subterms
   disjoint ..................... 8
superposition, 10
term, 7
ground ....................... 7
term-rewriting system, 8
termination, 9, 11, 39
   fair .......................... 13
   simple ....................... 16
   weak ........................ 11
termination function, 17
termination function, 17
termination function, 17
term
   equational ..................... 9
total recursive function, 22
Tree Theorem, 16
unifiable, 28
unifier, 28
unit strategy, 57
validity, 9, 10
valley proof, 10
variant, 8
variety, 11
weak confluence, 28
well-foundedness, 12
well-ordering, 12
Whitman normal form, 49
word problem, 11
universal .................... 11

Zorn’s Lemma, 12
Concept Index