Chapter 9

Rewriting

Nachum Dershowitz

David A. Plaisted

Second readers: Bernhard Gramlich, Mitch Harris, and Konstantin Korovin.

Contents

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov

If the work entailed amounts to a virtual rewriting,
the resulting typescript or manuscript is a re-write.

—Eric Partridge, *Slang To-day & Yesterday* (1933)

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 presimplified 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:

$$\begin{array}{lcllcl} B\ B & \to & B & W\ W & \to & B \\ W\ B & \to & W & B\ W & \to & W \\ B\ W & \to & W\ B & W\ B & \to & B\ W \end{array}$$

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:

$$\begin{array}{lcllcl} R\ Y & \to & G\ G & Y\ R & \to & G\ G \\ G\ Y & \to & R\ R & Y\ G & \to & R\ R \\ R\ G & \to & Y\ Y & G\ R & \to & Y\ Y \end{array}$$

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, consider the following program to rearrange a list of natural numbers in non-increasing order by inserting elements one-by-one into position:

$$\begin{array}{rcllrcl} max(0,x) & \to & x & & & \\ max(x,0) & \to & x & max(s(x),s(y)) & \to & s(max(x,y)) \\ min(0,x) & \to & 0 & & & \\ min(x,0) & \to & 0 & min(s(x),s(y)) & \to & s(min(x,y)) \\ sort(\epsilon) & \to & \epsilon & sort(x:y) & \to & insert(x,sort(y)) \\ insert(x,\epsilon) & \to & x:\epsilon & insert(x,y:z) & \to & max(x,y):insert(min(x,y),z)) \end{array}$$

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

$$\begin{array}{lcllcl}
x\backslash x & \to & e & x\cdot(x\backslash y) & \to & y\\
x/x & \to & e & (y/x)\cdot x & \to & y\\
e\cdot x & \to & x & x\backslash(x\cdot y) & \to & y\\
x\cdot e & \to & x & (y\cdot x)/x & \to & y\\
e\backslash x & \to & x & x/(y\backslash x) & \to & y\\
x/e & \to & x & (x/y)\backslash x & \to & y
\end{array}$$

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

$$\begin{array}{lcllcl}
x\cdot(x\backslash y) & = & y & (y/x)\cdot x & = & y\\
x\backslash(x\cdot y) & = & y & (x/y)\backslash x & = & y\\
x\backslash(x\cdot y) & = & y & (y\cdot x)/x & = & y
\end{array}$$
$$y/y \;=\; x\backslash x$$

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

$$
\begin{array}{rcl@{\qquad}rcl}
eval(\mathbf{zap0}, \langle x,\ y\rangle) & \to & \langle 0,\ y\rangle & eval(\mathbf{zap1}, \langle x,\ y\rangle) & \to & \langle x,\ 0\rangle \\
eval(\mathbf{inc0}, \langle x,\ y\rangle) & \to & \langle s(x),\ y\rangle & eval(\mathbf{inc1}, \langle x,\ y\rangle) & \to & \langle x,\ s(y)\rangle \\
eval(\mathbf{dec0}, \langle 0,\ y\rangle) & \to & \langle 0,\ y\rangle & eval(\mathbf{dec1}, \langle x,\ 0\rangle) & \to & \langle x,\ 0\rangle \\
eval(\mathbf{dec0}, \langle s(x),\ y\rangle) & \to & \langle x,\ y\rangle & eval(\mathbf{dec1}, \langle x,\ s(y)\rangle) & \to & \langle x,\ y\rangle \\
eval(\mathbf{ifpos0}\ p, \langle 0,\ y\rangle) & \to & \langle 0,\ y\rangle & eval(\mathbf{ifpos1}\ p, \langle x,\ 0\rangle) & \to & \langle x,\ 0\rangle
\end{array}
$$

$$
eval(\mathbf{ifpos0}\ p, \langle s(x),\ y\rangle) \to eval(p, \langle s(x),\ y\rangle)
$$

$$
eval(\mathbf{ifpos1}\ p, \langle x,\ s(y)\rangle) \to eval(p, \langle x,\ s(y)\rangle)
$$

$$
\mathbf{whilepos0}\ p \to (\mathbf{ifpos0}\ p;\ \mathbf{whilepos0}\ p)
$$

$$
\mathbf{whilepos1}\ p \to (\mathbf{ifpos1}\ p;\ \mathbf{whilepos1}\ p)
$$

$$
eval((p;\ q), u) \to eval(q, eval(p, u))
$$

Figure 1: Two-counter machine interpreter.

can be represented as a pair $\langle x,\ y\rangle$. 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 *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:

$$
\begin{array}{rcl@{\qquad}rcl}
top(push(x,y)) & \to & x & pop(push(x,y)) & \to & y \\
empty?(\epsilon) & \to & T & empty?(push(x,y)) & \to & F \\
empty?(x) = F & | & & push(top(x), pop(x)) & \to & x
\end{array}
$$

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 *fgffg*, rewrites any suffix *fgffgx* having prefix *fgffg*. 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. Other surveys of rewriting include [Avenhaus and Madlener 1990, Dershowitz and Jouannaud 1990, Klop 1992, Plaisted 1993, Avenhaus and Madlener 1989, 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 $l \to r$, 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 $\mathcal{F}$, constants $\mathcal{C}$, and variables $\mathcal{X}$ is either a variable $x \in \mathcal{X}$, an individual constant $c \in \mathcal{C}$, or an expression of the form $f(t_1, t_2, \ldots, t_n)$, where $f \in \mathcal{F}$ is a function symbol of arity (number of arguments) n and the t_i are terms.[1] Each function symbol has one or more *arities*, which are nonnegative integers indicating how many arguments it can take. A symbol allowing an arbitrary number of arguments is called *varyadic*. 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 $\mathcal{T}$. 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* $\succsim$ is a reflexive transitive relation. We try to use the sign $>$ for partial orderings and $\succsim$ 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 $x \succsim y$ but not $y \succsim x$. The relation $\succ$ is called the *strict part* of $\succsim$ and is a partial ordering. If both $x \succsim y$ and $y \succsim x$, we say that x and y are equivalent (though they may be distinct) and write $x \approx y$.

We say that a term u is a *subterm* of t and write $t \geq_{\text{sub}} u$ 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 $t >_{\text{sub}} u$ 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 α is a position and t is a term, we would write $t|_\alpha$ for the subterm of t at position α, defined by $t|_\epsilon \equiv t$ for the empty sequence (top position) ϵ; 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 $\diamond$. A lone $\diamond$ is thus the

[1] In this chapter, we consider only unsorted terms, for simplicity.

simplest context. If t is a context with m holes (an *m-context*) and $s_1, \ldots, s_m$ are terms, then $t[s_1, \ldots, s_m]$ indicates t with the $\diamond$'s replaced by $s_1, \ldots, s_m$ in a predetermined order. At the same time, $t[s]$ 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 partial 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. (Though traditional, there is usually no need to restrict the domain of a substitution to be a finite subset of the variables.) Substitutions $\sigma : \mathcal{X} \to \mathcal{T}$ are extended to a total mapping $\sigma : \mathcal{T} \to \mathcal{T}$ of (first-order) contexts; if t is a term or a context and σ is a substitution, we use postfix notation $t\sigma$ for the image of t under σ. This is defined as follows: If $\sigma : x_i \mapsto s_i$, then $x_i\sigma \equiv s_i$; if x is a variable not in the domain of σ, then $x\sigma \equiv x$; if $f(t_1, \ldots, t_k)$ is a nonvariable term then $f(t_1, \ldots, t_k)\sigma \equiv f(t_1\sigma, \ldots, t_k\sigma)$. Also, $\diamond\sigma \equiv \diamond$. 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 $t[u]\sigma \equiv (t\sigma)[u\sigma]$. A term r is a *(renamed) variant* of s if they are instances of each other. If $t\sigma$ is a ground term, we call σ a *ground substitution* for t and $t\sigma$, a *ground instance* of t. Similar terminology applies to instances of equations, rewrite rules, and formulæ.

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 $t[\diamond]$, rules $l \to r$ in R, and substitutions σ, $t[l\sigma] \to_R t[r\sigma]$. That is, if $t[l\sigma]$ 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* $t[l\sigma]$ to $t[r\sigma]$. A redex u is *innermost* for R if it is an 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 $r \leftrightarrow s$ if either $r \to s$ or $r \leftarrow s$. Composition of two relations is indicated by $\circ$. Thus $r \leftarrow^* \circ \to^* s$ 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 *equational 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 $u \to_{E_\to} v$. We may write $\to_E$ 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 $\{\langle r, s\rangle : r \leftrightarrow_E s \text{ and } r > s\}$. This relation may also be simply denoted by $\rightarrow_E$ when the ordering in question is understood.

A *derivation* for a binary relation $\rightarrow$ (for instance, a rewrite relation $\rightarrow_R$) is a sequence of the form $t_0 \rightarrow t_1 \rightarrow t_2 \cdots$. 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, et cetera. 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 $t_0 \rightarrow t_1 \rightarrow t_2 \cdots$. 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 $M_1 \equiv M_2 \pmod{X}$ if M_1 and M_2 agree on all function and constant symbols and on all variables not in X. If $s = t$ 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* $s = t$, written $M \models s = t$, if for all structures M' such that $M \equiv M' \pmod{X}$, $[\![s]\!]_{M'} = [\![t]\!]_{M'}$. One says that equation $s = t$ 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 $R_=$ to be the set of equations $l = r$ for each rule $l \rightarrow r$ in a system R. When R is convergent, $R_= \models s = t$ ($R_=$ logically implies $s = t$) iff 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 $R_=$. If R is not convergent, we may want to *complete* it, that is, find another system S such that their theories are the same

[2] The term *canonical* is sometimes used as a synonym for "convergent", but will not be so used here.

but S is convergent; then S may be used to decide the equational theory $R_=$ 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 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 $R_=$, 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 $s \rightarrow^* \circ \leftarrow^* t$ 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 $\rightarrow^* \circ \leftarrow^*$; termination ensures that $\rightarrow^* \circ \leftarrow^* = \rightarrow^! \circ \leftarrow^!$; 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 r there is an s such that $r \to_R^! s$. 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 $R \models s = t$ by enumerating (in some breadth-first fashion) all derivations from s and t until their normal forms u and v are found, that is, $s \to_R^! u$ and $t \to_R^! v$. Then $R \models s = t$ 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 $RYY \to GGY \to GRR \to YYR$, which rearranges to RYY, 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 $>$ of a set W is *well-founded* if there are no infinite descending chains $x_0 > x_1 > x_2 > \cdots$ 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 $\to$ on a set $\mathcal{T}$ of terms satisfies the *monotonicity property* if $s \to t$ implies $u[s] \to u[t]$ for all terms s, t in $\mathcal{T}$ and all contexts u over $\mathcal{T}$.

We will use the term "monotonic" in this sense only.

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

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 $\to$ on a set $\mathcal{T}$ of free terms satisfies the *full invariance property* if for all terms $s, t \in \mathcal{T}$ and all substitutions σ over $\mathcal{T}$, $s \to t$ implies $s\sigma \to t\sigma$.

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

The rewrite step $\to_R$ 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 $l \to r \in R$ (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 $\to_R^+$. 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 ∞ 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 $t \to_R^* u$ 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 $\succsim_i$ are quasi-orderings of sets of elements S_i, then the lexicographic extension $\succsim_{\text{lex}}$ of the $\succsim_i$ to arbitrary-length sequences in

$\cup_n(S_1 \times S_2 \times \cdots \times S_n)$ is defined as follows:[3]

$$\frac{}{\langle\,\rangle \approx_{\text{lex}} \langle\,\rangle} \qquad \frac{\langle s_1, \ldots, s_m\rangle \succsim_{\text{lex}} \langle t_1, \ldots, t_n\rangle}{\langle s_1, \ldots, s_m, s_{m+1}\rangle \succ_{\text{lex}} \langle t_1, \ldots, t_n\rangle}$$

$$\frac{\langle s_1, \ldots, s_n\rangle \approx_{\text{lex}} \langle t_1, \ldots, t_n\rangle \qquad s_{n+1} \approx_{n+1} t_{n+1}}{\langle s_1, \ldots, s_{n+1}\rangle \approx_{\text{lex}} \langle t_1, \ldots, t_{n+1}\rangle}$$

$$\frac{\langle s_1, \ldots, s_m\rangle \approx_{\text{lex}} \langle t_1, \ldots, t_m\rangle \qquad s_{m+1} \succ_{m+1} t_{m+1}}{\langle s_1, \ldots, s_{m+1}\rangle \succ_{\text{lex}} \langle t_1, \ldots, t_{m+1}, \ldots, t_n\rangle}$$

where $\succsim_{\text{lex}}$ is the union of the equivalence relation $\approx_{\text{lex}}$ and the partial ordering $\succ_{\text{lex}}$.

It is not hard to see that the relation $\succsim_{\text{lex}}$ 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 (multiset) extension* $\succsim_{\text{bag}}$ of a quasi-ordering $\succsim$ of elements is defined as follows:

$$\frac{}{[\,] \approx_{\text{bag}} [\,]} \qquad \frac{[s_1, \ldots, s_n] \approx_{\text{bag}} [t_1, \ldots, t_n] \qquad s \approx t}{[s_1, \ldots, s_n, s] \approx_{\text{bag}} [t_1, \ldots, t_n, t]}$$

$$\frac{[s_1, \ldots, s_m] \succsim_{\text{bag}} [t_1, \ldots, t_n] \qquad s \succ u_1, \ldots, u_k}{[s_1, \ldots, s_m, s] \succ_{\text{bag}} [t_1, \ldots, t_n, u_1, \ldots, u_k]}$$

where $s \succ u_1, \ldots, u_k$ (for $k \geq 0$) means that s is greater than each of the u_i and $\succsim_{\text{bag}}$ is the union of the equivalence relation $\approx_{\text{bag}}$ 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 $[3, 4, 4] >_{\text{bag}} [2, 2, 2, 1, 2, 4, 4]$, 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 $>_{\text{bag}}$ 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 $\succsim_{\text{bag}}$ 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 $\frac{A_1 A_2 \ldots A_n}{A}$ 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.*

Term orderings used for proving termination of rewriting or completeness of rewrite-based theorem provers usually have the following properties:

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

$$f(\ldots, x_i, \ldots) \succsim x_i$$

If f is varyadic, we also require

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

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 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 $\geq_{\text{emb}}$ on a set $\mathcal{T}$ of fixed-arity terms is the derivability relation of the rewrite rules

$$f(x_1, \ldots, x_i, \ldots, x_n) \to x_i$$

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

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

If $t \leq_{\text{emb}} s$, that is, if $s \to^* t$ 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.

The crucial point is that simplification orderings $\succsim$ contain the homeomorphic embedding relation $\geq_{\text{emb}}$. Strict simplification orderings $\succ$ 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 $\mathcal{T}$ with finitely many function symbols and constants contains two terms t_j and t_k $(j < k)$ such that $t_j \leq_{\text{emb}} t_k$.*

Kruskal extended Higman's [1952] work 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 $t_1 \to t_2 \to \cdots \to t_j \to \cdots \to t_k \to \cdots$ of terms is *self-embedding* if $t_j \leq_{\text{emb}} t_k$ 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 $>$ 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 $f(f(x)) \to f(g(f(x)))$.

4.18. THEOREM (Dershowitz 1982). *Suppose R is a finite rewrite system and $\succsim$ is a fully-invariant simplification ordering. If $R \subseteq \succ$, then R is terminating.*

PROOF. Suppose $s_1 \to_R s_2 \to_R \cdots \to_R s_n \to_R \cdots$. Then $s_1 \succsim s_2 \succsim s_3 \cdots$ because the quasi-ordering $\succsim$ 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 some 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* τ is a function from a set of terms $\mathcal{T}$ to a set W, supplied with a well-founded (quasi-) ordering $\gtrsim$ that has the monotonicity property, such that for all terms in $\mathcal{T}$,

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

For variable-arity symbols f, we also require

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

4.20. THEOREM. *Suppose R is a rewrite system over a set $\mathcal{T}$ of terms and τ is a termination function, then R is terminating if for all rules $r \to s$ in R and for all substitutions σ over $\mathcal{T}$,*

$$\tau(r\sigma) \succ \tau(s\sigma)$$

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

Typically, termination functions $\tau : \mathcal{T} \to W$ are defined as a set of homomorphisms $\tau_f : W^n \to W$, 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 τ 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 $\tau_f(x_1, \ldots, x_n) = 1 + x_1 + \cdots + x_n$ 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 $r \to s$ 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 $\tau_f(x_1, \ldots, x_n) = c_f + x_1 + \cdots + x_n$, 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, one can devise termination functions by letting τ_f be an arbitrary multivariate polynomial in $\bar{x}$ 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, Giesl 1995]. One must also show that

$\tau_f(x_1 \ldots x_n) \geq 0$ if $x_i \geq 0$ for all i, $1 \leq i \leq n$. It is necessary to restrict W to the natural numbers so that the ordering $>$ on integers is well-founded. Of course, if all the coefficients are positive integers, these conditions hold. This technique has the advantage of flexibility, since different polynomials can be designed for different systems; however, 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. □

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 σ such that $r \to_R^+ u[r\sigma]$. 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 system. These orderings can, for example, easily handle the distributive rule $x \times (y + z) \to x \times y + x \times z$. 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* $\succsim_{\text{lpo}}$ induced by a quasi-ordering $\succsim$ on function symbols is defined as follows:

$$\frac{s_i \succsim_{\text{lpo}} t}{f(s_1, \ldots, s_m) \succ_{\text{lpo}} t}$$

$$\frac{f \succ g \qquad f(s_1, \ldots, s_m) \succ_{\text{lpo}} t_1, \ldots, t_n}{f(s_1, \ldots, s_m) \succ_{\text{lpo}} g(t_1, \ldots, t_n)} \qquad \frac{f \approx g \qquad \langle s_1, \ldots, s_m \rangle \approx_{\text{lex}} \langle t_1, \ldots, t_n \rangle}{f(s_1, \ldots, s_m) \approx_{\text{lpo}} g(t_1, \ldots, t_n)}$$

$$\frac{f \approx g \qquad \langle s_1, \ldots, s_m \rangle \succ_{\text{lex}} \langle t_1, \ldots, t_n \rangle \qquad f(s_1, \ldots, s_m) \succ_{\text{lpo}} t_2, \ldots, t_n}{f(s_1, \ldots, s_m) \succ_{\text{lpo}} g(t_1, \ldots, t_n)}$$

where $\succsim_{\text{lpo}}$ is the union of the equivalence relation $\approx_{\text{lpo}}$ and the partial ordering $\succ_{\text{lpo}}$ and $\succ_{\text{lex}}$ and $\approx_{\text{lex}}$ are the lexicographic extensions of $\succ_{\text{lpo}}$ and $\approx_{\text{lpo}}$, respectively (see Definition 4.8).

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

This ordering has the useful property that $f(f(x, y), z) \succ_{\text{lpo}} f(x, f(y, z))$; informally, the reason is that—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 $x \times (y + z) \succ_{\text{lpo}} x \times y + x \times z$ (in the stable extension of the ordering), as follows:

$$\dfrac{\dfrac{\dfrac{x \approx_{\text{lpo}} x \quad \dfrac{y \approx_{\text{lpo}} y}{y + z \succ_{\text{lpo}} y}}{\langle x, y + z\rangle \succ_{\text{lex}} \langle x, y\rangle} \quad \dfrac{\dfrac{y \approx_{\text{lpo}} y}{y + z \succsim_{\text{lpo}} y}}{x \times (y + z) \succ_{\text{lpo}} y}}{x \times (y + z) \succ_{\text{lpo}} x \times y} \quad \times \succ + \quad \dfrac{\dfrac{\dfrac{x \approx_{\text{lpo}} x \quad \dfrac{z \approx_{\text{lpo}} z}{y + z \succ_{\text{lpo}} z}}{\langle x, y + z\rangle \succ_{\text{lex}} \langle x, z\rangle} \quad \dfrac{\dfrac{z \approx_{\text{lpo}} z}{y + z \succsim_{\text{lpo}} z}}{x \times (y + z) \succ_{\text{lpo}} z}}{x \times (y + z) \succ_{\text{lpo}} x \times z}}{x \times (y + z) \succ_{\text{lpo}} x \times y + x \times z}$$

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

$$\dfrac{s_i \succsim_{\text{mpo}} t}{f(s_1, \ldots, s_m) \succ_{\text{mpo}} t} \qquad \dfrac{f \succ g \qquad f(s_1, \ldots, s_m) \succ_{\text{mpo}} t_1, \ldots, t_n}{f(s_1, \ldots, s_m) \succ_{\text{mpo}} g(t_1, \ldots, t_n)}$$

$$\dfrac{f \approx g \qquad [s_1, \ldots, s_m] \approx_{\text{bag}} [t_1, \ldots, t_n]}{f(s_1, \ldots, s_m) \approx_{\text{mpo}} g(t_1, \ldots, t_n)} \qquad \dfrac{f \approx g \qquad [s_1, \ldots, s_m] \succ_{\text{bag}} [t_1, \ldots, t_n]}{f(s_1, \ldots, s_m) \succ_{\text{mpo}} g(t_1, \ldots, t_n)}$$

where $\succsim_{\text{mpo}}$ is the union of the equivalence relation $\approx_{\text{mpo}}$ and the partial ordering $\succ_{\text{mpo}}$ and $\succsim_{\text{bag}}$ is the bag extension of $\succsim_{\text{mpo}}$, 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 $x \times (y + z) \succ_{\text{mpo}} y \times x + z \times x$ (in the stable extension of the ordering), as follows: we have:

$$\dfrac{\dfrac{\dfrac{x \approx_{\text{mpo}} x \quad \dfrac{y \approx_{\text{mpo}} y}{y + z \succ_{\text{mpo}} y}}{[x, y + z] \succ_{\text{bag}} [y, x]}}{x \times (y + z) \succ_{\text{mpo}} y \times x} \quad \dfrac{\dfrac{x \approx_{\text{mpo}} x \quad \dfrac{z \approx_{\text{mpo}} z}{y + z \succ_{\text{mpo}} z}}{[x, y + z] \succ_{\text{bag}} [z, x]}}{x \times (y + z) \succ_{\text{mpo}} z \times x} \quad \times \succ +}{x \times (y + z) \succ_{\text{mpo}} y \times x + z \times x}$$

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 *max* and *min*, we can use the precedence $max, min \succ s$. For the base case of *insert*, we let $insert \succ :$ in the precedence. For the recursive rule of *sort*, we let $sort \succ insert$. Finally, for the recursive rule of *insert*, we make $insert \succ max, min$ 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 $\succ_{\text{mpo}}$. Removing a head makes a branch smaller in $\succ_{\text{mpo}}$. 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 $f(s) \to u[f(t_1), \ldots, f(t_n)]$ 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 $f(s) \succ u[f(t_1), \ldots, f(t_n)]$, so the rule $f(s) \to u[f(t_1), \ldots, f(t_n)]$ is decreasing.

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

$$\begin{array}{lcllcl} 0 + x & \to & x & s(x) + y & \to & s(x + y) \\ 0 \times x & \to & 0 & s(x) \times y & \to & y + (x \times y) \\ \mathit{fact}(0) & \to & s(0) & \mathit{fact}(s(x)) & \to & s(x) \times \mathit{fact}(x) \end{array}$$

For this system, we choose the precedence $\mathit{fact} \succ \times \succ + \succ s \succ 0$. 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 $l \to r$. 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 $\mathit{fact}(s(x)) \to s(x) \times \mathit{fact}(x)$, the dominant term on the left is $\mathit{fact}(s(x))$ and on the right is $\mathit{fact}(x)$. Since $s(x) \succ x$, $\mathit{fact}(s(x)) \succ \mathit{fact}(x)$ 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

$$\begin{aligned} f(\ldots x_i \ldots) &\rightarrow \mu_t(0, t(0, \ldots x_i \ldots), \ldots x_i \ldots) \\ \mu_t(n, s(z), \ldots x_i \ldots) &\rightarrow n \\ \mu_t(n, 0, \ldots x_i \ldots) &\rightarrow \mu_t(s(n), t(s(n), \ldots x_i \ldots), \ldots x_i \ldots) \end{aligned}$$

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 $t(n, \ldots a_i \ldots) \neq 0$. 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 $\succsim$, as follows:

$$\succsim^{\sup}_{\text{mpo}} \quad = \quad \bigcap_{\succsim' \supseteq \succsim} \succsim'_{\text{mpo}}$$

Only total extensions $\succsim'$ 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 $\geq'$ of the precedence, $f(a) >'_{\text{mpo}} g(b)$. If $f \geq a$ then $f > g$, and f is the maximal symbol of the two terms and $f(a) >'_{\text{mpo}} g(b)$. If $a \geq f$ then a is the maximal symbol of the two terms, and $f(a) >'_{\text{mpo}} g(b)$. Since one or the other must hold in any total extension of the precedence, we have $f(a) >^{\sup}_{\text{mpo}} g(b)$.

A more efficient way of determining if $s >^{\sup}_{\text{mpo}} t$ is to construct a constraint B involving the precedence $\geq$ and function symbols such that $s \geq_{\text{mpo}} t$ iff the constraint holds. This can be derived systematically from the rules for the multiset path ordering. To decide whether $s >^{\sup}_{\text{mpo}} t$, we need to show that all extensions of $\geq$ to a total precedence $\geq'$ satisfy the constraint. This avoids repeated computations on the term structures. The precedence $\geq$ 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 $C \wedge Q \Rightarrow B$. This is a straightforward theorem-proving problem, and can be approached by a number of methods that are reasonably efficient on small formulæ. 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 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 NP-complete in [Nieuwenhuis 1993, Narendran, Rusinowitch and Verma 1998].

Another important class of orderings, the *numeric path orderings*, uses homomorphic interpretations, perhaps in conjunction with a precedence. The *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, Kalmus and Martin 1990]. In [Korovin and Voronkov 2001] a polynomial-time algorithm is 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 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 Gnaedig 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 $l\sigma > s\sigma$ for nonvariable subterms s of the right side of a rule $l \to r$, and that $l\sigma > t$ for all t derivable from $s\sigma$ without any top-level rewrite. One way to establish that this termination condition holds is to show that $u \geq' v$ whenever u rewrites to v (using a quasi-simplification ordering), but that $l\sigma > s\sigma$ for each nonvariable subterm s of the right-hand side using another, related (non-monotonic) ordering for which $u \geq' v$ implies $f(\ldots u \ldots) \geq f(\ldots v \ldots)$, 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:

$$\begin{array}{rclcrcl} 0+x & \to & x & \qquad & 0\times x & \to & 0 \\ s(x)+y & \to & s(x+y) & & s(x)\times y & \to & y+(x\times y) \end{array}$$

We can use the following termination function $\tau_0 : \mathcal{T} \to \mathbf{N}$:

$$\begin{array}{rclcrcl} \tau_0(0) & = & 1 & \qquad & \tau_0(s(x)) & = & \tau_0(x)+2 \\ \tau_0(x+y) & = & 2\tau_0(x)+\tau_0(y) & & \tau_0(x\times y) & = & \tau_0(y)\cdot 2^{\tau_0(x)} \end{array}$$

Or, to avoid exponentials, we could use the lexicographic combination $\langle \tau_1(t), \tau_2(t) \rangle$ of two simpler interpretations:

$$\begin{array}{rclcrcl} \tau_1(0) & = & 1 & \qquad & \tau_2(0) & = & 0 \\ \tau_1(s(x)) & = & \tau_1(x)+2 & & \tau_2(s(x)) & = & \tau_2(x)+2 \\ \tau_1(x+y) & = & \tau_1(x)+\tau_1(y) & & \tau_2(x+y) & = & 2\tau_2(x)+\tau_2(y) \\ \tau_1(x\times y) & = & \tau_1(x)\cdot\tau_1(y) & & \tau_2(x\times y) & = & 0 \end{array}$$

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:

$$\begin{array}{rclrcl} \tau'(0) & = & 0 & \tau'(s(x)) & = & \tau'(x)+1 \\ \tau'(x+y) & = & \tau'(x)+\tau'(y) & \tau'(x\times y) & = & \tau'(x)\cdot\tau'(y) \end{array}$$

to show that $u \to t$ implies $u \geq' t$. Then for $>$ use the termination function:

$$\begin{array}{rclrcl} \tau(0) & = & \langle 0,0\rangle & \tau(s(x)) & = & \langle 0,0\rangle \\ \tau(x+y) & = & \langle 1,\tau'(x)\rangle & \tau(x\times y) & = & \langle 2,\tau'(x)\rangle \end{array}$$

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 $s(x)\times y \to (y\times x)+y$, 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 $v_1, v_2, \cdots, v_n$ 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 $\{r_1 \to s_1, r_2 \to s_2, \ldots\}$ and recall that $R_=$ is the associated equational system $\{r_1 = s_1, r_2 = s_2, \ldots\}$. We write $t =_R u$ iff $R_= \models t = u$, 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 $\to^*$ 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

$$\rightarrow^* \circ \uparrow^{n-1} \circ \leftarrow^* \subseteq \rightarrow^* \circ \downarrow \circ \leftarrow^* \;=\; \downarrow$$

□

Since $s \leftrightarrow^*_R t$ 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 (see [Thomas 1990]) execute a special form of ground rewriting and have been successfully used for proving decision properties in the ground case. 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 theorem 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, $r \to s$ and $r \to t$ imply that $s \leftrightarrow t$, or s and t are identical, or there is a term u such that $s \to u$ and $t \to u$.

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 slightly weaker version of "strong confluence" which also allows for $s \to^* t$ together with $t \to^* s$, 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 $\to_R$ and $\to_S$ *(sub-) commute* if $\leftarrow_R \circ \to_S \subseteq \to_S^= \circ \leftarrow_R^=$.

Recall that $\to^=$ is the reflexive closure of $\to$, allowing for at most one step.

5.10. LEMMA (Hindley-Rosen Lemma [Hindley 1964, Rosen 1973]). *If two strongly confluent relations commute, then their union is confluent.*

The following widely-applicable criterion for confluence extends the Hindley-Rosen method by dividing a relation $\to$ into a family of subrelations $\to_i$. For any set K of indices i, we define $\to_K = \cup_{i \in K} \to_i$.

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 $\leftarrow_i \circ \to_j$*

is contained in the relation $\rightarrow_{I'}^* \circ \rightarrow_j^= \circ \rightarrow_{K'}^* \circ \leftarrow_{K'}^* \circ \leftarrow_i^= \circ \leftarrow_{J'}^*$, *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 $s \downarrow t$ 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 $r_1, r_2, \cdots, r_n$ of elements such that $s \equiv r_1$ and $t \equiv r_n$ and for all i, $r_i \leftrightarrow r_{i+1}$. 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 $r_{i-1} \leftarrow r_i \rightarrow r_{i+1}$ then $s \downarrow t$ immediately. Suppose for some r_i we have $r_{i-1} \leftarrow r_i \rightarrow r_{i+1}$. 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, $S \succ_{\text{bag}} T$. By induction on the well-founded ordering $\succ_{\text{bag}}$, 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 α is a *unifier* of two terms s and t if $s\alpha$ is identical to $t\alpha$. In this case we say that s and t are *unifiable*. We say that a substitution α is *as general as* a substitution β relative to a pair $s = t$ of terms if there is a substitution γ such that $s\alpha\gamma \equiv s\beta$ and $t\alpha\gamma \equiv t\beta$. A substitution α is a *most general unifier* of terms s and t, denoted $\text{mgu}(s, t)$, if α is a unifier of s and t and α is as general as any other unifier of s and t.[6]

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 they are distinct) and μ is a most general unifier of l and a nonvariable subterm s' of s, then the equation $s\mu[r\mu] = t\mu$, where $r\mu$ has replaced $s'\mu$ $(= l\mu)$ in $s\mu$, is a

[6] If s and t are unifiable then most general unifiers exist, are essentially unique, and are easy to compute. Unification is the subject of Chapter 8 of this Handbook [Baader and Snyder 2001].

critical pair of those rules. We also require either that 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 $s \to t$ and $l \to r$ 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 $s\mu[l\mu]$ can rewrite either to $t\mu$, applying the first rule at the top, or can rewrite to $s\mu[r\mu]$, applying the second rule to the redex $l\mu$. So if $u_1 = u_2$ is a critical pair, then $u_1 \leftarrow \circ \rightarrow u_2$ 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 *max* and *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

$$\begin{array}{rcl@{\qquad}rcl} 0+x & \to & x & x+0 & \to & x \\ (-x)+x & \to & 0 & x+(-x) & \to & 0 \\ -0 & \to & 0 & -(-x) & \to & x \\ (-x)+(x+y) & \to & y & x+((-x)+y) & \to & y \end{array}$$

follows from some combination of the three axioms: $x + 0 \to x$, $0 + x \to x$, and $(-x) + (x + y) \to y$. This system has numerous critical pairs, all of which are joinable. For example, the rules $x + (-x) \to 0$ and $x + ((-x) + y) \to y$ 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 $u \to_R s$ and also $u \to_R s'$ for rewrite system R . There must be contexts c and c', rules $l \to r$ and $l' \to r'$ of R and substitutions σ and σ' such that $u \equiv c[l\sigma] \equiv c'[l'\sigma']$, $s \equiv c[r\sigma]$, and $s' \equiv c'[r'\sigma']$. We need to show that $s \downarrow_R s'$.

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

Another possibility is that one redex is "inside a variable" of another; that is, $u \equiv c[l\sigma]$, with σ 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 τ is the same as σ, except that instead of $x \mapsto t[l'\sigma']$ it maps x to $t[\diamond]$, with that hole filled by $l'\sigma'$. So $u \equiv c[l\tau][l'\sigma']$, $c' \equiv c[l\tau]$, $s \equiv c[r\sigma] \equiv c[r\tau][l'\sigma', \ldots, l'\sigma']$, with one occurrence of $l'\sigma'$ filling each hole left by τ for an x that occurs in r, and $s' \equiv c'[r'\sigma'] \equiv c[l\tau][r'\sigma'] \to_R c[r\tau][r'\sigma', \ldots, r'\sigma']$, while $s \equiv c[r\tau][l'\sigma', \ldots, l'\sigma'] \to_R^* c[r\tau][r'\sigma', \ldots, r'\sigma']$ by successive rewrites. Thus $s \downarrow_R s'$.

The only other possibility is that the redexes overlap in a nonvariable subterm. Then $u \equiv c[l\sigma]$, with $l \equiv d[t]$, 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, $s = s' \equiv c[r\sigma] = c[d\sigma[r'\sigma']] \equiv c[p\tau] = c[q\tau]$, for some critical pair $p = q$ or $q = p$ (which is $r\mu = d\mu[r'\mu]$ for most general unifier μ of l' and t), and substitution τ (such that $\mu\tau = \sigma$). Since all critical pairs are joinable, $p\tau \downarrow_R q\tau$ 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 $s \rhd t$ 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 $l \to r$ in R, the right-hand side r is irreducible (unrewritable) and if $l \rhd l'$, 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 variant of itself.

Orthogonality is usually defined in this particular sense, but the weaker notion suffices for most purposes. (Conditions 1 and 2 are often included in the very definition of rewrite system, but that is ill-advised.)

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 $\rightarrow_R^{\|}$ 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 $\{a \to b, f(x) \to g(f(x), f(x))\}$ then $f(a)$ rewrites to both $f(b)$ and $g(f(a), f(a))$. Both of these terms parallel-rewrite to $g(f(b), f(b))$, by replacing both 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. □

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

$$\begin{aligned} d(x,x) &\to e \\ c(x) &\to d(x, c(x)) \\ a &\to c(a) \end{aligned}$$

The system is non-overlapping because the outermost function symbols of each left-hand side are distinct, and there are no nonvariable proper subterms. Note that $c(a) \to^* e$ and $c(a) \to^* c(e)$, but we do not have $e \downarrow c(e)$.

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:

$$\begin{aligned} I \circ x &\to x \\ (K \circ x) \circ y &\to x \\ ((S \circ x) \circ y) \circ z &\to (x \circ z) \circ (y \circ z) \end{aligned}$$

The combinators K and S were dubbed "kestrel" and "starling" by Smullyan [Smullyan 1968]; 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 $\langle \diamond, \diamond \rangle$ are binary operators):

$$\begin{aligned} Ix &\to x & (xy)z &\to x(yz) \\ xI &\to x & \langle x, y\rangle z &\to \langle xz, yz\rangle \\ F\langle x, y\rangle &\to x & E\langle Cx, y\rangle &\to x\langle I, y\rangle \\ S\langle x, y\rangle &\to y & (Cx)y &\to C(x\langle yF, S\rangle) \end{aligned}$$

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) iff 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) iff it is terminating [O'Donnell 1977], as is the case for the λI 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 $[x|y] \to x$ and $[x|y] \to y$.

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

$$\textbf{Expand:} \qquad \frac{E}{E,\ C} \qquad \text{if } E \models C$$

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:

$$\textbf{Contract:} \qquad \frac{E,\ C}{E} \qquad \begin{array}{l}\text{if for every proof } p \text{ of } E \cup C \vdash e,\\ \text{there is a proof } q \text{ of } E \vdash e, \text{ such}\\ \text{that } p \underline{\gg} q\end{array}$$

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 strict simplification ordering to *replace* equations with simpler ones, without changing the theory:

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

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 $s = t$ 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:

$$\begin{aligned} sss0 = 0,\ ss0 = 0,\ ssss0 = s0 &\vdash sss0 = 0,\ ss0 = 0,\ ss0 = s0 \\ &\vdash ss0 = 0,\ ss0 = s0 \\ &\vdash ss0 = 0,\ 0 = s0 \\ &\vdash s0 = 0 \end{aligned}$$

We are using $E \vdash E'$ to indicate one application of an inference rule. The result is the one-rule rewrite system, $s0 \to 0$, which reduces all terms s^i0 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 $u[l\sigma]$ 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 $\to_E$ 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 $\to_E^!$ to reduce both its sides. A trivial equation $s' = t'$ is obtained eventually iff $E \models s = t$.

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

$$\begin{array}{lcllcl} (xy)x & \to & x & \qquad x(yz) & \to & xz \\ (xy)z & \leftrightarrow & (xw)z & ((xy)w)z & \to & xz \end{array}$$

We are using the symbol $\to$ 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

$$y > w \mid \quad (xy)z \quad \to \quad (xw)z$$

(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

$$\begin{array}{rrcll} & (xa)x & \to & x & \qquad x(yz) \to xz \\ y \neq a \mid & (xy)z & \to & (xa)z & \qquad ((xa)a)z \to xz \end{array}$$

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 μ is a most general unifier of l and a nonvariable subterm s' of s, and for some substitution σ, we have $s\mu\sigma \not\leq t\mu\sigma$ and $s\mu\sigma \not\leq s\mu[r\mu]\sigma$ 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 $\mathbf{cp}(E)$ 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 $\mathbf{cp}(E)$. Also, if R is a set of rewrite rules, then $\mathbf{cp}(E \cup R) = \mathbf{cp}(E \cup R_{=})$, permitting critical pairs between rules and equations.

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

PROOF. As in Lemma 5.15, for any peak $s \leftarrow_E \circ \to_E t$ between ground terms s, t, there either exists a rewrite proof $s \downarrow_E t$ or a critical-pair proof $s \leftrightarrow_{\mathbf{cp}(E)} t$. In the latter event, $s \equiv u[l\sigma]$ and $t \equiv u[r\sigma]$ for some critical pair $l = r$, and by assumption $u[l\sigma] \downarrow_E u[r\sigma]E$. So the relation $\to_E$ is locally confluent, and by the Diamond Lemma the rewrite relation $\to_E$ 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:

$$\frac{E,\ R}{E,\ R,\ s = t} \qquad \text{if } s = t \in \mathbf{cp}(E \cup R)$$

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

$$\frac{E,\ R,\ l = r}{E,\ R,\ l \to r} \qquad \text{if } l > r$$

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):

$$\frac{E,\ R,\ l = r}{E,\ R,\ l = r'} \qquad \text{if } r \to_R r' \qquad\qquad \frac{E,\ R,\ l \to r}{E,\ R,\ l \to r'} \qquad \text{if } r \to_{E \cup R} r'$$

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

$$\frac{E,\ R,\ s[l\sigma] = t}{E,\ R,\ s[r\sigma] = t} \qquad \text{if } l = r \in E,\ l \text{ not a variant of } s$$

$$\frac{E,\ R,\ s[l\sigma] \to t}{E,\ R,\ s[r\sigma] \to t} \qquad \text{if } l \to r \in R \text{ or } l = r \in E,\ l \text{ not a variant of } s$$

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 $l \to r'$, since $l = r = r'$ and $l > r > r'$, and then contracts the rules by deleting $l \to r$ which, given $l \to r'$ and the fact that $r \to r'$, 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:

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

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 $>$ of a set of terms $\mathcal{T}$ is called a *complete simplification ordering* if it totally orders all the ground terms in $\mathcal{T}$. 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

$$\mathbf{cp}(E_\infty) \quad \subseteq \quad E_0 \cup E_1 \cup \cdots$$

where $E_\infty = \liminf_j E_j$ $(= \cup_{i \geq 0} \cap_{j \geq i} E_j)$ 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_∞ 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

$$\begin{array}{rclcrcl} x \cdot 1 & = & x & & x \cdot y & = & y \cdot x \\ x \cdot (y \cdot z) & = & (x \cdot y) \cdot z & & x \cdot x^- & = & 1 \end{array}$$

and a lexicographic path ordering (in which $^- > \cdot > 1$ 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:

$$\begin{array}{rcl@{\qquad}rcl} 1^- & \to & 1 & x \cdot y & \leftrightarrow & y \cdot x \\ x \cdot 1 & \to & x & x \cdot (y \cdot z) & \leftrightarrow & y \cdot (x \cdot z) \\ 1 \cdot x & \to & x & (x \cdot y) \cdot z & \to & x \cdot (y \cdot z) \\ (x^-)^- & \to & x & x \cdot (x^- \cdot z) & \to & z \\ x \cdot x^- & \to & 1 & (y \cdot x)^- & \to & x^- \cdot y^- \end{array}$$

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'. 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 [Devie 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 strict monotonic well-ordering of ground terms:

Double: Create one rule to take two steps:

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

6.9. THEOREM (Bofill, Godoy, Nieuwenhuis and Rubio 1999). *Suppose R is a canonical system for axioms E and $>$ is a reduction ordering for which $R \subseteq >$. With any fair strategy, using the rules* **deduce**, **double**, **orient**, *and* **delete**, *the set of persisting rules will include R.*

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 rely on Theorem 4.16 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 redone, 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 $\gg$ 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 $>$ supplied to completion. Then a critical pair $s = t$ is redundant if there exists an equational proof $s \leftrightarrow u_1 \leftrightarrow \cdots \leftrightarrow u_n \leftrightarrow t$, $n \geq 1$, such that term from which the critical pair was formed is greater than each of the intermediate u_i. For the Group Fragment (5.16), the critical pair $(-0)+y = y$, obtained by rewriting $(-0)+(0+y)$ with $0+x \to x$ and $(-x)+(x+y) \to y$, is redundant since $(-0)+(0+y)$ is greater than each of the terms in the alternative proof $(-0)+y \to 0+y \to y$. 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]:

$$x/((((x/x)/y)/z)/(((x/x)/x)/z)) \quad \to \quad y$$

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 $x/x \to e$. 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 $r \to f(x_1, \ldots, x_n)$ and $s \to f(x_1, \ldots, x_n)$ 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 1985*b*].

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 $x \cdot 1 \to x$ 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 $\star$, then the terms $(a \star b) \star a^-$, $a^- \star (a \star b)$, $a \star (b \star a^-)$, etc., will all be rewritable by $x \star x^- \to 1$. 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, we use the equivalence relation $s =_E t$, defined as $E \models s = t$. 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 $[s]_E \to [t]_E$ if $s \to t$. Retracting this back to individual terms, we say that $u \to_{R/E} v$ if there are terms s and t such that $u =_E s$, $s \to_R t$, 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 $u \to v$ if u has a subterm s such that $s =_E l\sigma$ and $l \to r \in R$ and $v \equiv u$ with s replaced by $r\sigma$. In this case, we write $u \to_{E\backslash R} v$ and call the relation $E\backslash R$ 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 $l \to r$ (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 $E\backslash R$, where R has $a \star b \to d$ and E is associativity and commutativity of the operator. Then $(a \star c) \star b \to_{R/E} c \star d$, since $(a \star c) \star b =_E c \star (a \star b)$. However, it is not true that $(a \star c) \star b \to_{E\backslash R} c \star d$ since there is no subterm of $(a \star c) \star b$ that is E-equivalent to $a \star b$. On the other hand, $(b \star a) \star c \to_{E\backslash R} d \star c$, since $b \star a =_E a \star b$.

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 $E\backslash R$ rewriting much more efficient than R/E rewriting, but semantic matching is undecidable in general (Hilbert's Tenth Problem being a special case). 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 will also be a normal form of a term. 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 $t \to^*_{R/E} u$ and $t \to^*_{R/E} v$ then there are u' and v' such that $u \to^*_{R/E} u'$, $v \to^*_{R/E} v'$, 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 $R_= \cup E$-equivalent terms to E equivalent terms by rewriting. Then an E-equivalence procedure can be used, if there is one.

Though $E\backslash R$ rewriting is much more efficient than R/E rewriting, $\to_{R/E}$ has better logical properties for deciding $R \cup E$ 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 $E\backslash R$ is *Church-Rosser modulo* E if any two $R_= \cup E$-equivalent terms can be $E\backslash R$ 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 equational system. If all of the following hold:*

1. *R is finite,*
2. *$\to_{R/E}$ is terminating,*
3. *R is Church-Rosser modulo E,*
4. *E-matching is solvable,*

then equivalence in the combined theory of $R_= \cup E$ is decidable, by normalizing with $\to_{E\backslash R}$.

Note that $\to_{E\backslash R}$ is a subset of $\to_{R/E}$, so if R/E is terminating, so is $E\backslash R$. 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 $\rightarrow_{E\backslash R}$) 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 $\leftarrow_T \circ \leftrightarrow_E$ is contained in $\rightarrow_T^* \circ \leftrightarrow_E^* \circ \leftarrow_T^*$.

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 $\rightarrow_R \subseteq \rightarrow_T \subseteq \rightarrow_{R/E}$. Suppose $\rightarrow_{R/E}$ 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 $E\backslash S$ rewriting is the same as R/E rewriting. Therefore we can use the computationally simpler $E\backslash S$ 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 $E\backslash R$ rewriting can be used rather than R/E rewriting, that is, a term $f(s_1, f(s_2, \ldots, f(s_{n-1}, s_n)\cdots))$, 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, $AC\backslash R$ 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, $AC\backslash R$ rewriting does not simulate R/AC rewriting on the original system. For example, consider $R = \{a \star b \rightarrow d\}$ with AC axioms for $\star$. Suppose $s \equiv (a \star b) \star c$ and t is $d \star c$. Then $s \rightarrow_{R/AC} t$; in fact, $s \rightarrow_{AC\backslash R} t$. However, if we flatten the terms, then s becomes $\star(a,b,c)$ and s no longer rewrites to t since the subterm $\star(a,b)$ has disappeared. To overcome this, we must add *extensions* to rewrite rules to simulate their effect on flattened terms. The extension of the rule $a \star b \rightarrow d$ is $x \star a \star b \rightarrow x \star d$, where x is a new variable. With this extended rule, we do have that $a \star b \star c \rightarrow_{AC\backslash R} d \star c$.

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 $s' \rightarrow_{S \setminus E} t'$. 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 $\mathbf{cp}_{\mathrm{AC}}(E)$ 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:

$$\frac{E,\ R}{E,\ R,\ s = t} \qquad \text{if } s = t \in \mathbf{cp}_{\mathrm{AC}}(E \cup R)$$

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

$$\frac{E,\ R,\ f(s,t) = r}{E,\ R,\ f(s,t) = r,\ f(s, f(t,z)) = f(r,z)} \qquad \text{if } f \text{ is AC, } f(s,t) \not< r$$

   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:

$$\frac{E,\ R,\ l = r}{E,\ R,\ l \rightarrow r} \qquad \text{if } l > r$$

4. Any equation between AC-variants is deleted:

$$\frac{E,\ R,\ s = t}{E,\ R} \qquad \text{if } s =_{AC} t$$

5. AC-rewriting is used for simplification:

$$\frac{E,\ R,\ l = r}{E,\ R,\ l = r'} \qquad \text{if } r \rightarrow_{R/AC} r' \qquad\qquad \frac{E,\ R,\ l \rightarrow r}{E,\ R,\ l \rightarrow r'} \qquad \text{if } r \rightarrow_{E \cup R/AC} r'$$

$$\frac{E,\ R,\ s[l\sigma] = t}{E,\ R,\ s[r\sigma] = t} \qquad \text{if } l \to_{AC\setminus E} r,\ l \text{ not a variant of } s$$

$$\frac{E,\ R,\ s[l\sigma] \to t}{E,\ R,\ s[r\sigma] \to t} \qquad l \to_{AC\setminus E\cup R} r,\ l \text{ not a variant of } s$$

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:

$$\begin{array}{lcl} R\ Y & \to & G\ G \\ R\ R & \to & G\ Y \\ R\ G & \to & Y\ Y \end{array}$$

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

$$\begin{array}{lcl} Y\ R\ z & \to & G\ G\ z \\ R\ R\ z & \to & G\ Y\ z \\ G\ R\ z & \to & Y\ Y\ z \end{array}$$

and their commutative variants. The GR and extended YR rules produce a critical pair $YYY = GGG$, which gets oriented from left to right: $YYY \to GGG$. 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

$$x \cdot 1 \ = \ x \qquad x \cdot x^- \ = \ 1$$

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:

$$\begin{array}{rclrcl} 1^- & \to & 1 & x \cdot 1 & \to & x \\ (x^-)^- & \to & x & (y \cdot x)^- & \to & x^- \cdot y^- \\ x \cdot x^- & \to & 1 & x \cdot (x^- \cdot z) & \to & z \end{array}$$

The last rule is a composed version of the extension $x \cdot (x^- \cdot z) \to 1 \cdot z$ of $x \cdot x^- \to 1$. This extended rule, together with $(y_0 \cdot x_0)^- \to x_0^- \cdot y_0^-$, forms a critical pair

$$(x_2 \cdot z_1)^-(x_2 \cdot (x_1 \cdot x_2)^- \cdot z_2)^- \ = \ (z_1 \cdot z_2)^-$$

[7] The rewriting solution to the Chameleon puzzle is due to Claude Marché.

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 of beans to be removed are adjacent. With extended rules this system is confluent and terminating. For instance, $BW \to W$ and the extended rule $WWz \to Bz$, rewrite a BWW bean arrangement to WW and BB, 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

$$\begin{array}{rclrclrcl} aa & = & a & \qquad bb & = & b & \qquad cc & = & c \\ (a+b+c)(a+b+c) & & & = & & a+b+c & & & \end{array}$$

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

$$\begin{array}{rcl} ba & = & -ab - bc - cb - ac - ca \\ bca & = & abc + acb + cab + cac + cbc + ab + ab + ac + ac + cb + cb + bc + ca \end{array}$$

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 [1985*a*]. 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:

$$\begin{array}{rclrcl} x \sqcap x & = & x & x \sqcup x & = & x \\ x \sqcap y & = & y \sqcap x & x \sqcup y & = & y \sqcup x \\ (x \sqcap y) \sqcap z & = & x \sqcap (y \sqcap z) & (x \sqcup y) \sqcup z & = & x \sqcup (y \sqcup z) \\ (x \sqcup y) \sqcap x & = & x & (x \sqcap y) \sqcup x & = & x \\ & x \sqcup (y \sqcap z) & = & (x \sqcap y) \sqcup (x \sqcap z) & & \end{array}$$

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

$$x \sqcap (y \sqcup z) \quad \rightarrow \quad (x \sqcup y) \sqcap (x \sqcup z)$$

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:

$$\begin{array}{rcl} (xy)z & \rightarrow & x(yz) \\ xy & \leftrightarrow & yx \\ x(yz) & \leftrightarrow & y(xz) \end{array}$$

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 [1984*a*, 1985*a*] 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 1984*b*]

$$(xy)(zw) = (xz)(yw) \qquad (xy)x = x$$

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

$$\begin{array}{rcl@{\qquad}rcl} x \star x & \to & x & x(y \star z) & \to & xz \\ (xy)z & \to & x \star z & x \star (yz) & \to & x \star z \\ x(yz) & \to & xz & (x \star y) \star z & \to & x \star z \\ (x \star y)z & \to & xz & x \star (y \star z) & \to & x \star z \\ (xy) \star z & \to & xz & (xy)(zw) & \leftrightarrow & (xz)(yw) \end{array}$$

An *absorbing* rule is one in which every nonvariable subterm on the left contains all the variables [Pedersen 1985*b*], 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 [1985*b*] 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 [1985*a*] 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 formulæ:

$$\begin{array}{rclcrcl} x \cdot T & \to & x & \qquad & x \oplus F & \to & x \\ x \cdot F & \to & F & & (x \oplus y) \cdot z & \to & (x \cdot z) \oplus (y \cdot z) \\ x \cdot x & \to & x & & x \cdot x \cdot y & \to & x \cdot y \\ x \oplus x & \to & F & & x \oplus x \oplus y & \to & y \end{array}$$

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

$$[\exists x p(x) \wedge \forall x\,(p(x) \Rightarrow p(f(x)))] \;\Rightarrow\; \exists x p(f(f(x)))$$

Its Skolemized negation is equivalent to the following equations:

$$\begin{array}{rcl} p(f(f(x))) & = & F \\ p(a) & = & T \\ p(x) \vee p(f(x)) & = & p(f(x)) \end{array}$$

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

$$\begin{array}{rcl} T & = & T \vee [p(f(a)) \vee p(f(f(a)))] \\ & = & p(a) \vee [p(f(a)) \vee p(f(f(a)))] \\ & = & [p(a) \vee p(f(a))] \vee p(f(f(a))) \\ & = & p(f(a)) \vee p(f(f(a))) \\ & = & p(f(f(a))) \qquad = \quad F \end{array}$$

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 1985*a*], see also [Bachmair and Ganzinger 2001] (Chapter 2 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 $\mathcal{T}(\mathcal{X})$ over a (countable) set $\mathcal{X}$ of variables, for some vocabulary. Then $\mathcal{T}(\mathcal{X})/=_E \models s = t$ iff $s = t$ is a logical consequence of E. And $\mathcal{T}(\emptyset)/=_E \models s = t$ iff $s = t$ is in the inductive theory of E. The quotient set $\mathcal{T}(\mathcal{X})/=_E$ is a *free algebra* in the class of models of E and $\mathcal{T}(\emptyset)/=_E$ 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 procedure 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 $E \models (\exists x)s = t$, 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

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

$$\frac{x}{y} \quad = \quad \frac{x - y}{y} + 1 \qquad \text{only if } y \neq 0$$

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 $l \to r$ with a condition C is written $C \mid l \to r$, as in

$$y \neq 0 \mid \quad \frac{x}{y} \quad \to \quad \frac{x-y}{y} + 1$$

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 formulæ, 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,

$$\begin{array}{lrcl} (x > y) = T & \mid \quad max(x,y) & \to & x \\ (x > y) = F & \mid \quad max(x,y) & \to & y \\ & s(x) > s(y) & \to & x > y \\ & s(x) > 0 & \to & T \\ & 0 > x & \to & F \end{array}$$

define maximum and greater-than for tally numbers $s^i(0)$. 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

$$u_1 = v_1 \wedge \cdots \wedge u_n = v_n \;\Rightarrow\; l = r$$

In a rule of form $C \mid l \to r$, we call C the *premiss* and $l \to r$ 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 $\to$, 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

$$u_1 \downarrow v_1 \wedge \cdots \wedge u_n \downarrow v_n \;\mid\; l \to r$$

intending that terms $u[l\sigma]$, containing an instance of left-hand side l, rewrite to $u[r\sigma]$ only when all the conditions $u_i\sigma \downarrow v_i\sigma$ hold for that substitution σ. 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

$$\begin{array}{rcl|rcl} null(\epsilon) & \to & T & head(x:y) & \to & x \\ null(x:y) & \to & F & tail(x:y) & \to & y \\ append(\epsilon, y) & \to & y & & & \\ null(x) \downarrow F \;\mid\; append(x,y) & \to & head(x) : append(tail(x), y) & & & \end{array}$$

and its derivation

$$append(a:(b:\epsilon), c:\epsilon) \;\to^*\; a:(b:(c:\epsilon))$$

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 $t_1 \to_R t_2 \to_R t_3 \cdots$. 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 $s \to_R t$, it may be necessary to determine whether $u \to_R v$ 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 $s \to_R t$ 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 $>$ be a reduction ordering. The conditional equation $c\mu \wedge p\mu \Rightarrow s\mu[r\mu] = t\mu$ is a *conditional critical pair* of the conditional equations $c \Rightarrow l = r$ and $p \Rightarrow s = t$, if l unifies via most general unifier μ with a nonvariable subterm of s, the ordering is such that $l\mu \not< c\mu$, $s\mu \not< p\mu$, $l\mu \not< r\mu$, $s\mu \not< t\mu$, and $c\mu \wedge p\mu$ is satisfiable in E. A conditional critical pair $p \Rightarrow s = t$ is *joinable* if $s\sigma$ and $t\sigma$ are joinable for all σ satisfying p.

By $l\mu \not< c\mu$, we mean that no term in $c\mu$ is always greater than $l\mu$. We use $\mathbf{cp}(E)$ to denote the set of conditional critical pairs so defined.

If $c \mid l \to r$ and $p \mid s \to t$ are two different rules and l and s unify with most general unifier σ, then the critical pair $p\sigma \wedge c\sigma \Rightarrow r\sigma = t\sigma$ 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. $l\sigma > c\sigma$ for each rule $c \mid l \rightarrow r$ in R and substitution σ.

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) iff there is a rewrite proof of $s\sigma = t\sigma$ for each critical pair $c \Rightarrow s = t$ and substitution σ 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, Kounalis 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:

$$\textbf{Superpose:} \qquad \frac{E}{E,\ e} \qquad \text{if } \ e \in \mathbf{cp}(E)$$

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 it is potentially the largest term involved. Note that the two conditional equations may actually be renamed variants of one and the same equation.

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

$$\textbf{Narrow:} \quad \frac{E,\ q \wedge s[l'] = t \Rightarrow u = v}{\begin{array}{c} E,\ q \wedge s[l] = t \Rightarrow u = v, \\ p\mu \wedge q\mu \wedge s[r]\mu = t\mu \Rightarrow u\mu = v\mu \end{array}} \quad \text{if} \quad \begin{cases} p \Rightarrow l' = r \in E, \\ l \text{ is not a variable} \\ \mu = \mathrm{mgu}(l, l') \\ s[l]\mu \not\succ p\mu, q\mu, t\mu, s[r]\mu \end{cases}$$

The condition $s\mu \not\succ p\mu$ means that $s\mu$ is not smaller than any side of any equation in $p\mu$. Whenever this or subsequent rules refer to a conditional equation like $q \wedge s = t \Rightarrow u = v$, 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 ($x = x$):

$$\textbf{Reflect:} \quad \frac{E,\ q \wedge s = t \Rightarrow u = v}{\begin{array}{c} E,\ q \wedge s = t \Rightarrow u = v, \\ q\mu \Rightarrow u\mu = v\mu \end{array}} \quad \text{if} \begin{cases} \mu = \text{mgu}(s,t) \\ s\mu \not\succ q\mu \end{cases}$$

The contraction rules all simplify the set of conditional equations:

$$\frac{E,\ q \Rightarrow u = u}{E} \qquad \frac{E,\ q \wedge s = s \Rightarrow u = v}{E,\ q \Rightarrow u = v}$$

$$\frac{E,\ q[l\sigma] \Rightarrow u = v}{E,\ q[r\sigma] \Rightarrow u = v} \quad \text{if} \ \ p \Rightarrow l = r \in E,\ l\sigma \succ r\sigma, p\sigma,\ E \models p\sigma$$

$$\frac{E,\ q \Rightarrow u[l\sigma] = v}{E,\ q \Rightarrow u[r\sigma] = v} \quad \text{if} \begin{cases} p \Rightarrow l = r \in E,\ l\sigma \succ r\sigma, p\sigma,\ E \models p\sigma \\ v \succ u[l\sigma] \vee (v \succ r \wedge l \not\sim u[l\sigma]) \end{cases}$$

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 $l \not\sim u$ we mean that l is not a variant of u. 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 $>$ be any complete simplification ordering extending the given partial ordering $\succ$. 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:

$$s = t_1 \ \underset{|}{\longleftrightarrow}_{P_1} \ t_2 \ \underset{|}{\longleftrightarrow}_{P_2} \ \cdots \ \underset{|}{\longleftrightarrow}_{P_m} \ t_{m+1} = t$$

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 σ_k for variables in the equation, and subproofs P_k (of the same form) for each conditions $u_{k,j}\sigma_k = v_{k,j}\sigma_k$ of the applied instance $e_k\sigma_k$. 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 $s \leftrightarrow^*_{E_0} t$, there eventually exists an unconditional valley proof $s \downarrow_{E_k} t$. 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 1991]).* An inference sequence $E_0 \vdash E_1 \vdash \cdots$ is *fair* with respect to the unit strategy if

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

where E_∞ is the set of *persisting* conditional equations and $\mathbf{cp}^1(E_\infty)$ 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 1991). *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_∞.*

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 $s = w[l\sigma] \leftrightarrow w[r\sigma] = t$ in a ground proof or its subproofs, using a conditional equation $q \Rightarrow l = r$, with $s \geq t$ (in the complete simplification ordering extending $>$). To each such step, we assign the weight $\langle [q\sigma, s, l\sigma], q \Rightarrow l = r \rangle$ 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 $l\sigma$, and that for decreasing instances it is also greater than $q\sigma$.) Second components are compared

using the extension $\rhd$ of the encompassment ordering described earlier. Proofs are compared in the well-founded bag extension of the lexicographic ordering on steps. We use $\gg$ 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 $\gg$, the eventual existence of a normal-form proof follows: If $E \vdash E'$, then for any proof P in E of an equation $s = t$, there exists a proof P' in E' of $s = t$, such that $P \underline{\gg} P'$. 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 $\gg$. Furthermore, if P is a non-normal-form proof in E, then there exists a proof P' in $E \cup \mathbf{cp}^1(E)$ such that $P \gg P'$.

If $s = t$ is provable in E_0, then it has a proof P in the limit E_∞. If P is nonnormal, then it admits a smaller proof P' using (in addition to E_∞) a finite number of equations in $\mathbf{cp}^1(E_\infty)$. 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_∞ itself, and hence (by the bag nature of the proof measure), there is a proof of $s = t$ in E_∞ that is strictly smaller than P. Since the ordering on proofs is well-founded, by induction there must be a normal proof in E_∞. □

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 1991*b*]. 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 confluence 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 1991]).* An inference sequence $E_0 \vdash E_1 \vdash \cdots$ is *fair* with respect to the *decreasing strategy* if

$$\mathbf{cp}^+(E_\infty) \quad \subseteq \quad E_0 \cup E_1 \cup \cdots$$

where $\mathbf{cp}^+(E_\infty)$ 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, those were refutationally *in*complete, since they made no provision for unorientable equations $s = t$ such that $s \not\succ t$ and $t \not\succ s$.)

9.10. THEOREM (Dershowitz 1991). *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_∞.*

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:

$$\begin{array}{rcl|rcl} f(x,x) & \to & a & e & \to & c(e) \\ f(x,c(x)) & \to & b & & & \end{array}$$

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* (symbolized $R \perp S$) 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 to appear].

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 $s \to t$ 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

$$\begin{aligned} \mathit{if}(T,x,y) &\to x \\ \mathit{if}(F,x,y) &\to y \end{aligned}$$

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 nonorthogonal 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 $null(\epsilon)$ 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

$$\begin{aligned} p(c) \quad &| \quad a \to b \\ \neg p(c) \quad &| \quad a \to b \end{aligned}$$

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 [Brand et al. 1979, Plaisted 1987, Dershowitz and Plaisted 1988, Dershowitz and Okada 1990].

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

$$\begin{aligned} p(c) = T \quad &| \quad a \to b \\ \neg p(c) = T \quad &| \quad a \to b \end{aligned}$$

In this way, we can convert conditions involving arbitrary formulæ 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 $p \vee \neg q_1 \vee \ldots \vee \neg q_n$ as conditional rewrite rules, $q_1 \downarrow T \wedge \cdots \wedge q_n \downarrow T \mid p \rightarrow T$, 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 495] 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 $p \mid l \rightarrow r$ such that l unifies with a nonvariable (selected) subterm of s via most general unifier μ, then the conditions in $p\mu$ are solved, say via substitution ρ, and the new goal becomes $s\mu\rho =^? t\mu\rho$.

Suppose we wish to solve

$$append(x, y) \quad =^? \quad x$$

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

$$head((u : v), append(tail(u : v), y)) \quad =^? \quad u : v$$

Straightforward rewriting reduces this to

$$u : append(v, y) \quad =^? \quad u : v$$

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 $x \mapsto u : \epsilon$ and $y, v \mapsto \epsilon$.

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, Albert Rubio, 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

Anantharaman S. and Hsiang J. [1990], 'Automated proofs of the Moufang identities in alternative rings', *J. Symbolic Computation* **6**, 79–109.

Arnon D. S., Collins G. E. and McCallum S. [1984], 'Cylindrical algebraic decomposition. I. The basic algorithm', *SIAM J. Comput.* **13**(4), 865–877.

Arts T. and Giesl J. [2000], 'Termination of term rewriting using dependency pairs', *Theoretical Computer Science* **236**, 133–178.

Avenhaus J. and Madlener K. [1989], 'Term rewriting and equational reasoning'. to appear in R.B. Banoji: Formal Techniques in Artificial Intelligence: A Sourcebook.

Avenhaus J. and Madlener K. [1990], Term rewriting and equational reasoning, *in* R. B. Banerji, ed., 'Formal Techniques in Artificial Intelligence: A Sourcebook', Elsevier, Amsterdam, pp. 1–41.

Baader F. and Nipkow T. [1998], *Term Rewriting and All That*, Cambridge University Press, Cambridge, U.K.

Baader F. and Snyder W. [2001], Unification theory, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 8, pp. 445–532.

Bachmair L. and Dershowitz N. [1986], Commutation, transformation, and termination, *in* J. H. Siekmann, ed., 'Proceedings of the Eighth International Conference on Automated Deduction (Oxford, England)', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 5–20.

Bachmair L. and Dershowitz N. [1987], Inference rules for rewrite-based first-order theorem proving, *in* 'Proceedings of the Second IEEE Symposium on Logic in Computer Science', Ithaca, NY, pp. 331–337.

BACHMAIR L. AND DERSHOWITZ N. [1988], 'Critical pair criteria for completion', *J. Symbolic Computation* **6**(1), 1–18.

BACHMAIR L. AND DERSHOWITZ N. [1989], 'Completion for rewriting modulo a congruence', *Theoretical Computer Science* **67**(2 & 3).

BACHMAIR L. AND DERSHOWITZ N. [1994], 'Equational inference, canonical proofs, and proof orderings', *J. of the Association for Computing Machinery* **41**(2), 236–276.

BACHMAIR L., DERSHOWITZ N. AND PLAISTED D. A. [1989], Completion without failure, *in* H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures', Vol. 2: Rewriting Techniques, Academic Press, New York, chapter 1, pp. 1–30.

BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

BACHMAIR L. AND PLAISTED D. [1985], 'Termination orderings for associative-commutative rewriting systems', *J. Symbolic Computation* **1**, 329–349.

BAIRD T., PETERSON G. AND WILKERSON R. [1989], Complete sets of reductions modulo associativity, commutativity, and identity, *in* 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 29–44.

BALLANTYNE A. M. AND LANKFORD D. S. [1981], 'New decision algorithms for finitely presented commutative semigroups', *J. Computational Mathematics with Applications* **7**, 159–165.

BECKER T. AND WEISPFENNING V. [1993], *Gröbner Bases: A Computational Approach to Commutative Algebra*, Vol. 141 of *Graduate Texts in Mathematics*, Springer-Verlag, Berlin.

BELLEGARDE F. AND LESCANNE P. [1990], 'Termination by completion', *Applied Algebra on Engineering, Communication and Computer Science* **1**(2), 79–96.

BEN CHERIFA A. AND LESCANNE P. [1987], 'Termination of rewriting systems by polynomial interpretations and its implementation', *Science of Computer Programming* **9**(2), 137–159.

BENNINGHOFEN B., KEMMERICH S. AND RICHTER M. M. [1987], *Systems of Reductions*, Vol. 277 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

BERGSTRA J. A. AND KLOP J. W. [1986], 'Conditional rewrite rules: Confluency and termination', *J. of Computer and System Sciences* **32**, 323–362.

BERTLING H. AND GANZINGER H. [1989], Completion-time optimization of rewrite-time goal solving, *in* N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 45–58.

BIRKHOFF G. [1935], 'On the structure of abstract algebras', *Proceedings of the Cambridge Philosophical Society* **31**, 433–454.

BOFILL M., GODOY G., NIEUWENHUIS R. AND RUBIO A. [1999], Paramodulation with nonmonotonic orderings, *in* 'Proc. 14th Annual IEEE Symposium on Logic in Computer Science (Trento, Italy)', IEEE Comp. Sci. Press, pp. 225–234.

BOOK R. V. AND OTTO F. [1993], *String-Rewriting Systems*, Springer-Verlag.

BORRALLERAS C., FERREIRA M. AND RUBIO A. [2000], Complete monotonic semantic path orderings, *in* 'Proceedings of the International Conference on Automated Deduction', pp. 346–364.

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.

BREAZU-TANNEN V. AND GALLIER J. [to appear], 'Polymorphic rewriting conserves algebraic strong normalization', *Theoretical Computer Science* .

BROWN, JR. T. C. [1975], A Structured Design-Method for Specialized Proof Procedures, PhD thesis, California Institute of Technology, Pasadena, CA.

BURRIS S. AND LAWRENCE J. [1991], 'Term rewrite rules for finite fields', *International Journal of Algebra and Computation* **1**(3), 353–369.

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.

CHEW P. [1980], An improved algorithm for computing with equations, *in* 'Proceedings of the Twenty First Conference on Foundations of Computer Science', Los Angeles, CA, pp. 108–117.

CHEW P. [1981], Unique normal forms in term rewriting systems with repeated variables, *in* 'Proceedings of the 13th Annual Symposium on the Theory of Computing', pp. 7–18.

CHRISTIAN J. [1989], Fast Knuth-Bendix completion: Summary, *in* N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 551–555.

CICHON A. AND MARION J.-Y. [1999], Light LPO, Rapport de Recherche 99-R-138, Laboratoire Lorrain de Réchèrche en Informatique et ses Applications.

CIRSTEA H. AND KIRCHNER C. [1999], Combining higher-order and first-order computation using ρ-calculus: Towards a semantics of ELAN, *in* 'Frontiers of Combining Systems 2', Studies in Logic and Computation, Research Studies Press, Baldock, England, pp. 95–119.

COMON H. [1990], Solving inequations in term algebras (Preliminary version), *in* 'Proceedings of the Fifth Annual Symposium on Logic in Computer Science', IEEE, Philadelphia, PA, pp. 62–69.

COMON H. [2001], Inductionless induction, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 14, pp. 913–962.

COMON H., NARENDRAN P., NIEUWENHUIS R. AND RUSINOWITCH M. [1998], Decision problems in ordered rewriting, *in* 'Proc. of the 13th Annual IEEE Symposium on Logic in Computer Science', Indianapolis, IN, pp. 276–286.

COMON H. AND TREINEN R. [1997], 'The first-order theory of lexicographic path orderings is undecidable', *Theoretical Computer Science* **176**(1–2), 67–87.

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

DAUCHET M., TISON S., HEUILLARD T. AND LESCANNE P. [1987], Decidability of the confluence of ground term rewriting systems, *in* 'Proceedings of the Second Symposium on Logic in Computer Science', pp. 353–359.

DERSHOWITZ N. [1979], 'A note on simplification orderings', *Information Processing Letters* **9**(5), 212–215.

DERSHOWITZ N. [1981], Termination of linear rewriting systems, *in* 'Proceedings of the Eighth International Colloquium on Automata, Languages and Programming (Acre, Israel)', Vol. 115 of *Lecture Notes in Computer Science*, European Association of Theoretical Computer Science, Springer-Verlag, Berlin, pp. 448–458.

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

DERSHOWITZ N. [1985], 'Computing with rewrite systems', *Information and Control* **64**(2/3), 122–157.

DERSHOWITZ N. [1987], 'Termination of rewriting', *J. Symbolic Computation* **3**(1&2), 69–115. Corrigendum: *4*, 3 (December 1987), 409–410.

DERSHOWITZ N. [1991*a*], Canonical sets of Horn clauses, *in* J. L. Albert, B. Monien and M. R. Artalejo, eds, 'Proceedings of the Eighteenth International Colloquium on Automata, Languages and Programming (Madrid, Spain)', Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 267–278.

DERSHOWITZ N. [1991*b*], Ordering-based strategies for Horn clauses, *in* 'Proceedings of the Twelfth International Joint Conference on Artificial Intelligence', Sydney, Australia, pp. 118–124.

DERSHOWITZ N. [1995], Hierarchical termination, *in* N. Dershowitz and N. Lindenstrauss, eds, 'Proceedings of the Fourth International Workshop on Conditional and Typed Rewriting Systems (Jerusalem, Israel, July 1994)', Vol. 968 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 89–105.

DERSHOWITZ N. AND HOOT C. [1995], 'Natural termination', *Theoretical Computer Science* **142**(2), 179–207.

DERSHOWITZ N., HSIANG J., JOSEPHSON N. AND PLAISTED D. A. [1983], Associative-commutative rewriting, *in* 'Proceedings of the Eighth International Joint Conference on Artificial Intelligence', pp. 940–944.

DERSHOWITZ N., HSIANG J. AND SHIENG G.-S. [2000], 'Using Boolean rings and simplification to test satisfiability'. Submitted.

DERSHOWITZ N. AND JOUANNAUD J.-P. [1990], Rewrite systems, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 6, pp. 243–320.

DERSHOWITZ N., KAPLAN S. AND PLAISTED D. A. [1991], 'Rewrite, rewrite, rewrite, rewrite, rewrite,...', *Theoretical Computer Science* **83**(1), 71–96.

DERSHOWITZ N. AND MANNA Z. [1979], 'Proving termination with multiset orderings', *Communications of the ACM* **22**(8), 465–476.

DERSHOWITZ N., MARCUS L. AND TARLECKI A. [1988], 'Existence, uniqueness, and construction of rewrite systems', *SIAM J. on Computing* **17**(4), 629–639.

DERSHOWITZ N. AND OKADA M. [1988], Proof-theoretic techniques and the theory of rewriting, *in* 'Proceedings of the Third IEEE Symposium on Logic in Computer Science', Edinburgh, Scotland, pp. 104–111.

DERSHOWITZ N. AND OKADA M. [1990], 'A rationale for conditional equational programming', *Theoretical Computer Science* **75**, 111–138.

DERSHOWITZ N., OKADA M. AND SIVAKUMAR G. [1987], Confluence of conditional rewrite systems, *in* S. Kaplan and J.-P. Jouannaud, eds, 'Proceedings of the First International Workshop on Conditional Term Rewriting Systems (Orsay, France)', Vol. 308 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 31–44.

DERSHOWITZ N., OKADA M. AND SIVAKUMAR G. [1988], Canonical conditional rewrite systems, *in* 'Proceedings of the Ninth Conference on Automated Deduction (Argonne, IL)', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 538–549.

DERSHOWITZ N. AND PLAISTED D. A. [1988], Equational programming, *in* J. E. Hayes, D. Michie and J. Richards, eds, 'Machine Intelligence 11: The logic and acquisition of knowledge', Oxford Press, Oxford, chapter 2, pp. 21–56.

DERSHOWITZ N. AND REDDY U. [1993], 'Deductive and inductive synthesis of equational programs', *J. Symbolic Computation* **15**, 467–494.

DETLEFS D. AND FORGAARD R. [1985], A procedure for automatically proving the termination of a set of rewrite rules, *in* J.-P. Jouannaud, ed., 'Proceedings of the First International Conference on Rewriting Techniques and Applications (Dijon, France)', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 255–270.

DEVIE H. [1990], When ordered completion fails, *in* M. Okada, ed., 'Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems (Montreal, Canada)', Vol. 516 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

DICK J., KALMUS J. AND MARTIN U. [1990], 'Automating the Knuth Bendix ordering', *Acta Informatica* **28**(2), 95–119.

DOWES G., HARDIN T. AND KIRCHNER C. [1994], 'Higher-order unification via explicit substitutions'.

EDELSON R. [1990]. Private Communication.

EVANS T. [1951], 'On multiplicative systems defined by generators and relations, I', *Proceedings of the Cambridge Philosophical Society* **47**, 637–649.

FERREIRA M. C. F. AND ZANTEMA H. [1993], Total termination of term rewriting, *in* C. Kirchner, ed., 'Proceedings of the Fifth Conference on Rewriting Techniques and Applications', Vol. 690 of *Lecture Notes in Computer Science*, Springer, pp. 213–227.

FORET A. [1988], Rewrite rule systems for modal propositional logic, *in* 'Proceedings of an International Workshop on Algebraic and Logic Programming', Akademie-Verlag, Gaussig, GDR, pp. 146–157.

FRIBOURG L. [1985], SLOG: A logic programming language interpreter based on clausal superposition and rewriting, *in* 'Proceedings of the Symposium on Logic Programming', IEEE, Boston, MA, pp. 172–184.

GALLIER J. [1991], 'What's so special about Kruskal's Theorem and the ordinal Γ_0. A survey of some results in proof theory', *Annals of Pure and Applied Logic* **53**(3), 199–260.

GANZINGER H. [1991], 'A completion procedure for conditional equations', *Journal of Symbolic Computation* **11**, 51–81.

GENET T. AND GNAEDIG I. [1997], Termination proofs using GPO ordering constraints, *in* M. Dauchet, ed., 'Proceedings 22nd International Colloquium on Trees in Algebra and Programming (Lille, France)', Vol. 1214 of *Lecture Notes in Computer Science*, pp. 249–260.

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

GIESL J. [1995], Generating polynomial orderings for termination proofs, *in* 'Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (Kaiserslautern, Germany)', Vol. 914 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 426–431.

GIOVANNETTI E. AND MOISO C. [1987], A completeness result for conditional narrowing, *in* 'Presented at the First International Workshop on Conditional Term Rewriting Systems', Orsay, France.

GOGUEN J. A., KIRCHNER C. AND MESEGUER J. [1987], Concurrent term rewriting as a model of computation, *in* R. Keller and J. Fasel, eds, 'Proceedings of Graph Reduction Workshop (Santa Fe, NM)', Vol. 279 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 53–93.

GOGUEN J. A. AND MESEGUER J. [1986], EQLOG: Equality, types, and generic modules for logic programming, *in* D. DeGroot and G. Lindstrom, eds, 'Logic Programming: Functions, Relations, and Equations', Prentice-Hall, Englewood Cliffs, NJ, pp. 295–363.

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

GRAMLICH B. AND WIRTH C.-P. [1996], Confluence of terminating conditional rewrite systems revisited, *in* H. Ganzinger, ed., 'Proceedings of the International Conference on Rewriting Techniques and Applications', Vol. 1103 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, USA, pp. 245–259.

GUTTAG J. V., KAPUR D. AND MUSSER D. R. [1983], 'On proving uniform termination and restricted termination of rewriting systems', *SIAM J. on Computing* **12**(1), 189–214.

HANUS M. [1994], 'The integration of functions into logic programming: From theory to practice', *J. Logic Programming* **19&20**, 583–628.

HANUS M. [1995], On extra variables in (equational) logic programming, *in* 'Proc. Twelfth International Conference on Logic Programming', MIT Press, pp. 665–679.

HIGMAN G. [1952], 'Ordering by divisibility in abstract algebras', *Proceedings of the London Mathematical Society (3)* **2**(7), 326–336.

HIGMAN G. AND NEUMANN B. [1952], 'Groups as groupoids with one law', *Publ. Math. Debrecen* **2**, 215–221.

HINDLEY J. R. [1964], The Church-Rosser Property and a Result in Combinatory Logic, PhD thesis, University of Newcastle-upon-Tyne.

HOFBAUER D. [1992], 'Termination proofs by multiset path orderings imply primitive recursive derivation lengths', *Theoretical Computer Science* **105**(1), 129–140.

HOFBAUER D. AND LAUTEMANN C. [1989], Termination proofs and the length of derivations (preliminary version), *in* 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 167–177.

HOFFMANN C. M. AND O'DONNELL M. J. [1982], 'Programming with equations', *ACM Transactions on Programming Languages and Systems* **4**(1), 83–112.

HSIANG J. [1985], 'Refutational theorem proving using term-rewriting systems', *Artificial Intelligence* **25**, 255–300.

HSIANG J. AND DERSHOWITZ N. [1983], Rewrite methods for clausal and non-clausal theorem proving, *in* 'Proceedings of the Tenth International Colloquium on Automata, Languages and Programming (Barcelona, Spain)', Vol. 154 of *Lecture Notes in Computer Science*, European Association of Theoretical Computer Science, Springer-Verlag, Berlin, pp. 331–346.

HSIANG J. AND HUANG G. S. [1996], Some fundamental properties of Boolean ring normal forms, *in* D. Du, J. Gu and P. M. Pardalos, eds, 'Satisfiability Problem: Theory and Applications', Vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, NSF Science and Technology Center, American Mathematical Society, pp. 587–602.

HSIANG J. AND RUSINOWITCH M. [1987], On word problems in equational theories, *in* T. Ottmann, ed., 'Proceedings of the Fourteenth EATCS International Conference on Automata, Languages and Programming (Karlsruhe, West Germany)', Vol. 267 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 54–71.

HSIANG J. AND RUSINOWITCH M. [1991], 'Proving refutational completeness of theorem proving strategies. The transfinite semantic tree method', *J. of the Association for Computing Machinery* **38**(3), 559–587.

HUET G. [1980], 'Confluent reductions: Abstract properties and applications to term rewriting systems', *J. of the Association for Computing Machinery* **27**(4), 797–821.

HUET G. [1981], 'A complete proof of correctness of the Knuth-Bendix completion algorithm', *J. Computer and System Sciences* **23**(1), 11–21.

HUET G. [1985], Cartesian closed categories and Lambda-calculus, *in* 'Proceedings of the LITP Spring School on Combinators and Functional Programming Languages', Vol. 242 of *Lecture Notes in Computer Science*, Springer-Verlag, Val d'Ajol, France, pp. 123–135.

HUET G. AND LANKFORD D. S. [1978], On the uniform halting problem for term rewriting systems, Rapport laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France.

HUET G. AND LÉVY J.-J. [1991], Computations in orthogonal rewriting systems, I and II, *in* J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA, pp. 395–443.

HULLOT J.-M. [1980], A catalogue of canonical term rewriting systems, Technical Report CSL-113, SRI International, Menlo Park, CA.

JOUANNAUD J.-P. AND KIRCHNER H. [1986], 'Completion of a set of rules modulo a set of equations', *SIAM J. on Computing* **15**, 1155–1194.

JOUANNAUD J.-P. AND LESCANNE P. [1982], 'On multiset orderings', *Information Processing Letters* **15**, 57–63.

JOUANNAUD J.-P., LESCANNE P. AND REINIG F. [1982], Recursive decomposition ordering, *in* D. Bjørner, ed., 'Proceedings of the Second IFIP Workshop on Formal Description of Programming Concepts', North-Holland, Garmisch-Partenkirchen, West Germany, pp. 331–348.

JOUANNAUD J.-P. AND OKADA M. [1991], Satisfiability of systems of ordinal notations with the subterm property is decidable, *in* J. L. Albert, B. Monien and M. R. Artalejo, eds, 'Proceedings of the Eighteenth EATCS Colloquium on Automata, Languages and Programming (Madrid, Spain)', Vol. 510 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 455–468.

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.

KAPLAN S. [1987], 'Simplifying conditional term rewriting systems: Unification, termination and confluence', *J. Symbolic Computation* **4**(3), 295–334.

KAPLAN S. [1988], 'Rewriting with a nondeterministic choice operator', *Theoretical Computer Science* **56**, 37–57.

KAPUR D., MUSSER D. AND NARENDRAN P. [1988], 'Only prime superpositions need be considered in the Knuth-Bendix procedure', *Journal of Symbolic Computation* **6**(1), 19–36.

KAPUR D. AND NARENDRAN P. [1985*a*], An equational approach to theorem proving in first-order predicate calculus, *in* 'Proceedings of the Ninth International Joint Conference on Artificial Intelligence', Los Angeles, CA, pp. 1146–1153.

KAPUR D. AND NARENDRAN P. [1985*b*], 'A finite Thue system with decidable word problem and without equivalent finite canonical system', *Theoretical Computer Science* **35**, 337–344.

KAPUR D., NARENDRAN P. AND OTTO F. [1990], 'On ground confluence of term rewriting systems', *Information and Computation* **86**(1), 14–31.

KAPUR D. AND SIVAKUMAR G. [1997], A total, ground path ordering for proving termination of AC-rewrite systems, *in* 'Proceedings of the Eighth International Conference on Rewriting Techniques and Applications (Barcelona, Spain)', Vol. 1232 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 142–156.

KAPUR D. AND ZHANG H. [1989], A case study of the completion procedure: Proving ring commutativity problems. Unpublished Draft.

KENNAWAY R., KLOP J. W., SLEEP R. AND DE VRIES F.-J. [1995], 'Transfinite reductions in orthogonal term rewriting systems', *Information and Computation* **119**(1), 18–38.

KIRCHNER C., KIRCHNER H. AND RUSINOWITCH M. [1990], 'Deduction with symbolic constraints', *RAIRO Theoretical Informatics and Applications* **4**(3), 9–52. Special issue on Automatic Deduction.

KIRCHNER H. AND HERMANN M. [1990], Meta-rule synthesis from crossed rewrite systems, *in* '2nd International Workshop on Conditional and Typed Rewriting Systems (Montreal, Canada)', Vol. 516 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 143–154.

KLOP J. W. [1980], *Combinatory Reduction Systems*, Centre for Mathematics and Computer Science, Amsterdam. Mathematical Centre Tracts No. 127.

KLOP J. W. [1992], Term rewriting systems, *in* S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, eds, 'Handbook of Logic in Computer Science', Vol. 2, Oxford University Press, Oxford, chapter 1, pp. 1–117.

KNUTH D. E. AND BENDIX P. B. [1970], Simple word problems in universal algebras, *in* J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford, U. K., pp. 263–297. Reprinted in *Automation of Reasoning 2*, Springer-Verlag, Berlin, pp. 342–376 (1983).

KOROVIN K. AND VORONKOV A. [2000], Knuth-Bendix constraint solving is NP-complete, Preprint CSPP-8, Department of Computer Science, University of Manchester.

KOROVIN K. AND VORONKOV A. [2001], Verifying orientability of rewrite rules using the Knuth-Bendix order, *in* 'Proceedings of the Twelfth International Conference on Rewriting Techniques and Applications', Lecture Notes in Computer Science, Springer-Verlag.

KOUNALIS E. AND RUSINOWITCH M. [1988], On word problems in Horn theories, *in* E. Lusk and R. Overbeek, eds, 'Proceedings of the Ninth International Conference on Automated Deduction (Argonne, Illinois)', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 527–537.

KRISHNAMOORTHY M. S. AND NARENDRAN P. [1985], 'On recursive path ordering', *Theoretical Computer Science* **40**, 323–328.

KRUSKAL J. [1960], 'Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture', *Transactions of the American Mathematical Society* **95**, 210–225.

KURIHARA M. AND KAJI I. [1990], 'Modular term rewriting systems and the termination', *Information Processing Letters* **34**, 1–4.

LANKFORD D., BUTLER G. AND BALLANTYNE A. [1984], A progress report on new decision algorithms for finitely presented Abelian groups, *in* R. E. Shostak, ed., 'Proceedings of the Seventh International Conference on Automated Deduction (Napa, CA)', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 128–141.

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, Mathematics Department, Louisiana Tech. University, Ruston, LA. Revised October 1979.

LANKFORD D. S. [1980], The uniform word problem for J-algebras is decidable, Memo MTP-10, 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.

LE CHENADEC P. [1985], *Canonical Forms in Finitely Presented Algebras*, Pitman-Wiley, London.

LESCANNE P. [1984], Term rewriting systems and algebra, *in* R. E. Shostak, ed., 'Proceedings of the Seventh International Conference on Automated Deduction (Napa, CA)', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 166–174.

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

MANNA Z. AND NESS S. [1970], On the termination of Markov algorithms, *in* 'Proceedings of the Third Hawaii International Conference on System Science', Honolulu, HI, pp. 789–792.

MARCHIORI M. [1995], Modularity of completeness revisited, *in* 'Proceedings of the Sixth International Conference on Rewriting Techniques and Applications (Kaiserslautern, Germany)', Vol. 914 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 2–10.

MARCHIORI M. [1996], Unravelings and ultra-properties, *in* 'Proc. Algebraic and Logic Programming', Vol. 1139 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 107–121.

MARTIN U. [1989], 'A geometrical approach to multiset orderings', *Information Processing Letters* **67**, 37–54.

MARTIN U. AND NIPKOW T. [1990], Ordered completion, *in* M. Stickel, ed., 'Proceedings of the Tenth International Conference on Automated Deduction (Kaiserslautern, West Germany)', Vol. 449 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 366–380.

MCCUNE W. [1997], 'Solution of the Robbins problem', *J. Automated Reasoning* **19**(3), 263–276.

MCCUNE W. W. [1989], *Otter 1.0 Users' Guide*, Mathematics and Computer Science Division, Aggonne National Laboratory, Argonne, Illinois.

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

MUSSER D. [1980], On proving inductive properties of abstract data types, *in* 'Proceedings, 7th ACM Symposium on Principles of Programming Languages', pp. 154–162.

NAOI T. AND INAGAKI Y. [1989], Algebraic semantics and complexity of term rewriting systems, *in* 'Proceedings of the 3rd International Conference on rewriting techniques and applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 311–325.

NARENDRAN P., RUSINOWITCH M. AND VERMA R. [1998], RPO constraint solving is in NP, *in* 'Proceedings of the Annual Conference of the European Association of Computer Science Logic (Brno, Czech Republic)', Vol. 1584 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 385–398.

NASH-WILLIAMS C. S. J. A. [1963], 'On well-quasi-ordering finite trees', *Proceedings of the Cambridge Philosophical Society* **59**(4), 833–835.

NELSON C. G. AND OPPEN D. C. [1980], 'Fast decision procedures based on congruence closure', *J. of the Association for Computing Machinery* **27**(2), 356–364.

NEWMAN M. H. A. [1942], 'On theories with a combinatorial definition of 'equivalence'', *Annals of Mathematics* **43**(2), 223–243.

NIEUWENHUIS R. [1993], 'Simple LPO constraint solving methods', *Information Processing Letters* **47**(2).

NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.

NIPKOW T. [1990], 'Unification in primal algebras, their powers and their varieties', *J. of the Association for Computing Machinery* **37**, 742–776.

O'DONNELL M. J. [1977], *Computing in systems described by equations*, Vol. 58 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

OHLEBUSCH E. [1999], On quasi-reductive and quasi-simplifying deterministic conditional rewrite systems, *in* 'Proceedings of the 4th International Symposium on Functional and Logic Programming', Vol. 1722 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 179=–193.

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

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

PEDERSEN J. [1985*a*], Obtaining complete sets of reductions and equations without using special unification algorithms, *in* B. F. Caviness, ed., 'Proceedings of the European Conference on Computer Algebra: Research Contributions (Linz, Austria)', Vol. 204 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 422–423.

PEDERSEN J. [1985*b*], 'The word problem in absorbing varieties', *Houston Journal of Mathematics* **11**(4), 575–590.

PEDERSEN J. [1988], Computer solution of word problems in universal algebra, *in* M. Tangora, ed., 'Computers in Algebra', Vol. 111 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 103–128.

PEDERSEN J. [1989], Morphocompletion for one-relation monoids, *in* N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 574–578.

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

PETERSON G. [1990], Complete sets of reductions with constraints, *in* M. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', pp. 381–395.

PETERSON G. E. AND STICKEL M. E. [1981], 'Complete sets of reductions for some equational theories', *J. of the Association for Computing Machinery* **28**(2), 233–264.

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.

PLAISTED D. [1986], A simple non-termination test for the Knuth-Bendix method, *in* 'Proceedings of the Eighth International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, pp. 79–88.

PLAISTED D. [1987], A logic for conditional term rewriting systems, *in* S. Kaplan and J.-P. Jouannaud, eds, 'Proceedings of the First International Workshop on Conditional Term Rewriting Systems (Orsay, France)', Vol. 308 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 212–227.

PLAISTED D. A. [1985], 'The undecidability of self embedding for term rewriting systems', *Information Processing Letters* **20**(2), 61–64.

PLAISTED D. A. [1993], Equational reasoning and term rewriting systems, *in* D. Gabbay, C. Hogger, J. A. Robinson and J. Siekmann, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Vol. 1, Oxford University Press, Oxford, chapter 5, pp. 273–364.

PORAT S. AND FRANCEZ N. [1985], Fairness in term rewriting systems, *in* J.-P. Jouannaud, ed., 'Proceedings of the First International Conference on Rewriting Techniques and Applications

(Dijon, France)', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 287–300.

RAOULT J.-C. AND VUILLEMIN J. [1980], 'Operational and semantic equivalence between recursive programs', *J. of the Association for Computing Machinery* **27**(4), 772–796.

REDDY U. S. [1986], On the relationship between logic and functional languages, *in* D. DeGroot and G. Lindstrom, eds, 'Logic Programming: Functions, Relations, and Equations', Prentice-Hall, Englewood Cliffs, NJ, pp. 3–36.

RENEGAR J. [1992], 'On the computational complexity and geometry of the first-order theory of the reals. III. Quantifier elimination', *J. Symbolic Computation* **13**(3), 329–352.

ROSEN B. K. [1973], 'Tree-manipulating systems and Church-Rosser theorems', *J. of the Association for Computing Machinery* **20**(1), 160–187.

RUBIO A. [1999], A fully syntactic AC-RPO, *in* P. Narendran and M. Rusinowitch, eds, 'Proceedings of the International Conference on Rewriting Techniques and Applications', Vol. 1631 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 133–147.

RUBIO A. AND NIEUWENHUIS R. [1993], A precedence-based total AC-compatible ordering, *in* C. Kirchner, ed., 'Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (Montreal, Canada)', Vol. 690 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

RUSINOWITCH M. [1987], 'Path of subterms ordering and recursive decomposition ordering revisited', *J. Symbolic Computation* **3**(1&2), 117–132.

SIEKMANN J. AND SZABO P. [1982], 'A Noetherian and confluent rewrite system for idempotent semigroups', *Semigroup Forum* **25**(1/2), 83–110.

SMULLYAN R. [1985], *To Mock a Mockingbird*, Knopf.

SNYDER W. [1989], Efficient ground completion: An $O(n \log n)$ algorithm for generating reduced sets of ground rewrite rules equivalent to a set of ground equations e, *in* N. Dershowitz, ed., 'Proceedings of the Third International Conference on Rewriting Techniques and Applications (Chapel Hill, NC)', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 419–433.

STEINBACH J. [1994], 'Generating polynomial orderings', *Information Processing Letters* **49**(2), 85–93.

STONE M. [1936], 'The theory of representations for Boolean algebra', *Transactions of the American Mathematical Society* **40**, 37–111.

TARSKI A. [1951], *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, CA.

THOMAS W. [1990], Automata on infinite objects, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 4, pp. 133–191.

THUE A. [1914], 'Probleme über veranderungen von zeichenreihen nach gegeben regeln', *Skr. Vid. Kristianaia I. Mat. Naturv. Klasse* **10/34**.

TISON S. [1989], Fair termination is decidable for ground systems, *in* 'Proceedings of the Third International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, pp. 462–476.

TOYAMA Y. [1987], 'On the Church-Rosser property for the direct sum of term rewriting systems', *J. of the Association for Computing Machinery* **34**(1), 128–143.

TOYAMA Y., KLOP J. W. AND BARENDREGT H. P. [1995], 'Termination for direct sums of left-linear complete term rewriting systems', *J. of the Association for Computing Machinery* **42**(6), 1275–1304.

VAN OOSTROM V. [1994], 'Confluence by decreasing diagrams', *Theoretical Computer Science* **126**, 259–280.

VERMA R. M. [1995], 'A theory of using history for equational systems with applications', *J. of the Association for Computing Machinery* **42**(5), 984–1020.

Verma R. M., Rusinowitch M. and Lugiez D. [2001], 'Algorithms and reductions for rewriting systems', *Fundamenta Informaticae* p. to appear.

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'.

Yasuhara A. [1971], *Recursive Function Theory and Logic*, Academic Press.

Zantema H. [1994], 'Termination of term rewriting: Interpretation and type elimination', *J. Symbolic Computation* **17**, 23–50.

Zantema H. [1995], 'Termination of term rewriting by semantic labelling', *Fundamenta Informaticae* **24**, 89–105.

Zhang H. [1994], 'A new method for the Boolean ring based theorem proving', *J. Symbolic Computation* **17**(2), 189–211.

Zhang H. and Kapur D. [1990], 'Unnecessary inferences in associative-commutative completion procedures', *Mathematical Systems Theory* **23**, 175–206.

Zhegalkin I. I. [1927], 'On a technique of evaluation of propositions in symbolic logic', *Matematicheskii Sbornik* **34**(1), 9–27.

Index

H

I

J

K

L

M

N

O

P