Simplifying the Knuth-Bendix Completion Procedure for Equational Systems

Adam Steele

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

June, 1989

ISBN 0-315-51359-4

ABSTRACT

Simplifying the Knuth-Bendix Completion Procedure for Equational Systems

Adam Steele

The Knuth-Bendix completion procedure takes a set of equations and attempts to produce a set of convergent rewrite rules (oriented equations) that has the same power with respect to provability as the original system of equations. We will formulate the completion process as an equational inference system, and show how this process may be associated with a rewriting relation on arbitrary mixed equational and rewrite proofs. The correctness of completion follows from our ability to provide a well-founded ordering for this relation.

The efficiency of the completion process depends critically on the number of rewrite rules generated, and this number may be reduced by the use of so called critical-pair criteria. We will introduce a new generalization of a criterion called compositeness, and show how the framework in which we prove its correctness, proof normalization, will point the way towards techniques for theorem proving in infinite systems of rewrite rules.

Dedication

To my advisor Mitsu, my pal Skippy, and all my infinite friends.

Tom appeared on the sidewalk with a bucket of whitewash.... He surveyed the fence, and all gladness left him and a deep melancholy settled down upon his spirit. Thirty yards of board fence, nine feet high. Life to him seemed hollow, and existence but a burden.

Mark Twain, *The Adventures of Huckleberry Finn*

Table of Contents

Figures

Mathematical Symbols

@	Subterm position indicator
$\rightarrow_R$	Term rewriting relation generated by R
$=>_S$	Proof rewriting relation generated by S
$\leftrightarrow_E$	Term equality relation generated by E
$\downarrow_R$	Joinability within R
$>>$	Multiset ordering
$\geq_s$	Subsumption quasi-ordering

Introduction

1. Equational Theories

Systems of equations have been used to define Abstract Data Types[13, 33], interpreters for non-procedural programming languages[29] and even programs themselves[18]. The advantage of this approach lies in the fact that equational systems combine a high degree of expressive power with a clear, clean semantics (the logical consequences of the system of equations).

Let us consider the example of using a set of equations to define an abstract data type (ADT). A data type is a collection of data domains. basic data elements, and operations such that all data items in the data domains may be generated from operations on the basic data items. An ADT is a class of data types which is closed under the renaming of data domains, items, and operations. and hence is independent of representation.

The fact that an ADT can be defined by set of equations will allow us to prove certain general properties about ADT's. but within the context of this thesis "theorem proving" will have a more specific connotation. The theorems we are attempting to prove are theorems concerning the equivalence of terms under the equational theory (the congruence relation generated by a given system of equations). These equivalences are what will describe the properties of the ADT defined by a set of equations. The most notable of these being the relation between inputs and results that any mechanism implementing the ADT will have to satisfy.

Before we consider the problem of theorem proving within systems of equations let us examine some examples of equational theories. These theories will range in flavour from the mathematical to the computational. and the first theory we will examine is the one generated by the three equations that define a group structure.

$$0 + x = x$$

$$i(x) + x = 0$$

$$(x + y) + z = x + (y + z)$$

The style of equational theorem proving that we are considering will allow us prove theorems such as "$(x + 0) + i(x) = 0 + x$". We may also consider the following equations that define the functions **factorial**, and **append**.

$$\mathbf{fact}(0) = \mathsf{s}(0)$$

$$\mathbf{fact}(\mathsf{s}(x)) = \mathbf{times}(\mathsf{s}(x),\mathbf{fact}(x))$$

$$\mathbf{times}(x,0) = 0$$

$$\mathbf{times}(x,\mathsf{s}(y)) = \mathbf{plus}(\mathbf{times}\ x,y),x)$$

$$\mathbf{plus}(x,0) = x$$

$$\mathbf{plus}(x,\mathsf{s}(y)) = \mathsf{s}(\mathbf{plus}(x,y))$$

The above set of equations is the standard definition for the factorial function in terms of the successor function s(x). In this theory we will be able to prove that **fact(s(s(s(0)))** = s(s(s(s(s(s(0))))))). However, in a computational context we would be more interested in transforming the left-hand side of the equation (the input) into the right-hand side (the result). We shall see how term-rewriting theory very easily accomodates this notion of computation into the general theory of equational theorem proving.

With systems of equations we are not restricted to arithmetic theories, but are free to define theories that meet our particular needs; such as the set of equations below that define the reverse of a list built using the constructor **cons**(x,y).

$$\mathbf{append}(\mathbf{nil},y) = y$$

$$\mathbf{append}(\mathbf{cons}(x,y),z) = \mathbf{cons}(x,\mathbf{append}(y,z))$$

$$\mathbf{reverse}(\mathbf{nil}) = \mathbf{nil}$$

$$\mathbf{reverse}(\mathbf{cons}(x,y)) = \mathbf{append}(\mathbf{reverse}(y),\mathbf{cons}(x,\mathbf{nil}))$$

$$\mathbf{reverse}(\mathbf{reverse}(x)) = x$$

$$\mathbf{car}(\mathbf{cons}(x,y)) = x$$

$$\mathbf{cdr}(\mathbf{cons}(x,y)) = y$$

Despite their benefits, equational systems do, however, have some drawbacks as a framework in which to do practical theorem proving. Equational systems. because of the symmetric nature of their axioms are inherently non-terminating (e.g. consider the sequence $a=b=a=b= \cdots$). Also theorem proving in systems in which the symmetric axiom holds is highly non-deterministic (e.g. in a PROLOG program which has a rule expressing symmetry we may always apply this rule to the current resolvent, opening up a branch of the proof tree that may not satisfy the query; in place of applying a rule which will get us closer to our goal).

In rewrite systems (systems in which term replacement is directed) any sequence of replacements can be shown to terminate if it is possible to embed the reduction ordering into an abstract ordering structure that is known to be well-founded. And. in addition, if the reduction relation $\longrightarrow_R$ is *confluent* (roughly speaking, the property that any rewriting sequence in which there are ambiguities will reduce to the same term). then the normal form of a sequence of reductions (which is guaranteed to exist in a terminating rewrite system) will be unique, and the replacements required to reach this normal form can be carried out deterministically (i.e. without having to consider backtracking to undo failed rewritings).

This normal form generates a class of terms equivalent under interconvertability (bi-directional rewriting); and will allow us to provide a decision procedure for the equivalence of terms in a given equational theory if there exists a finite *convergent* (terminating and confluent) set of rewrite rules corresponding to the original system of equations. We will also attempt to show how the framework which we are using, *proof normalization*, will allow us to consider theorem proving in systems of equations for which the completion procedure does not terminate (i.e. the process generates an infinite number of rewrite rules).

In general, when we are considering term rewriting theory as a computational framework we are usually not interested in testing the equivalence of two arbitrary terms, but rather, in reducing an input term to its normal form. It is this normal form which will define the result of the computation. As noted above, term rewriting systems have the potential for achieving this reduction process in an effective manner (with, as we shall see, the potential for parallel execution), and the additional advantage that the relation between input and result is exactly that of the original equational system.

1.1. The Completion Process

The process of producing a term rewriting system from a system of equations, such that the systems of equations and rewriting rules have the same power with respect to provability is called *completion*. As we saw previously, we can guarantee the termination of a rewriting system by attempting to order the rules of the system with respect to a well-founded ordering.

In his seminal paper[24] Knuth showed that the confluence of a rewriting system could be guaranteed if a certain restricted set of rewriting ambiguities (*critical-pairs*) were *joinable* (i.e. they rewrite to the same term). We can attempt to *complete* non-confluent systems by selecting critical-pairs and attempting to join them with respect to the existing rules of a system; if the pair is not joinable then it is added to the system as a rule to repair confluence. Previous work[26,35] has shown that it is possible to identify certain joinable pairs without having to perform the (expensive) process of reducing them to their normal forms. The framework within which we are describing the completion process allows us to provide an elegant proof of the correctness of our extension to these so-called *critical-pair criteria.*

After introducing some basic definitions in the following section we will prove the correctness of the completion process (including the simplification rules) in a manner similar to that introduced by Bachmair Dershowitz and Hsiang[2]. Using these techniques we prove all the currently existing critical-pair criteria, and our new more generalized criterion *GCC*, correct using the techniques provided by Bachmair and Dershowitz[4].

Proof normalization is more than just a proof technique for studying the completion process. In fact, its true power is only realized when we consider it as a theorem proving technique within a particular equational theory. Theorem proving in finite, convergent rewriting systems (i.e. the completion process has terminated successfully) is achieved by reducing terms to their normal forms with respect to the rules of the system and then comparing these normal forms.

When we move to the case when the completion process does not terminate then we have to consider theorem proving in infinite systems of rewrite rules. The information available to us when we consider theorem proving as an attempt to create a normal proof allows us not only to provide a semi-decision procedure for equality (by attempting to generate a normal rewrite proof between two terms), but in addition, if we consider the structure of the equational system generating the set of rewrite rules, we may be able to determine whether any of the potentially infinite set of equations (generated in the completion process) will be able to produce a rule that will further reduce a given term. With this knowledge we may be able to turn our theorem prover into a decision procedure, since if we have reduced two terms to non-equivalent normal forms, and there are no potential equations that will generate rules which may futher reduce these normal forms, then we can consider the two original terms to be non-equivalent under the equational theory. Finally, we will examine our current state of implementation and examine our future plans.

In our opinion the Knuth-Bendix procedure is important because it has the potential of allowing us to do equational theorem proving in an effective (i.e. terminating and deterministic) fashion. Proving universal theorems in systems of equations corresponds to functional programming. While if we can prove existential theorems (which we can achieve with a technique called *narrowing* in which the left-hand sides of rewrite rules unify rather than match on to subterms of the term being reduced), this corresponds to logic programming.

It is our conviction that term rewriting theory provides the simplest, and most computationally interesting way of unifying these two important paradigms of programming.

Definitions and Basic Concepts

2. Introduction

In this section we provide the basic definitions concerning equational and term rewriting systems that we will be using throughout our exposition. We will start with the basic objects of our theory, terms; and then move on to the congruence relations that will define our notions of equivalence between the terms in this theory. The congruences that we will provide are generated from systems of equations, and systems of rewrite rules on terms and also on proofs. Finally we will define and prove results concerning the two most important concepts of term rewriting theory; *termination* and *confluence.*

2.1. Basic Definitions

We define the set of *terms* $T(F \cup C \cup V)$ over some (finite) set of operator symbols F, constant symbols C, and variable symbols V by the following inductive definition: 1) any variable $v \in V$ is a term, any constant $c \in C$ is also a term; 2) if $f \in F$ is a operator of arity n, and $t_1, \ldots, t_n$ are terms in $T(F \cup C \cup V)$, then $f(t_1, \ldots, t_n)$ is also a term.

We may also consider the set of terms as labelled trees whose leaf nodes have labels which are elements of C and V, and whose internal nodes have labels from the set F. If, from the context, it is clear what the sets F, C and V are, then we denote the set of terms by T alone. We let the symbols $f, g, h, \cdots$ denote operator symbols; the symbols $a, b, c, \cdots$ the constant symbols; the symbols $x, y, z, \cdots$ denote the variables; and the symbols $s, t, u, \cdots$ denote terms.

A *substitution* σ is a mapping from the set V onto the set Γ. We denote $s\sigma$ as the term s with all occurrences of its variables $v_1 \cdots v_n$ uniformly replaced by the terms $t_1 \cdots t_n$ as defined by the substitution. With this notion of binding values to variables with substitutions we can define the *subsumption* quasi-ordering on terms as:

$$s \geq_s t \text{ iff } s = t\sigma$$

We write a term s containing subterm(s) t as $s[t]$, and we say that t occurs at *position(s)* in s (positions are often indicated by the indices of the tree representation of a term). If we wish to specify the term occurring at a position p in the term s we denote this in the following way $s@p$. A subterm t is called *proper* if it is distinct from s. The term $s[u/t]$ denotes the result of replacing a particular occurrence of t by u. A relation R on terms is *monotonic* if sRt implies $u[s]Ru[t]$. R is also called *stable* if sRt implies $s\sigma Rt\sigma$.

With the definitions above we are now in a position to define the relations generated by a set of equations and rewrite rules. An *equation* is pair of terms (s,t), written $s=t$. For any set of equations E we let $\leftrightarrow_E$ denote the smallest symmetric relation containing E that is both stable and monotonic; i.e. $s \leftrightarrow_E t$ iff $s \equiv w[u\sigma]$ and $t \equiv w[v\sigma/u\sigma]$ where $u \doteq v$ is in the set E. (Note: $u \doteq v$ denotes. ambiguously, either $u=v$ or $v=u$). The relation $\leftrightarrow_E^*$ is the smallest stable congruence that contains E; a congruence is, by definition. monotonic.

A *rewrite rule* is a pair of terms (l,r), written $l \rightarrow r$. A set R of rewrite rules generates a *rewriting relation* $\rightarrow_R$, which is the smallest stable, monotonic relation that contains R; i.e. $s \rightarrow_R t$ (s *rewrites* to t) iff $s \equiv w[l\sigma]$ and $t \equiv w[r\sigma/l\sigma]$ with $l \rightarrow r$ is in R. We say that the term $l\sigma$ is a *redex* and that it occurs at a *redex position* in the term $s\sigma$.

The symbols $\rightarrow_R^*$, $\rightarrow_R^+$ and $\leftarrow_R$ denote the reflexive-transitive, transitive, and inverse of the rewriting relation; $\leftrightarrow_R$ and $\leftrightarrow_R^*$ the symmetric closure and reflexive, symmetric, and transitive closure of the rewriting relation and its inverse, respectively. A term t is in *normal form* with respect to R, if there is no term u, such that $t \rightarrow_R u$. We denote the normal forms of the term t by $NF(t)$;

if the normal form of t is unique then we call it $\hat{t}$.

Let E be a set of equations and R be a set of rewrite rules. A *proof* of $s=t$ in $E \cup R$ is a sequence $(s_0, \ldots, s_n)$ such that s_0 is s and s_n is t, and for $1 \leq i \leq n$, one of $s_{i-1} \leftrightarrow_E s_i$, $s_{i-1} \rightarrow s_i$ or $s_{i-1} \leftarrow s_i$ holds.

A proof step $s \leftrightarrow_E t$ is called and *equality step*; proof steps $s \rightarrow_R t$ and $s \leftarrow_R t$ are called *rewrite steps*; and a proof of the form $s \leftarrow_R u \rightarrow_R t$ is called a *peak*. If we have a proof $s \leftrightarrow_E t_1 \leftrightarrow_E \cdots t$, where all the steps are equality steps we call this proof an *equality proof*. A proof that uses both $\leftrightarrow_E$ and $\leftrightarrow_R$ in its individual proof steps is called a *mixed equational and rewrite proof*, or just a *mixed proof*. If the proof only uses rewriting steps i.e. we have a proof of the form $s \leftrightarrow_R t_1 \leftrightarrow_R \cdots t$ then this is called a *rewrite* or "*zig-zag*" *proof*. A special type of rewriting proof is a proof of the form $s_0 \overset{*}{\rightarrow}_R s_k \overset{*}{\leftarrow}_R s_n$, this is called a *normal rewrite* or *normal proof*. Note: we usually abbreviate a proof of the form $s_i \rightarrow_R s_{i+1} \cdots \rightarrow_R s_j$ by $s_i \overset{*}{\rightarrow}_R s_j$. A *subproof* of a proof $(s_0, \ldots, s_n)$ is any proof $(s_i, \ldots, s_j)$ where $0 \leq i \leq j \leq n$. The notation $P[P']$ indicates that P' is a subproof of the proof P.

An *elimination pattern* is a pair of proof patterns, $P => P'$. Where a *proof pattern* in an arbitrary $E \cup R$ is a schema for a class of proofs that share a common structure. For example, $s \leftrightarrow_E t$ characterizes all one step equality proofs in E; $s \rightarrow_R t$ characterizes all single step rewrite proofs in R; $s \overset{*}{\rightarrow}_R u \overset{*}{\leftarrow}_R t$ characterizes all rewrite proofs in R; and $t \leftarrow_R u \rightarrow_R t$ all peaks. An *instance* of a pattern is any specific proof with the corresponding structure.

If S is a set of elimination patterns then $=>_S$ is the smallest stable, monotonic relation that contains S ($=>_S$ is a rewriting relation on proofs). The binary relation $=>_S$ on proofs is *monotonic* if $P =>_S P'$ implies $Q[P] =>_S Q[P'/P]$ for all proofs Q, P and P' . The relation is *stable* if

$$(s,...,u,...t) =>_S (s,...,v,...t) \text{ implies}$$
$$(w[s\sigma],...,w[u\sigma],...w[t\sigma]) =>_S (w[s\sigma],...,w[v\sigma],...w[t\sigma])$$

for all terms s, t, u, v, and w and all substitutions σ.

Following Bachmair, Dershowitz and Hsiang[2] we call the binary relation $=>_S$ on proofs a *proof ordering*, if $=>_S$ is contained in a well-founded ordering.

2.2. Termination Properties

Let R be a set of rewriting rules we say that a term rewriting system R is *terminating* (noetherian) if and only if there are no infinite sequences of rewritings $s_1 \rightarrow_R s_2 \rightarrow_R \cdots$. Proving that the rewriting relation $\rightarrow_R$ is terminating is usually done by embedding the reduction ordering $\rightarrow_R$ into an abstract ordering $>$ that is known to be *well-founded* (a well-founded ordering is a partial order in which every decreasing sequence has a minimum element; if the well-founded ordering is total we call it a *well-ordering*). We associate these two orderings by requiring that for all terms s and t, and a given function f we have

$$\forall st \; [s \rightarrow_R t \supset f(s) > f(t)].$$

We prove this by showing that for all substitutions σ on the rules $l_i \rightarrow r_t$ in the set R, we have $l_i\sigma > r_i\sigma$; and that the monotonicity property ($\forall st \; [s > t \supset w[s] > w[t]]$) holds for all term s, t and w. The first property implies that applying a reduction at a subterm of a term will reduce the order of that subterm, while the second property guarantees that the reduction in order of the subterm will decrease the order of of the original term. The well-foundedness of $>$ guarantees each reduction sequence will have a minimal element and hence the rewriting process will terminate[11].

Our interest in terminating (noetherian) rewriting relations stems from the fact that the well-foundedness of the reduction ordering allows us to perform transfinite induction over our term structure. We consider the following definition of *transfinite induction*†[15]. Let $>$ be a well-founded ordering on the set from which s and t are taken, and let P be a predicate over this set.

$$\text{(TI)} \qquad \forall s \; [\forall t \; [s > t \supset P(t)] \supset P(s)] \;\; \supset \;\; \forall s \; [P(s)]$$

If we apply this principle to term rewriting theory then it is usually called the *Principle of Noetherian Induction.* Let P be any predicate on our set of terms T, and let R be a terminating rewrite system, then for terms s and t we have

$$\text{(NI)} \qquad \forall s \; [\forall t \; [s \longrightarrow_R^+ t \supset P(t)] \supset P(s)] \;\; \supset \;\; \forall s \; [P(s)]$$

It is easy to see that since $\forall st \; [s \longrightarrow_R^+ t \supset s > t]$ the Principle of Noetherian Induction is an instance of Transfinite Induction, but has the advantage of not requiring the construction of a well-founded ordering, using instead the reduction ordering itself (which is embedded in the well-founded ordering $>$ to prove termination).

Well-founded Orderings

Well-founded orderings are the basic tool for proving termination in all manner of rewriting systems. We will provide below the basic definition of a fairly common scheme for well-ordering terms. Before this we must provide some background definitions concerning multisets.

A *multiset* M is a collection of objects from a base set S in which we may have repeated occurrences of a single object. If $>$ is a partial ordering on the set S, then the corresponding *multiset extension* $>>$ of $>$ is the smallest transitive relation such that

† Sometimes the principle of transfinite induction is restricted to well-orderings, and our more general statement is sometimes called *bar induction*[6].

$$M\cup\{x\} >> M\cup\{y_1, \ldots, y_n\},$$

whenever $n \geq 0$ and $x > y_i$, for $1 \leq i \leq n$.

We can calculate the ordering >> either from the definition above, or by the following method. Let $X=[x_1,x_2,..,x_m]$ and $Y=[y_1,y_2, \ldots, y_n]$ (where [] are the braces enclosing multisets). If we order the elements of the sets X and Y such that $x_{j_1} \geq x_{j_2} \cdots x_{j_m}$ and $y_{k_1} \geq y_{k_2} \cdots y_{k_n}$ and then compare lexicographically the two terms $x_{j_1}x_{j_2} \cdots x_{j_m}$ and $y_{k_1}y_{k_2} \cdots y_{k_n}$, then $X >> Y$ if and only if $x_{j_1}x_{j_2} \cdots x_{j_m} >_{lex} y_{k_1}y_{k_2} \cdots y_{k_n}$.

Higman[16] originally proved that the ordering >> on multisets is well-founded if and only if the ordering > is well-founded (Higman originally called this property the *finite basis* property), but it was Dershowitz and Manna[9] who independently introduced this theorem and applied it to problems in computer science e.g. proving termination of programs and systems of rewrite rules.

We now examine a notation of ordinal terms similar to that of ordinal terms up to $\phi\omega 0$ (in the Fefermann-Schutte system of ordinal notations)[31] and show how we can order these terms. Our ability to order terms from any set $T(F \cup C \cup V)$ will follow from their close connection this notation.

Definition 2.1

Ordinal Terms

1. If $c \in C$ then c is an ordinal term.
2. If α is an ordinal term, then so is (f, α), where f is an operator from the set F.
3. If $\alpha_1, \ldots, \alpha_n$ are ordinal terms, then so is $\alpha_1 \# \cdots \# \alpha_n$.

Definition of $\alpha=\beta$

1. If $\alpha,\beta\in C$ then $\alpha=\beta$, iff $\alpha=_C\beta$
2. $(g,\alpha)=(f,\beta)$ iff $f=g$ and $\alpha=\beta$.
3. $\alpha_1\#, \ldots, \#\alpha_j=\beta_1\#, \ldots, \#\beta_k$ iff $[\alpha_1, \ldots, \alpha_j]=[\beta_1, \ldots, \beta_k]$

Definition of $\alpha>\beta$

1. If $\alpha\notin C$ and $\beta\in C$ then $\alpha>\beta$,
   otherwise, if $\alpha,\beta\in C$ $\alpha>\beta$ iff $\alpha>_C\beta$.
2. $(g,\alpha)>(f,\beta)$ iff
   $g>f$ and $(g,\alpha)>\beta$, or
   $g=f$ and $\alpha>\beta$, or
   $g<f$ and $\alpha\geq(f,\beta)$.
3. $\alpha_1\#, \ldots, \alpha_j>\beta_1\#, \ldots, \#\beta_k$ iff $[\alpha_1, \ldots, \alpha_j]>>[\beta_1, \ldots, \beta_k]$

We associate the above ordering structure with terms from our set T in the following way. If we have a term $f(t_1, \ldots, t_n)\in T$ then the corresponding ordinal term is $(f, t_1\#, \ldots, \#t_n)$; of course, this implies that there exists a partial ordering on both the constant symbols and the operator symbols of T.

Dershowitz and Okada[12] have provided another well-founded ordering for terms based on Ackermann's system of ordinal notation[1]. This larger more general ordering structure is expected to reduce the frequency of failure in the completion process (cf. the next chapter), but there is much research to be done in actually implementing this ordering test. Depending on the parameters supplied Ackermann's ordering ranges from the lexicographic path ordering[21] on the one hand to the recursive path ordering[10] on the other. Getting the right mix between these two orderings so that we may order a set of equations that neither on their own may order may prove to be very computationally expensive (NP-hard), and we must develop techniques in order to restrict the number of potential orderings that we will have to try.

The termination of the rewriting relation implies that every term has a normal form; what we will provide in the next section are the conditions that guarantee that this normal form is unique.

2.3. Confluence Properties

What we will attempt to do in this section is to relate the notion of equivalence in equational theories with that of term rewriting theory. In addition we will provide the sufficient conditions required to effectively use the notion of equivalence in a term rewriting theory to decide equivalence in the equational system from which the rewriting system was derived.

Our notion of equivalence for a given term rewriting system R is that of *interconvertability* i.e. $s \leftrightarrow_R^* t$; and in the following theorem we will show this to be equivalent to the definition of equivalence in the original equational system E.

Theorem 2.1

$s \leftrightarrow_R^* t$ iff $s \leftrightarrow_E^* t$ where R is the rewriting system produced by orienting the equations in E.

Proof

For the basis case we have $s \leftrightarrow_R t$ iff $s \leftrightarrow_E t$ for all axioms in both theories: we proceed by induction on n the number of proofs steps in both the equational and rewrite proofs.

Definition 2.2

A rewrite system R is said to have the *Church-Rosser* property[5] if, for all terms s and t with $s \leftrightarrow_R^* t$, there exists a term u such that $s \rightarrow_R^* u \leftarrow_R^* t$ (note: we often say that the term u *joins* s and t, and denote this by $s \downarrow_R t$).

What we will now provide are the sufficient conditions required to show that a system has the Church-Rosser property, it is this property that allows us to determine the equivalence of terms just by rewriting i.e. we wish to show that the normal form of a term is unique and that it generates a class of terms equivalent to the term from which the normal from was derived.

Definition 2.3

A rewrite system R is *confluent* iff

$\forall stv\ [(s \rightarrow^{*}_{R} t \wedge s \rightarrow^{*}_{R} v) \supset t \downarrow_{R} v]$

$$\begin{tikzcd}
 & u \arrow[dl, "R^{*}"'] \arrow[dr, "R^{*}"] & \\
s \arrow[dr, "R^{*}"'] & & t \arrow[dl, "R^{*}"] \\
 & v &
\end{tikzcd}$$

Figure 2.1 - Confluence

We are going to use this notion of confluence in the following lemma. It reduces the test of term equivalence under interconvertability to that of the testing of the equality of normal forms.

Lemma 2.1

If R is a confluent rewriting system, then $\forall st\ [s \leftrightarrow^{*}_{R} t$ iff $s \downarrow_{R} t]$.

Proof

By induction on n, where n is the number of proof steps in $s \leftrightarrow^{n}_{R} t$.

Thus, to prove that the system R has the Church-Rosser property it is sufficient to prove that R is confluent. What the confluence of the system R also implies is that the reduction process is insensitive to the order in which reductions are applied. This is important since it opens up the opportunity for applying the reductions in a highly parallel fashion.

What we will do now is reduce the question of confluence to that of local confluence, a condition that requires that all single step ambiguities be joinable.

Definition 2.4

A rewrite system R is said to be *locally confluent* iff

$$\forall tvw\ [(t \longrightarrow_R v \wedge t \longrightarrow_R w) \supset v \downarrow_R w]$$

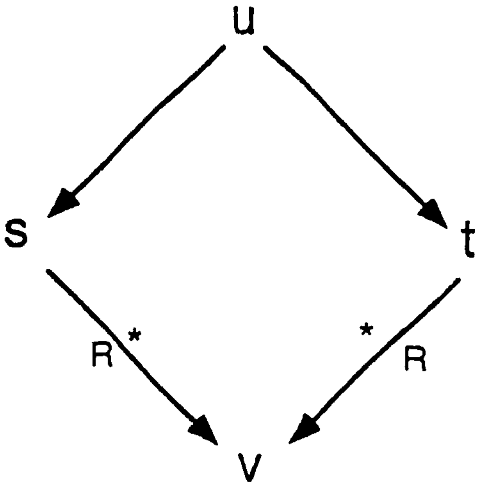

Figure 2.2 - Local Confluence

The following lemma is due originally to Newman[28], but is proved following Huet[19].

Lemma 2.2

If R is a noetherian term rewriting system, then R is confluent iff it is locally confluent.

Proof

The "only if" portion of the proof is trivial. For the "if" portion of the proof we want to show that if our rewrite system R is locally confluent then

$\forall tvw\ [(t \rightarrow_R^* v \wedge t \rightarrow_R^* w) \supset v \downarrow_R w]$ *i.e.* $\exists p\ [v \rightarrow_R^* p \wedge w \rightarrow_R^* p]$

We will prove this using noetherian induction.

Let t rewrite to v in m steps i.e. $t \rightarrow_R^m v$ and also let t rewrite to w in n steps i.e. $t \rightarrow_R^n w$;

(a) If $m=0$, we let $p=w$; if $n=0$, we let $p=v$.

(b) Otherwise, let $t \rightarrow_R v_1 \rightarrow_R^* v$ and $t \rightarrow_R w_1 \rightarrow_R w$; by local confluence, $\exists r\,[v_1 \rightarrow_R^* r \wedge w_1 \rightarrow_R^* r]$. Now by the IH (assuming confluence at v_1), we have we have $v \downarrow_R r$ joinable by p, and again by the IH (at w_1) we have $r \downarrow_R w$ joinable by q, now by the transitivity of reducibility we have $v \downarrow_R w$ joinable by q (since $v \rightarrow_R^* r$), which completes our proof.

For relations that are not noetherian we require more restrictive local hypotheses (such as *left-linearity*† to prove confluence within the whole system).

Local confluence, although a stronger condition than confluence, is still too general a condition because it requires the possibility of testing an infinite number of terms for joinability. What we will do in the following lemma (due originally to Knuth)[24] is reduce the confluence condition of joinability for all one step rewriting ambiguities to that of joinability for an "essential" set of ambiguities called *critical-pairs.* These critical-pairs determine the first possible positions at which an ambiguity may occur. Knuth's original proof required that the rewrite system R be noetherian, but the proof we give (again following Huet[19], lemma 3.1) will

† A term rewriting system is left-linear if the left-hand sides of all the rules do not contain more than one occurrence of a given variable.

not require this restriction. While not requiring the termination of the system in general Huet does, however, require termination of any rewriting on critical-pairs since his proof requires that they be joinable.

Definition 2.5

Two rules $s \longrightarrow t$ and $l \longrightarrow r$ *overlap* on a term u if there subterms of u which s and l match onto (i.e. we have $u[s\sigma]$ and $u[l\sigma]$). The overlap is *critical* if the left-hand side of one rule matchs onto a non-variable instance of the other. If the overlap is at a variable instance, or is disjoint, then the overlap is called *non-critical.*

Definition 2.6

Two rules $s \longrightarrow t$ and $l \longrightarrow r$ determine a *critical-pair* $<p,q>$ if for some minimal (most-general) substitution σ there exists a non-variable instance of the term $s\sigma$ such that we have a *critical overlap* $p \longleftarrow_R s\,\sigma[l\sigma] \longrightarrow_R q$ with:

$$p = t\sigma \text{ and}$$
$$q = s\sigma[r\sigma/l\sigma]$$

What the critical-pair $<t\sigma, s\sigma[r\sigma]>$ means is that the left-hand side of the rule $s \longrightarrow t$ may rewrite in two different ways i.e. we can have $s\sigma[l\sigma] \longrightarrow_R t\sigma$ and $s\sigma[l\sigma] \longrightarrow_R s\sigma[r\sigma/l\sigma]$. We call the term $s\sigma[l\sigma]$ the *superposition* of the rules $s \longrightarrow t$ and $l \longrightarrow r$. Note: we assume, without loss of generality that the variables of these two rules are distinct, under renaming.

$$\begin{tikzcd}
 & s\sigma[l\ \sigma] \arrow[dl] \arrow[dr] & \\
t\sigma & & s\sigma[r\ \sigma]
\end{tikzcd}$$

Figure 2.3 - A General Critical Pair Overlap

For example, consider the following two rules $h(f(a,x)) \longrightarrow g(x)$ and $f(x,b) \longrightarrow b$. There exists a critical overlap $h(f(a,b))$ between them, giving rise to a critical-pair $<g(b),h(b)>$.

$$\begin{tikzcd}
 & h(f(a,b)) \arrow[dl] \arrow[dr] & \\
g(b) & & h(b)
\end{tikzcd}$$

Figure 2.4 - An Example Critical Pair Overlap

Lemma 2.3

A term rewriting system R is locally confluent iff for every critical-pair $<i\sigma,s\sigma[r\sigma/l\sigma]> \in CP(R)$ we have $t\sigma \downarrow_R s\sigma[r\sigma/l\sigma]$; where $CP(R)$ is the set of all critical-pairs between rules in R.

Proof

For the "only-if" portion of the proof, any critical-pair $<t\sigma,s\sigma[r\sigma]>$ such that $s\sigma[l\sigma]\longrightarrow_R t\sigma$ and $s\sigma[l\sigma]\longrightarrow_R s\sigma[r\sigma/l\sigma]$ is joinable by our hypothesis of local confluence.

For the "if" part, we assume that for every critical-pair $<t\sigma,s\sigma[l\sigma]>$ we have $t\sigma\downarrow_R s\sigma[r\sigma/l\sigma]$. Now, for any arbitrary term $w\sigma[s\sigma][l\sigma]$ with $w\longrightarrow_R v_1$ and $w\longrightarrow_R v_2$, where $v_1=w\sigma[t\sigma/s\sigma]$ and $v_2=w\sigma[r\sigma/l\sigma]$; we want to show that we have $v_1\downarrow_R v_2$. We consider the following cases:

(a) $s\sigma$ and $l\sigma$ are disjoint instances. We let $v_1=w\sigma[t\sigma/s\sigma][l\sigma]$ and $v_2=w\sigma[s\sigma][r\sigma/l\sigma]$ we now have a new term $v' =w\sigma[t\sigma/s\sigma][r\sigma/l\sigma]$ which, by the commutativity of reducibility, joins the terms v_1 and v_2

(b) In this case we assume, without loss of generality, that we have the superposition $s\sigma[l\sigma]$ i.e. the rule l matches on to a subterm $l\sigma$ of $s\sigma$. We have, again, two possible cases:

(i) The term $l\sigma$ occurs at a variable position x in the term $s\sigma$. We consider a new substitution σ' such that $\sigma' =\sigma$ for all variables other than x; and we let $x\sigma' =r\sigma$. Also, let $v_1=w\sigma[t\sigma/s\sigma]$ and $v_2=w\sigma[s\sigma[r\sigma/l\sigma]]$ Since we have $w\sigma[s\sigma]\longrightarrow_R w\sigma[t\sigma/s\sigma]$ by stability we also have $w\sigma' [s\sigma']\longrightarrow_R w\sigma' [t\sigma' /s\sigma']$. But $w\sigma[s\sigma[l\sigma]]\longrightarrow^*_R w\sigma' [t\sigma']$ since $x\sigma\longrightarrow_R x\sigma'$. Hence the terms v_1 and v_2 are joined by $w\sigma' [t\sigma']$

(ii) The term $l\sigma$ occurs at a non-variable position in the term $s\sigma$. We have $v_1=w[t\sigma]$ and $v_2=w\sigma[s\sigma[r\sigma/l\sigma]]$, but by our hypothesis there exists a term u that joins $t\sigma$ $s\sigma[r\sigma]$, hence by monotonicity, we have a term $v' =w\sigma[u\sigma]$ such that v' joins v_1 and v_2.

Corollary

All the non-critical overlaps of R are joinable in R.

Proof

Consider cases a and b(i) above.

What we have provided above are the sufficient conditions required for a term rewriting system that guarantees that every term will have a unique normal form with respect to the rules of the system. This normal form generates a class of terms equivalent under the notion of equality for term rewriting systems.

Theorem 2.2

If a term rewriting system R is terminating and contains no unjoinable critical-pairs, then R is *convergent*, and hence we can decide equivalence in the theory generated by R solely by constructing a normal rewrite proof.

Proof

By lemma 2.3 the system R is locally confluent. Local confluence and lemma 2.2 imply that R is confluent. Finally lemma 2.1 implies that we can determine an equivalence $s \overset{*}{\longleftrightarrow}_R t$ by testing the equivalence of $\hat{s}$ and $\hat{t}$ (both are guaranteed to exist since R is terminating).

Corollary

If we have obtained R by orienting the equations from a set E, and R is convergent, then by theorem 2.1 we can decide equivalence in the theory generated by E by constructing an normal rewrite proof in R.

What we will do in the next section is show how we can guarantee the conditions for convergence (i.e. termination and confluence) in a given set of rewrite rules that we derive from a arbitrary set of equations (perhaps by adding additional rewrite rules) in such a way that the rewrite rules still have the same power with respect to provability as the original equational system. This rewrite system will solve the *word problem* (the equational equivalence of *ground* (i.e variable free) terms for an equational theory). As a (sad) final note however, the unsolvability of the word problem in general implies,'that we will not always be able to find a finite, convergent term rewriting system for an arbitrary equational theory.

We also may not be able to find a finite, convergent term rewriting system for certain decidable equational theories because of the nature of their axioms (e.g. the axioms have some essential associativity and commutativity). It is our belief that equational systems for which the completion procedure produces a finite. convergent rewriting system are the exception and not the rule. hence we need to provide techniques for theorem proving within these infinite systems, a topic which we will examine in a later chapter.

The Correctness of Basic Completion

3. Introduction

The process of producing a convergent term rewriting system from a system of equations such that the systems of equations and rewrite rules have the same power with respect to provability is called *completion.* This process proceeds by selecting equations to be oriented with respect to some well-founded ordering structure; and then, in some kind of algorithmic fashion, (possibly) generating new equations that will "repair" the power lost due to losing the symmetric property of the original equations.

In this section we will introduce the *Knuth-Bendix completion procedure*[24] (The version of the Knuth-Bendix procedure we will provide is that described by Gerard Huet in his 1981 paper)[20] that attempts to produce a convergent system R with the same power as the original set of equations E from which R was derived. We will first introduce Huet's version of the Knuth-Bendix procedure, and then formulate completion as an equational inference system, the objects of which are the pairs (E_i,R_i) of equations E_i and rewrite rules R_i.

3.1. The Standard Completion Procedure

The initial data for the following procedure is: a set of equations E and a well-founded ordering $>$. Let $E_0 := E$; $R_0 := \emptyset$; $i := 0$; $p := 0$.

```
loop
  while $E_i \neq \emptyset$ do
    select an equation $s=t \in E_i$
    if $\hat{s} = \hat{t}$ then
      $E_{i+1} := E_i - \{s=t\}$; $i := i+1$
    else if $s$ and $t$ are comparable then
      if $\hat{s} > \hat{t}$ then $l := \hat{s}$; $r := \hat{t}$ else $l := \hat{t}$; $r := \hat{s}$
      iet $K$ be the set of rules with label k whose left-hand sides
      are reducible by $l \rightarrow r$ to $l_k'$ and $l_k > l$ or $l >_s l_k$
      $E_{i+1} := E_i - \{s=t\} \cup \{l_k' = r_k \mid k \in K\}$
      $p := p+1$
      $R_{i+1} := \{j: l_j \rightarrow r_j' \mid j \notin K\} \cup \{p: l \rightarrow r'\}$
      where $r_j'$ is a normal form of $r_j$ with respect to rules from $R_i \cup \{l \rightarrow r\}$
      rule $p$ is unmarked.
      $i := i+1$
    else
      exit(failure - incomparable terms)
    endif
  endwhile
  Compute critical-pairs
    if all rules marked exit(convergent)
    select an unmarked rule with label $k$ from $R_i$
    let $E_{i+1}$ be the critical-pairs of $R_i$ between
    rule $k$ and all rules with labels less than or equal to $k$
    mark rule $k$
endloop
```

Figure 3.1 - Knuth-Bendix Completion Procedure

As noted previously, the unsolvability of the word problem, and the properties of certain equational systems imply that we will not always be able to find a finite convergent system of rewriting rules for an arbitrary equational theory. We have three possible cases to consider. We can terminate without failure and hence have a convergent system (the system has no critical-pairs and the reduction ordering is contained within the well-founded ordering $>$). We can terminate with failure (because the two sides of an equation are incomparable), little of interest may be noted in this case except that the combined system of equations and rewrite rules $E_i \cup R_i$ has the same power as the original system E. In the case of failure the only thing to be done is to restart the procedure with a new ordering that will order the two sides of the problem equation. It is expected that the larger, more general proof theoretic ordinals introduced by Dershowitz and Okada[12] will reduce the frequency of failure of the completion process by reducing the number of incomparable terms.

The final case we may encounter is that in which the completion process does not terminate and we compute successive approximations R_i of an infinite convergent system R_∞. (Note: we will wait until the following secticns to define R_∞ in detail, and prove that it is convergent).

Currently, there are no techniques for dealing with these infinite convergent systems but we will demonstrate in a later section that the framework of proof normalization provides the possibility, if not for a decision procedure, then a semi-decision procedure for equality within these infinite systems of rewriting rules.

3.2. Standard Completion as an Inference System

We can formulate completion as the proof system C (for *standard completion*) containing the inference rules listed below (let $>$ be a well-founded term ordering):

1) Orienting an equation.

$$\frac{(E\cup\{s \doteq t\},R)}{(E,R\cup\{s\longrightarrow t\})}$$

if $s>t$

2) Adding an equational consequence.

$$\frac{(E,R)}{(E\cup\{s=t\},R)}$$

if $<s,t> \in CP(R)$

3) Simplifying an equation.

$$\frac{(E\cup\{s\doteq t\},R)}{(E\cup\{u\doteq t\},R)}$$

if $s\longrightarrow_R u$

4) Deleting a trivial equation.

$$\frac{(E\cup\{s=s\},R)}{(E,R)}$$

We also have to provide the following rules for the simplification of rewrite rules (note: this is where Huet's completion procedure differs from that originally provided by Knuth and Bendix).

5) Simplifying the right-hand side of a rewrite rule.

$$\frac{(E,R\cup\{s\longrightarrow t\})}{(E,R\cup\{s\longrightarrow u\})}$$

if $t\longrightarrow_R u$ by $l\longrightarrow r\in R$

6) Simplifying the left-hand side of a rewrite rule.

$$\frac{(E,R\cup\{s\longrightarrow t\})}{(E\cup\{u=t\},R)}$$

if $s\longrightarrow_R u$ by $l\longrightarrow r\in R$ with $s\sigma>l\sigma$. or $s>_s l$

We write $(E,R)\vdash(E',R')$ if (E',R') can be obtained from (E,R) by an application of an inference rule from C. A *derivation* is a (possibly infinite) sequence $(E_0,R_0)\vdash(E_1,R_1)\cdots$. The *limit* of a derivation is the pair (E_∞,R_∞) where E_∞ is the set $\cup_{i\geq 0}\cap_{j\geq i}E_j$ of all *persisting equations* and R_∞ is the set $\cup_{i\geq 0}\cap_{j\geq i}R_j$ of all *persisting rules.*

The following lemma expresses the *soundness* of standard completion.

Lemma 3.1

If $(E,R) \vdash (E',R')$ then the congruence relations $\overset{*}{\leftrightarrow}_{E \cup R}$ and $\overset{*}{\leftrightarrow}_{E' \cup R'}$ are the same.

Proof

The proof is by cases considering the six inference rules of standard completion. For example, if we examine the third rule any proof that uses the equation $s=t$ in (E,R) will use after the application of the inference rule the compound step $s \rightarrow_R u \leftrightarrow_E t$ in (E',R'). Any proof that uses the equation $u=t$ in (E',R') will use the step $u \leftarrow_R s \leftrightarrow_E t$ in (E,R).

3.3. The Correctness of Standard Completion

Definition 3.1

A completion procedure C is correct if the limit system R_∞ generated by C is convergent; and $s \leftrightarrow_E t$ iff $s \leftrightarrow_{R_\infty} t$ where E is the set of equations from which R_∞ is derived.

We will prove the *correctness* of standard completion in a manner similar to that provided by Bachmair and Dershowitz[4]. Our proof will make use of the concept of *proof orderings* (defined earlier) originally provided by Bachmair, Dershowitz and Hsiang[2].

The correctness of the Knuth-Bendix procedure (in the case when R_∞, the limit rewrite system, is infinite) was proved first by Huet[20], using a fairly complicated argument using noetherian induction upon the reduction ordering. Basing the proof of correctness on proof orderings not only provides a simpler proof of correctness (for the infinite case), but also provides a framework in which the so called *critical-pair criteria* (which will be introduced in the following chapter) may be proved correct.

The essential idea here is that standard completion and the rewrite systems R_i provide a rewriting relation on mixed equational and rewrite proofs. Each step of the proof rewriting process transforms a mixed proof (by orienting equations, and removing peaks and non-persisting rewrite steps) to a normal form that is a persisting, normal rewrite proof. The correctness of this process follows from the well-foundedness of the reduction ordering on proofs (under some assumptions of fairness).

We give now, the set C of elimination patterns for standard completion:

1) *Equality patterns*

$s \leftrightarrow_E t \;\; => \;\; s \rightarrow_{R'} t$ if $s > t$

$s \leftrightarrow_E t \;\; => \;\; s \leftarrow_{R'} t$ if $t > s$

$s \leftrightarrow_E t \;\; => \;\; s \rightarrow_{R'} u \leftrightarrow_{E'} t$

$s \leftrightarrow_E t \;\; => \;\; s \leftrightarrow_{E'} u \leftarrow_{R'} t$

$s \leftrightarrow_E s \;\; => \;\; s$

2) *Overlap patterns*

non-critical overlaps:

$s \leftarrow_R u \rightarrow_R t \;\; => \;\; s \overset{*}{\rightarrow}_R v \overset{*}{\leftarrow}_R t$

critical overlaps (critical pair joinable in R):

$s \leftarrow_R u \rightarrow_R t \;\; => \;\; s \overset{*}{\rightarrow}_R v \overset{*}{\leftarrow}_R t$

critical overlaps (critical pair not joinable in R,
but existing as an equation in E'):

$s \leftarrow_R u \rightarrow_R t \;\; => \;\; s \leftrightarrow_{E'} t$

3) *Simplification patterns*

$$s \longrightarrow_R t \;\;=>\;\; s \longrightarrow_{R}' u \longleftarrow_{R}' t$$

$$s \longleftarrow_R t \;\;=>\;\; s \longrightarrow_{R}' u \longleftarrow_{R}' t$$

$$s \longrightarrow_R t \;\;=>\;\; s \longrightarrow_{R}' v \longleftrightarrow_{E}' t$$

$$s \longleftarrow_R t \;\;=>\;\; s \longleftrightarrow_{E}' v \longleftarrow_{R}' t$$

In the simplification patterns the step $s \longrightarrow_R t$ is by the the application of rule $l \longrightarrow r$ at some term $l\sigma$ in s; with $s \longrightarrow_R' u$ by application of the simplified rule $l \longrightarrow r'$ at the original $l\sigma$; also the step $s \longrightarrow_R' v$ is by the rule $l' \longrightarrow r'$ at a position $l'\ \sigma'$ with either $l\sigma > l'\ \sigma'$ (by stability) or we have $l\sigma = l'\ \sigma'$ with $l >_s l'$.

Lemma 3.2

Whenever $(E,R) \vdash (E'\ ,R'\ )$ and P is a proof in $E \cup R$, then there exists a proof P' in $E' \cup R'$, such that $P =>^{*}_{C} P'$.

Proof

The completion process corresponds to the application of the rewrite rules above (the rewrite rules reflect the changes to the sets E and R). Therefore, if we apply the rules to all steps in P that use the equation or rule that has changed in the completion step then we will have generated a new proof P' .

We next prove that the ordering $=>^{+}_{C}$ is well-founded. To do this we will embed $=>_C$ in a well-founded ordering on multisets whose elements are the complexity measure of each individual step in a proof.

Lemma 3.3

The ordering $=>^{+}_{C}$ is a proof ordering.

Proof

We will construct a well-founded ordering $>_C$; and prove that $P =>_C P'$ implies that $P >_C P'$, for all proofs P and P' .

We first define the *complexity* of a single proof step $c(s,t)$ as follows:

if $s \rightarrow_R t$ by $l \rightarrow r$ at sub-term $l\sigma$, then $c(s,t)$ is $(\{s\},l\sigma,l,t)$;

if $s \leftarrow_R t$ by $l \rightarrow r$ at sub-term $l\sigma$, then $c(s,t)$ is $(\{t\},l\sigma,l,s)$;

if $s \leftrightarrow_E t$ then $c(s,t)$ is $(\{s,t\},-,-,-)$.

Only the first component is relevant in the last case. The ordering $>_{C'}$ is the lexicographic combination of the multiset extension $>>$ of the abstract ordering $>$, the abstract ordering $>$, the proper subsumption ordering $>_s$, and the abstract ordering $>$†.

What remains to show is that the rewriting relation $=>_C$ on proofs is contained in the abstract ordering $>_C$. We will do this first by showing that $>_{C'}$ orders all the elimination patterns in C, and later that we can extend this to an ordering $>_C$ into which we can embed the reduction ordering $=>_C$. Consider the following equality patterns.

† In the original complexity measure of Bachmair and Dershowitz the second component was the proper subterm ordering, and the objects ordered were the redex positions. We have introduced this new ordering, in which the second component is the redex term ordered by the abstract ordering $>$, in order that it be consistent with the components we will add to the ordering we will introduce in the next chapter that will allow us to prove the correctness of our new critical-pair criteria.

1) *Equality patterns*

$(s \leftrightarrow_E t) >_C (s \rightarrow_R{}' t)$, since $[s,t] \gg [t]$;

$(s \leftrightarrow_E t) >_C (s \rightarrow_R{}' u \leftrightarrow_E{}' t,$

since $[s,t] \gg [s]$ and $[s,t] \gg [u,t]$;

$(s \leftrightarrow_E s) >_C (s)$, since $[s,s] \gg \emptyset$.

2) *Overlap patterns*

$(s \leftarrow_R u \rightarrow_R t) >_C (s \overset{*}{\rightarrow}_R v \overset{*}{\leftarrow}_R t)$,

since all terms on the right-hand side are smaller than u;

$(s \leftarrow_R u \rightarrow_R t) >_C (s \leftrightarrow_E{}' t)$, since $[u] \gg [s,t]$.

3) *Simplification patterns*

$[([s],l\sigma,l,t)] \gg_C [([s],l\sigma,l,u),([t],l' \sigma' ,l' ,u)]$,
since $t>u$ and $s>t$;

$[([s],l\sigma,l,t)] \gg_C [([t,u],-,-,-),([s],l\sigma,l' ,v)]$,

since $s>t$, $s>u$, and either $l\sigma>l' \sigma'$ or $l>_s l'$.

Let P be the proof $(s_0, \ldots, s_n)$ and by $M(P)$ let us denote the multiset $[c(s_0,s_1), \ldots, c(s_{n-1},s_n)]$. We now define the ordering $>_C$ in the following fashion $P >_C P'$) if and only if $M(P) \gg_{C'} M(P')$, where $\gg_C$ is the multiset extension of the ordering $>_C$. Since the abstract ordering $>$ is well-founded, monotonic, and stable; and the proper subsumption ordering $>_s$ is well-founded, we can conclude that $>_C$ is a proof ordering.

We will now examine the function of each of the components in the complexity measure. This is important because we will prove the correctness of our new critical-pair criterion by extending the ordering $>_C$ with terms that have functions analogous to those in the original ordering.

The first component of the ordering (ordered by the multiset extension $>>$ of the abstract ordering $>$) handles all the cases of equality and overlap patterns. The remaining three components handle the simplification patterns.

In the first case of the simplification patterns we have to show that the term $([s],l\sigma,l,t)$ is greater than both $([s],l\sigma,l,u)$ and $([t],l' \sigma' ,l' ,u)$ since $>>_C$ is a multiset ordering. $([s],l\sigma,l,t)$ is greater that $([t],l' \sigma' ,l' ,u)$ because s is greater than t (because we are using a lexicographic ordering the first component has precedence over all others). The fourth component is required to order the terms $([s],l\sigma,l,t)$ and $([s],l\sigma,l,u)$ since the first component, and the following two components are the same (these are the components required to order the other simplification patterns described below).

We will now consider the function of the second and third components in showing that the tuple $([s],l\sigma,l,t)$ is greater than both $([t.u],-,-,-)$ and $([s],l' \sigma' ,l' v)$. In the first case this is easy because both t and u are smaller that s. For the second case we consider first the case when s is reduced by a rule $l' \rightarrow r'$. If the reduction occurs at a smaller term than $l\sigma$ the second component is smaller, and if the reduction occurs at the term $l\sigma$ then we consider the subsumption ordering between l and l' to order the tuples using the third component.

What the above lemma will show is that the inference system C can be used to *normalize* arbitrary equational proofs by orienting equality steps $s \leftrightarrow_E t$, removing non-persistent rewrite rules and removing peaks $s \leftarrow_R u \rightarrow_R t$ by either showing that s and t can be joined, or introducing a new equation that will join them.

The peaks in an equational proof correspond to the one step rewriting ambiguities introduced in the previous section. As we saw in that section, lemma 2.3

implied that all one step rewriting ambiguities may be shown joinable by showing that a restricted set of these ambiguities called critical-pairs are joinable. In general, it is to these critical-pairs that we will restrict our attention when considering the proof normalization process in its dual role as framework for theorem proving and as a proof technique.

Since critical-pairs that are identified in the completion process are added to equational system that we are attempting to complete, before we prove the correctness of completion we need the following notion of *fairness*. This property of fairness will guarantees that all critical pairs will eventually be considere or completion.

Definition 3.2

A derivation $(E_0,R_0) \vdash (E_1,R_1) \vdash \cdots$ is *fair* if a) $E_\infty = \emptyset$; and b) the critical-pairs of R_i are a subset of $\cup_j E_j$, $j \geq 0$. A completion procedure is *fair* if it gives rise to a fair derivation.

This leads us immediately to the following theorem.

Theorem 3.1

If the completion procedure C is fair, and does not fail for inputs E, R, and supplied ordering $>$ then any arbitrary proof P in $E_i \cup R_i$, $i \geq 0$ may be transformed into a normal rewrite proof Q in R_∞, by the relation $=>_C$, i.e. we have $P =>_C^* Q$.

Proof

The proof will be by transfinite induction on the well-founded ordering $>_C$ i.e. for any proof P in $E_i \cup R_i$, $i \geq 0$ we will show that if we assume for all P' in $E_j \cup R_j$, $j \geq 0$ such that $P >_C P'$, and $P' =>_C^* Q'$ where Q' is a normal rewrite proof in R_∞, then $P =>_C^* Q$, where Q is a normal rewrite proof in

R_∞.

We let P be a proof in $E_i \cup R_i$, and we will consider the four following cases.

1. P contains a equality step using a non-persisting equation $s=t$ i.e. $P \equiv P_1 \leftrightarrow_{E_i} P_2$ in $E_i \cup R_i$. In the example where the $s=t$ has been transformed into the rule $s \rightarrow t$ in R_j. We have a proof $P' \equiv P_1' \rightarrow_{R_j} P_2'$ in $E_j \cup R_j$, $j > i$ with $P_1 =>^{*}_{C} P_1'$, and $P_2 =>^{*}_{C} P_2'$, (by repeated application of lemma 3.2). Since at least one step in P' is smaller than the corresponding step in P ($P >_C P'$, because $>_C$ is based upon a multiset ordering), by our induction hypothesis P' can be transformed in a normal rewrite proof Q' in R_∞, and we have $P =>^{+}_{C} P' =>^{*}_{C} Q'$. We can make an exactly similar argument in the case when the equality step has been simplified, and/or deleted. The second assumption in the fairness condition guarantees that there will be no persisting equations, and hence no persisting equational steps.

2. P contains a rewrite step using a non-persisting rule i.e. $P \equiv P_1 \rightarrow_{R_i} P_2$ in $E_i \cup R_i$, We have a proof $P' \equiv P_1' P_3' P_2'$ in $E_j \cup R_j$, $j > i$ with $P_1 =>^{*}_{C} P_1'$, $P_2 =>^{*}_{C} P_2'$, and $\rightarrow_{R_i} =>^{*}_{C} P_3'$ (by repeated application of lemma 3.2), i.e. the rewrite step $\rightarrow_{R_i}$ is removed by $=>_C$ because the rule generating it has been simplified. Since at least one step in P' is smaller than the corresponding step in P, by our induction hypothesis P' can be transformed in a normal rewrite proof Q' in R_∞, and we have $P =>^{+}_{C} P' =>^{*}_{C} Q'$. Similarly for the proof $P \equiv P_1 \leftarrow_{R_i} P_2$ in $E_i \cup R_i$.

3. P contains a peak generated from persisting rewrite rules i.e. we have $P \equiv P_1 \leftarrow_{R_i} u \rightarrow_{R_i} P_2$ in $E_i \cup R_i$, we have three possible cases for our

proofs P in $E_i \cup R_i$ and P' in $E_j \cup R_j$.

3.1 P contains a non-critical overlap, hence we will have $P \Rightarrow^{+}_{C} P'$ where $P' \equiv P_1' \rightarrow^{*}_{R_j} v \leftarrow^{*}_{R_j} P_2'$ in $E_j \cup R_j$, $j \geq i$, with $P_1 \Rightarrow^{*}_{C} P_1'$, and $P_2 \Rightarrow^{*}_{C} P_2'$, (by repeated application of lemma 3.2), at least one step in P' is smaller than in P and by the induction hypothesis P' can be transformed into a rewrite proof Q' in R_∞.

3.2 P contains a critical overlap, but since it is generated by an overlap between persisting rules, by the second assumption of the fairness condition the critical pair $<s', t'>$ that generates the critical overlap exists as an equation $s' = t'$ in some E_j. If $j > i$ we will have $P \Rightarrow^{+}_{C} P'$ where $P' \equiv P_1' \leftrightarrow_{E_j} P_2'$ with $P_1 \Rightarrow^{*}_{C} P_1'$, and $P_2 \Rightarrow^{*}_{C} P_2'$, (by repeated application of lemma 3.2), at least one step in P' is smaller than in P and hence P' can be transformed into a rewrite proof Q' in R_∞. If $j \leq i$ then will have a proof P_3 linking s' and t' that will connect the critical overlap in $E_i \cup R_i$. And we have $P' \equiv P_1 P_3 P_2$, P' is smaller than P, because P_3 connects s and t below the peak u, and so P' will eventually be transformed into rewrite proof Q' in R_∞.

In all of the above cases we have shown that the proof P can be transformed into some proof P' such that $P' \Rightarrow^{*}_{C} Q'$ where Q' is a proof in R_∞.

4. Finally, if P is a persisting normal rewrite proof in R_i then P will exists as a normal rewrite proof in R_∞.

To recap. we have shown that an arbitrary proof P in $E_i \cup R_i$ will either exist as, or can be transformed into a normal rewrite proof Q in R_∞.

Corollary

If C is fair then C is a correct completion procedure.

Proof

R_∞ is convergent (by theorem 2.2) because it is terminating ($\rightarrow_{R_\infty}$ is contained in the well-founded abstract ordering $>$), and R_∞ contains no unjoinable critical pairs (since any critical overlap will have a persisting normal rewrite proof). And, finally, by lemma 3.1 R_∞ has the same power with respect to provability as E.

The above proof not only provides a simpler, more flexible proof of correctness for standard completion (in the more general case when R_∞ is not finite) than that provided by Huet[20]; but if instead of considering proof normalization as a technique for demonstrating the correctness of completion by showing that an arbitrary proof may be transformed into a persisting normal rewrite proof; and. instead. consider its application to a given proof. We have the potential for a very powerful theorem proving technique.

Since we are dealing with a specific proof we may use the information regarding the current step which is being considered in order to select equations from the set E_i. Also. and perhaps more importantly, the specific proof step under consideration will allow us to provide a substitution for an unorientable equation that will allow us to turn it into a rule (ground terms can always be oriented without failure) that can be successfully applied. We will consider proof normalization as a theorem proving technique more completely in a later chapter.

3.4. An Example of Basic Completion

We will consider the following example of producing a convergent system of rewrite rules for the standard set of three axioms for groups. We will group together some of the actions that would be performed individually in the completion process in order to keep our example to a reasonable length.

Consider the following set of equations E.

	E	R
1.	$0+x=x$	$\varnothing$
2.	$i(x)+x=0$	
3.	$(x+y)+z=x+(y+z)$	

We orient all the equations.

E		R
$\varnothing$	1.	$0+x\rightarrow x$
	2.	$i(x)+x\rightarrow 0$
	3.	$(x+y)+z\rightarrow x+(y+z)$

There exists a critical overlap $(i(x)+x)+z$ between rules 3 and 2 giving a critical-pair $<i(x)+(x+z),0+z>$. We add the equation $i(x)+(x+y)=y$ to the system.

	E		R
1.	$i(x)+(x+y)=y$	1.	$0+x\rightarrow x$
		2.	$i(x)+x\rightarrow 0$
		3.	$(x+y)+z\rightarrow x+(y+z)$

We orient this first equation, giving us a new rule 4.

E		R
$\varnothing$	1.	$0+x\rightarrow x$
	2.	$i(x)+x\rightarrow 0$
	3.	$(x+y)+z\rightarrow x+(y+z)$
	4.	$i(x)+(x+y)\rightarrow y$

There exist the following overlaps; $i(0)+(0+x)$ between rule 4 and rule 1, and $i(i(x))+(i(x)+x)$ between rule 4 and rule 2. These overlaps give rise to the critical-pairs $<x,i(0)+x)>$ and $<x,i(i(x)+0>$ respectively. We add these to the set E.

	E		R
1.	$x=i(0)+x$	1.	$0+x\rightarrow x$
2.	$x=i(i(x))$	2.	$i(x)+x\rightarrow 0$
		3.	$(x+y)+z\rightarrow x+(y+z)$
		4.	$i(x)+(x+y)\rightarrow y$

We orient the two equations in E.

	E		R
	$\varnothing$	1.	$0+x\rightarrow x$
		2.	$i(x)+x\rightarrow 0$
		3.	$(x+y)+z\rightarrow x+(y+z)$
		4.	$i(x)+(x+y)\rightarrow y$
		5.	$i(0)+x\rightarrow x$
		6.	$i(i(x))\rightarrow x$

There is an overlap $i(i(0))+i(0)+x$ between rule 5 and rule rule 4, which gives rise to the critical-pair $<x,i(i(0))>$. We add this to the set E as an equational consequence.

	E		R
1.	$x=i(i(0))+x$	1.	$0+x\rightarrow x$
		2.	$i(x)+x\rightarrow 0$
		3.	$(x+y)+z\rightarrow x+(y+z)$
		4.	$i(x)+(x+y)\rightarrow y$
		5.	$i(0)+x\rightarrow x$
		6.	$i(i(x))\rightarrow x$

We orient the first equation.

	E		R
	$\varnothing$	1.	$0+x\rightarrow x$
		2.	$i(x)+x\rightarrow 0$
		3.	$(x+y)+z\rightarrow x+(y+z)$
		4.	$i(x)+(x+y)\rightarrow y$
		5.	$i(0)+x\rightarrow x$
		6.	$i(i(x))\rightarrow x$
		7.	$i(i(0))+x\rightarrow x$

There is an overlap $i(i(0))+i(0)$ between rule 7 and rule 2, which gives us the critical-pair $<i(0),0>$. We add this to the set of equations.

	E		R
1.	$i(0)=0$	1.	$0+x \to x$
		2.	$i(x)+x \to 0$
		3.	$(x+y)+z \to x+(y+z)$
		4.	$i(x)+(x+y) \to y$
		5.	$i(0)+x \to x$
		6.	$i(i(x)) \to x$
		7.	$i(i(0))+x \to x$

We orient this first equation.

E		R
$\varnothing$	1.	$0+x \to x$
	2.	$i(x)+x \to 0$
	3.	$(x+y)+z \to x+(y+z)$
	4.	$i(x)+(x+y) \to y$
	5.	$i(0)+x \to x$
	6.	$i(i(x)) \to x$
	7.	$i(i(0))+x \to x$
	8.	$i(0) \to 0$

E		R
$\varnothing$	1.	$0+x \to x$
	2.	$i(x)+x \to 0$
	3.	$(x+y)+z \to x+(y+z)$
	4.	$i(x)+(x+y) \to y$
	5.	$i(0)+x \to x$
	6.	$i(i(x)) \to x$
	7.	$i(i(0))+x \to x$
	8.	$i(0) \to 0$

Rule 8 simplifies the left-hand sides of both rule 5 and rule 7. These rules are removed from R and two new equations are added to E.

	E		R
1.	$0+x=x$	1.	$0+x \to x$
2.	$i(0)+x=x$	2.	$i(x)+x \to 0$
		3.	$(x+y)+z \to x+(y+z)$
		4.	$i(x)+(x+y) \to y$
		6.	$i(i(x)) \to x$
		8.	$i(0) \to 0$

However, both equations can be reduced to the form $s=s$; equation 1 by rule 1, and equation 2 by rule 8 and rule 1, hence they (the equations) may be removed.

E		R
$\emptyset$	1.	$0+x \rightarrow x$
	2.	$i(x)+x \rightarrow 0$
	3.	$(x+y)+z \rightarrow x+(y+z)$
	4.	$i(x)+(x+y) \rightarrow y$
	6.	$i(i(x)) \rightarrow x$
	8.	$i(0) \rightarrow 0$

There is a critical overlap $i(i(i(x)))+(i(i(x))+0)$ between rule 6 and rule 4. This generates the critical-pair $<0,i(i(i(x)))+x>$ which we can orient and add to set of rules as as a rule (skipping equational stage).

E		R
$\emptyset$	1.	$0+x \rightarrow x$
	2.	$i(x)+x \rightarrow 0$
	3.	$(x+y)+z \rightarrow x+(y+z)$
	4.	$i(x)+(x+y) \rightarrow y$
	6.	$i(i(x)) \rightarrow x$
	8.	$i(0) \rightarrow 0$
	9.	$i(i(i(x)))+x \rightarrow 0$

Rule number 4 overlaps on itself giving the overlap $i(i(x))+(i(x)+(x+y))$. this generates the critical-pair $<x+y,i(i(x))+y>$ (again, we bypass the equational step and orient this equational consequence).

E		R
$\emptyset$	1.	$0+x \rightarrow x$
	2.	$i(x)+x \rightarrow 0$
	3.	$(x+y)+z \rightarrow x+(y+z)$
	4.	$i(x)+(x+y) \rightarrow y$
	6.	$i(i(x)) \rightarrow x$
	8.	$i(0) \rightarrow 0$
	9.	$i(i(i(x)))+x \rightarrow 0$
	10.	$i(i(x)+y \rightarrow x+y$

Rule 10 reduces the left-hand side of rule 6 so we add $x+0=x$ to the set E. Rule 9 can be reduced to the form $s=s$ by rule 10 and rule 2 on the left-hand side, hence this rule is removed and no equation is added.

	E		R
1.	$x+0=x$	1.	$0+x \longrightarrow x$
		2.	$i(x)+x \longrightarrow 0$
		3.	$(x+y)+z \longrightarrow x+(y+z)$
		4.	$i(x)+(x+y) \longrightarrow y$
		6.	$i(i(x)) \longrightarrow x$
		8.	$i(0) \longrightarrow 0$
		10.	$i(i(x)+y \longrightarrow x+y$

We orient the equation.

E		R
$\varnothing$	1.	$0+x \longrightarrow x$
	2.	$i(x)+x \longrightarrow 0$
	3.	$(x+y)+z \longrightarrow x+(y+z)$
	4.	$i(x)+(x+y) \longrightarrow y$
	6.	$i(i(x)) \longrightarrow x$
	8.	$i(0) \longrightarrow 0$
	10.	$i(i(x)+y \longrightarrow x+y$
	11.	$x+0 \longrightarrow x$

Rule 10 and rule 11 generate the critical overlap $i(i(x))+0$ which gives rise to the critical-pair $<i(i(x)),x+0>$. reducing and orienting the equation, we add the rule $i(i(x) \longrightarrow x$ to the system.

E		R
$\varnothing$	1.	$0+x \longrightarrow x$
	2.	$i(x)+x \longrightarrow 0$
	3.	$(x+y)+z \longrightarrow x+(y+z)$
	4.	$i(x)+(x+y) \longrightarrow y$
	6.	$i(i(x)) \longrightarrow x$
	8.	$i(0) \longrightarrow 0$
	10.	$i(i(x)+y \longrightarrow x+y$
	11.	$x+0 \longrightarrow x$
	12.	$i(i(x) \longrightarrow x$

Rule 12 reduces the left-hand side of rule 10 to produce the trivial equality, and we remove it from the system. We have the following critical-pairs; $<0,x+i(x)>$ between rule 12 and rule 2, $<y,x+(i(x)+y)>$ between rule 12 and rule 4. These are not reducible so we orient them and add them to the system.

E		R
$\emptyset$	1.	$0+x\rightarrow x$
	2.	$i(x)+x\rightarrow 0$
	3.	$(x+y)+z\rightarrow x+(y+z)$
	4.	$i(x)+(x+y)\rightarrow y$
	6.	$i(i(x))\rightarrow x$
	8.	$i(0)\rightarrow 0$
	10.	$i(i(x)+y\rightarrow x+y$
	11.	$x+0\rightarrow x$
	12.	$i(i(x)\rightarrow x$
	13.	$x+i(x)\rightarrow 0$
	14.	$x+(i(x)+y)\rightarrow y$

Rule 13 and rule 3 overlap to form the critical-pair $<x+(y+i(x+y)),0>$, we orient it and add it to the system.

E		R
$\emptyset$	1.	$0+x\rightarrow x$
	2.	$i(x)+x\rightarrow 0$
	3.	$(x+y)+z\rightarrow x+(y+z)$
	4.	$i(x)+(x+y)\rightarrow y$
	6.	$i(i(x))\rightarrow x$
	8.	$i(0)\rightarrow 0$
	10.	$i(i(x)+y\rightarrow x+y$
	11.	$x+0\rightarrow x$
	12.	$i(i(x)\rightarrow x$
	13.	$x+i(x)\rightarrow 0$
	14.	$x+(i(x)+y)\rightarrow y$
	15.	$x+(y+i(x+y))\rightarrow 0$

Rule 15 overlaps with rule 4 to give the rule $x+i(y+x)\rightarrow i(y)$.

E		R
$\emptyset$	1.	$0+x\rightarrow x$
	2.	$i(x)+x\rightarrow 0$
	3.	$(x+y)+z\rightarrow x+(y+z)$
	4.	$i(x)+(x+y)\rightarrow y$
	6.	$i(i(x))\rightarrow x$
	8.	$i(0)\rightarrow 0$
	10.	$i(i(x)+y\rightarrow x+y$
	11.	$x+0\rightarrow x$
	12.	$i(i(x)\rightarrow x$
	13.	$x+i(x)\rightarrow 0$
	14.	$x+(i(x)+y)\rightarrow y$
	15.	$x+(y+i(x+y))\rightarrow 0$
	16.	$x+i(y+x)\rightarrow i(y)$

Rule 16 and rule 13 reduce and delete rule 15. Rule 16 overlaps with rule 4 to give the rule $i(x+y)\longrightarrow i(y)+i(x)$, and we add this to the system.

E		R
$\varnothing$	1.	$0+x\longrightarrow x$
	2.	$i(x)+x\longrightarrow 0$
	3.	$(x+y)+z\longrightarrow x+(y+z)$
	4.	$i(x)+(x+y)\longrightarrow y$
	6.	$i(i(x))\longrightarrow x$
	8.	$i(0)\longrightarrow 0$
	10.	$i(i(x)+y\longrightarrow x+y$
	11.	$x+0\longrightarrow x$
	12.	$i(i(x)\longrightarrow x$
	13.	$x+i(x)\longrightarrow 0$
	14.	$x+(i(x)+y)\longrightarrow y$
	16.	$x+i(y+x)\longrightarrow i(y)$
	17.	$i(x+y)\longrightarrow i(y)+i(x)$

Rule 17 and rule 14 reduce and delete rule 16. We have no more critical-pairs to orient so we have finally produced a convergent system.

E		R
$\varnothing$	1.	$0+x\longrightarrow x$
	2.	$i(x)+x\longrightarrow 0$
	3.	$(x+y)+z\longrightarrow x+(y+z)$
	4.	$i(x)+(x+y)\longrightarrow y$
	6.	$i(i(x))\longrightarrow x$
	8.	$i(0)\longrightarrow 0$
	10.	$i(i(x)+y\longrightarrow x+y$
	11.	$x+0\longrightarrow x$
	12.	$i(i(x)\longrightarrow x$
	13.	$x+i(x)\longrightarrow 0$
	14.	$x+(i(x)+y)\longrightarrow y$
	17.	$i(x+y)\longrightarrow i(y)+i(x)$

In the next chapter we will take a look at how we may introduce tests that will remedy some of the mistakes made in the order in which we selected equations from the set E.

A New More General Critical-Pair Criterion

4. Introduction

The efficiency of the completion procedure depends critically on the number of rewrite rules and critical-pairs generated, because extra rules have the opportunity to generate more critical-pairs, and also carry an overhead in any reduction mechanism. The fundamental way in which we reduce this number is by normalizing the sides of an equation before orienting it to generate a rule. If we can reduce the equation to its trivial form i.e. $s=s$, we can consider it to be superfluous and hence we do not need to add its oriented form to the set of rewrite rules.

Since we can only reduce a given equation with respect to rewrite rules that have already been computed, the sequence in which we select equations for conversion to rewriting rules is critical for the efficiency of the whole process. If an equation is selected "too early" i.e. the left and right-hand sides are potentially reducible by rules produced at a later stage in the completion process, we will have to simplify the rule after we have produced the reducing equation.

Having to reduce the right-hand side of a rule is not as serious as reducing the left-hand side since, if we reduce the left-hand side we have no guarantee that it is larger (with respect to our ordering $>$) than the right-hand side, and hence have to delete the rule and add the reduced form to the set of equations. Reducing the right hand-side maintains the decreasing property of the rule and hence we can keep it in the rule set (although we may have to do some work in order to update the redex replacement mechanism).

What we will provide in this chapter are tests which determine the redundancy of an equation $s=t$ by examining the structure of the critical overlap $s \leftarrow_R u \rightarrow_R t$ that produced the original equation. We will characterise these criteria for determining redundancy, the so called critical-pair criteria, and the correctness of these criteria will follow from the ability to provide a well-founded ordering that is consistent with the ordering provided to prove the correctness of the completion procedure in the previous chapter.

4.1. Basic Definitions Concerning CPC's

We will consider the definition and proof of correctness of three major critical-pair criteria in this section. We will then provide our extension and generalisation of the composite critical-pair criterion.

Definition 4.1

We say that a set *CPC* of elimination patterns of the form

$$s \leftarrow_R u \rightarrow_R t \; => \; s \overset{*}{\leftrightarrow}_{E \cup R} t$$

specifies a *critical-pair criterion,* where the reduction ordering is contained in the well-founded ordering $>$.

Although the above definition defines a rewriting relation that applies at general peaks, lemma 2.3 implies that we can restrict our attention to critical overlaps. Thus, in general, we will define our critical-pair criteria in terms of these critical overlaps.

By $=>_{CPC}$ we denote the rewrite relation on proofs generated by the set of elimination patterns CPC. $CPC(E,R)$ is the set of all critical-pairs $<s,t>$ in $CP(R)$ such that the critical overlap $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma]$ can be reduced using $=>_{CPC}$; i.e. there exists a proof P such that $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma] =>_{CPC} P$. We consider the pairs in $CPC(E,R)$ to be superfluous.

Definition 4.2

A derivation $(E_0,R_0)\vdash(E_1,R_1)\vdash\cdots$ is *fair relative* to a critical-pair criterion CPC if a) $E_\infty=\emptyset$, and b) $CP(R_\infty)-\cup_i CPC(E_i,R_i)$ is a subset of $\cup_j E_j$, $j\geq 0$.

Note: if we consider the trivial criterion CPC, i.e. the set $CPC(E,R)$ is always empty, then this definition reduces the the definition of fairness supplied in the previous chapter.

Definition 4.3

A critical-pair criterion CPC is *correct*, if for all unfailing derivations that are fair relative to CPC, the limit rewrite system R_∞ is convergent, and $s\leftrightarrow_E t$ iff $s\leftrightarrow_{R_\infty} t$.

This means that if we have a correct criterion CPC. all the pairs in $CPC(E.R)$ can be ignored in the completion process.

The following lemma will establish the correctness of a CPC.

Lemma 4.1

If the ordering $=>_C \cup =>_{CPC}$ can be embedded in a well-founded ordering $>_{CPC}$ and the correct completion procedure C is unfailing then any arbitrary proof P in $E_i\cup R_i$, $i\geq 0$ may transformed into an normal rewrite proof in R_∞, by the relation $=>_C \cup =>_{CPC}$. i.e. we have $P(=>_C \cup =>_{CPC})^* Q$.

Proof

The proof will be by transfinite induction on the well-founded ordering $>_{CPC}$ i.e. for any proof P in $E_i\cup R_i$, $i\geq 0$ we will show that if we assume for all P' in $E_j\cup R_j$, $j\geq 0$ such that $P>_{CPC}P'$. and P' $(=>_C\cup=>_{CPC})^* Q'$ where Q' is a normal rewrite proof in R_∞, then $P(=>_C\cup=>_{CPC})^* Q$.

where Q is a normal rewrite proof in R_∞.

We let P be a proof in $E_i \cup R_i$, and we will consider the four following cases (roughly the same four cases as considered in the proof of theorem 3.1).

1. P contains a equality step using a non-persisting equation $s=t$ i.e. $P \equiv P_1 \leftrightarrow_{E_i} P_2$ in $E_i \cup R_i$. In the example where the $s=t$ has been transformed into the rule $s \rightarrow t$ in R_j. We have a proof $P' \equiv P_1' \rightarrow_{R_j} P_2'$ in $E_j \cup R_j$, $j>i$ with $P_1(=>_C \cup =>_{CPC})^* P_1'$, and $P_2(=>_C \cup =>_{CPC})^* P_2'$, (by repeated application of lemma 3.2). Since at least one step in P' is smaller than the corresponding step in P ($P >_{CPC} P'$, because $>_{CPC}$ is based upon a multiset ordering), by our induction hypothesis P' can be transformed in a normal rewrite proof Q' in R_∞, and we have $P(=>_C \cup =>_{CPC})^+ P' (=>_C \cup =>_{CPC})^* Q'$. We can make an exactly similar argument in the case when the equality step has been simplified, and/or deleted. The first assumption in the fairness condition guarantees that there will be no persisting equations, and hence no persisting equational steps.

2. P contains a rewrite step using a non-persisting rule i.e. $P \equiv P_1 \rightarrow_{R_i} P_2$ in $E_i \cup R_i$, We have a proof $P' \equiv P_1' P_3' P_2'$ in $E_j \cup R_j$, $j>i$ with $P_1(=>_C \cup =>_{CPC})^* P_1'$, $P_2(=>_C \cup =>_{CPC})^* P_2'$, and $\rightarrow_{R_i}(=>_C \cup =>^{+}_{CPC}) P_3'$ (by repeated application of lemma 3.2), i.e. the rewrite step $\rightarrow_{R_i}$ is removed by $=>_C \cup =>_{CPC}$ because the rule generating it has been simplified. Since at least one step in P' is smaller than the corresponding step in P, by our induction hypothesis P' can be transformed into a normal rewrite proof Q' in R_∞, and we have $P(=>_C \cup =>_{CPC})^+ P' (=>_C \cup =>_{CPC})^* Q'$. Similarly for the proof $P \equiv P_1 \leftarrow_{R_i} P_2$ in $E_i \cup R_i$.

3. P contains a peak generated from persisting rewrite rules i.e. we have $P \equiv P_1 \leftarrow_{R_i} u \rightarrow_{R_i} P_2$ in $E_i \cup R_i$. We have three possible cases for our proofs P in $E_i \cup R_i$ and P' in $E_j \cup R_j$.

3.1 P contains a non-critical overlap, hence we will have $P(=>_C \cup =>_{CPC})^+ P'$ where $P' \equiv P_1' \overset{*}{\rightarrow}_{R_j} v \overset{*}{\leftarrow}_{R_j} P_2'$ in $E_j \cup R_j$, $j \geq i$, with $P_1(=>_C \cup =>_{CPC})^* P_1'$, and $P_2(=>_C \cup =>_{CPC})^* P_2'$. (by repeated application of lemma 3.2), at least one step in P' is smaller than in P and by the induction hypothesis P' can be transformed into a rewrite proof Q' in R_∞.

3.2 In the case when P contains a critical overlap we have two possible cases.

3.2.1The critical overlap is not joinable in R_i, but it is a member of the set $CPC(E_i, R_i)$ then we will have $P(=>_C \cup =>_{CPC})^+ P'$ where $P' \equiv P_1' P_3' P_2'$ in $E_j \cup R_j$. $j > i$ with $P_1(=>_C \cup =>_{CPC})^* P_1'$, and $P_2(=>_C \cup =>_{CPC})^* P_2'$. (by repeated application of lemma 3.2), and the proof P_3' being the proof (possibly further reduced) which replaced the critical overlap using $=>_{CPC}$. At least one step in P' is smaller than in P and hence P' can be transformed into rewrite proof Q' in R_∞.

3.2.2The critical overlap is not joinable in R_i, and is not a member of the set $CPC(E_i, R_i)$. Since it is generated by an overlap between persisting rules, by the second assumption of the fairness condition (if the CPC condition does not apply at a later stage of completion) the critical pair $< s', t' >$ that generates the critical overlap exists as an equation $s' = t'$ in some E_j. If

$j>i$ we will have $P(=>_C\cup=>_{CPC})^{+}P'$ where $P' \equiv P_1' \leftrightarrow_{E_j} P_2'$ with $P_1(=>_C\cup=>_{CPC})^{*}P_1'$, and $P_2(=>_C\cup=>_{CPC})^{*}P_2'$, (by repeated application of lemma 3.2), at least one step in P' is smaller than in P and hence P' can be transformed into a rewrite proof Q' in R_{∞}. If $j \leq i$ then will have a proof P_3 linking s' and t' that will connect the critical overlap in $E_i \cup R_i$. And we have $P' \equiv P_1 P_3 P_2$, P' is smaller than P, because P_3 connects s and t below the peak u, and so P' will eventually be transformed into rewrite proof Q' in R_{∞}.

In of the above cases we have shown that the proof P can be transformed into some proof P' such that $P'\ (=>_C\cup=>_{CPC})^{*}Q'$ where Q' is a proof in R_{∞}.

4. Finally, if P is a persisting normal rewrite proof in R_i then P will exists as a normal rewrite proof in R_{∞}.

To recap, we have shown that an arbitrary proof P in $E_i \cup R_i$ will either exist as, or can be transformed into a normal rewrite proof Q in R_{∞} using the relation $(=>_C\cup=>_{CPC})^{*}$.

Corollary

If the critical pair criteria is correct then any correct completion procedure using it will remain correct.

Proof

R_{∞} is convergent (by theorem 2.2) because it is terminating, and R_{∞} contains unjoinable critical pairs (since any critical overlap will have a persisting normal rewrite proof). Since $=>_{CPC}$ does not change the provability of R_{∞}.

by lemma 3.1 R_∞ has the same power with respect to provability as E.

A critical-pair criterion can decrease the number of critical-pairs generated during the completion process[23]. However, the efficiency gained by the criterion test. may be offset by the cost of checking it against all the critical-pairs. By providing a definition for soundness for critical-pair criteria below we can extend these tests to testing the Church-Rosser property in arbitrary systems of rewrite rules.

Definition 4.4

A criterion CPC is *sound* if, for each rewrite system R contained in $>$, the following property holds: R has the Church-Rosser property if and only if there exists a rewrite proof in R, for each critical-pair in $CP(R){-}CPC(\emptyset,R)$.

A sound critical-pair criterion may be used for testing the Church-Rosser property in finite systems or rewriting rules.

Lemma 4.2

Any correct criterion is sound.

Proof

By assumption all critical-pairs in $CP(R){-}CPC(\emptyset.R)$ are joinable. Any critical-pair in $CPC(\emptyset.R)$ is joinable because our assumption of correctness implies that there exists a normal rewrite proof in R_∞ for the zig-zag proof generated by the critical-pair criterion CPC. Since all critical-pairs are joinable, then by lemmata 2.1 to 2.3 the system R has the Church-Rosser property.

4.2. The Unblocked Criterion - BCP

The first CPC we will examine is the unblocked critical-pair criterion, first introduced by Slagle[32]; although it was applied to term rewriting theory by Lankford and Ballantyne[27].

Definition 4.5

> Let R be a rewrite system, and let $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma]$ be a critical overlap between rules $s \rightarrow t$ and $l \rightarrow r$ in R. The critical pair $<t\sigma, s\sigma[r\sigma/l\sigma]$ is called *unblocked* if any substituted variable occurrence $x\sigma$ in the overlap $s\sigma[l\sigma]$ is reduced by a rule $l' \rightarrow r'$ in R, otherwise it is called *blocked.*

If the critical-pair $<t\sigma.s\sigma[r\sigma/l\sigma]>$ is unblocked, then we have $x\sigma \rightarrow_R w$ for some variable x in s or l, and some term w. Let σ' be the substitution for which $x\sigma$ is w, while $\sigma' = \sigma$ for variables other than x. This gives rise to the following proof.

$$\begin{tikzcd}
 & s\sigma[l\sigma] \arrow[dl] \arrow[d, "R|*"'] \arrow[dr] & \\
t\sigma \arrow[d, "R|*"'] & s\sigma'[l\sigma'] \arrow[dl] \arrow[dr] & s\sigma[r\sigma] \arrow[d, "R|*"'] \\
t\sigma' & & s\sigma'[r\sigma']
\end{tikzcd}$$

Figure 4.1 - An Unblocked Critical Overlap

We will define the unblocked (sometimes called *blocking*) critical-pair criterion BCP as the following set of elimination patterns.

$$t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma] \Rightarrow$$

$$t\sigma \rightarrow_R^* t\sigma' \leftarrow_R s\sigma' \rightarrow_R s\sigma'[r\sigma'/l\sigma'] \leftarrow_R^* s\sigma[r\sigma/l\sigma]$$

with $s\sigma \rightarrow_R^+ s\sigma'$, and $s \rightarrow t$, $l \rightarrow r$, σ and σ' as described above. The set $BCP(E,R)$ contains all the unblocked critical-pairs in $CP(R)$.

Lemma 4.3

The *BCP* critical-pair criterion is correct.

Proof

To prove correctness we have to show that $=>_{BCP}$ is contained in $>_C$. This is trivial, since as we can see in figure 4.1 all elements of the right-hand proof pattern are below the overlap $s\sigma[l\sigma]$, and hence are dominated in the first component of the ordering. Hence the ordering $=>_{BCP} \cup => _C$ is well-founded.

Consider the following, alternate description of the unblocked reduction of the critical overlap.

$$\begin{tikzcd}
 & s\sigma[x\,\sigma[l'\,\sigma']] \arrow[dl] \arrow[dr] & & s\sigma[x\,\sigma[l'\,\sigma']] \arrow[dl] \arrow[dr] & \\
t\sigma & & s\sigma[x\,\sigma[r'\,\sigma']] & & s\sigma[r\,\sigma]
\end{tikzcd}$$

Figure 4.2 - An Alternate View of an Unblocked Overlap

This alternate view of an unblocked overlap gives rise to the following definition for the unblocked critical-pair criterion BCP' .

$$t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma] =>$$

$$t\sigma \leftarrow_R s\sigma[x\sigma] \rightarrow_R s\sigma[w/x\sigma] \leftarrow_R s\sigma[x\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma]$$

We will wait to prove the correctness of the critical-pair criterion BCP' after we have introduced our new general composite criterion *GCC* (the result will follow as a simple corollary).

4.3. The Connectedness Criterion - CCP

The connected critical pair criteria are all based on a lemma introduced by M.H.A Newman[28].

Definition 4.5

Let R and E be a set of rewrite rules and equations respectively, and let R be contained in the well-founded ordering $>$. Two terms s and t are *connected below u* in $E \cup R$ if there exists a proof $s \leftrightarrow_{E \cup R} u_1, \ldots, u_n \leftrightarrow_{E \cup R} t$ with

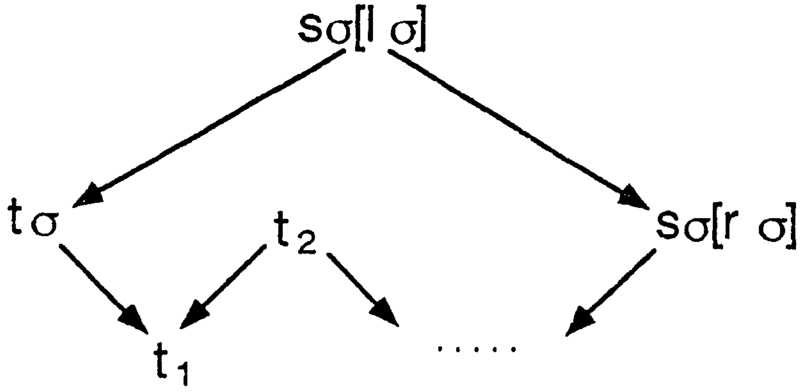

Figure 4.3 - Two terms connected below u

We define the *connected critical-pair criterion CCP* as the set of all elimination patterns

$$t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma] =>$$

$$t\sigma \leftrightarrow_{E \cup R} s\sigma[l\sigma]_1 \leftrightarrow . \ldots, u_n \leftrightarrow_{E \cup R} s\sigma[r\sigma/l\sigma]$$

with $u_i < s\sigma[l\sigma]$ for $1 \leq i \leq n$.

Lemma 4.5

The criterion *CCP* is correct.

Proof

The term u is a larger (with respect to $>$) than the terms s, t, and u_i, $1 \leq i \leq n$, the first component of the complexity measures of the proof $s \leftarrow_R u \rightarrow_R t$ is larger than the first component of the complexity measure of all the steps in $s \leftrightarrow_{E \cup R} u_1,..,u_n \leftrightarrow_{E \cup R} t$. Hence the ordering $(=>_C \cup =>_{CCP})^+$ is well-founded under the ordering $>_C$.

The use of connectedness as a technique for reducing the number of critical-pairs in a completion-like procedure was first applied to the construction of canonical bases for polynomial ideals by Bruno Buchberger[7, 8]. It was independently applied to term-rewriting theory by both Wolfgang Kuchlin[26] and Franz Winkler[35], and we describe some of their techniques below.

As we saw in the alternate view of the *BCP* if the overlap is reducible then we can decompose the critical-pair into two smaller pairs that may either be non-critical or may have been previously computed (both Kuchlin and Winkler's test provided book-keeping mechanisms for keeping track of previously computed critical-pairs, although Winkler's test restricted the position at which the overlap could be reduced).

Connectedness, unfortunately, is too general a criterion to provide much information concerning the potential joinability of critical-pairs that is significantly different from that provided by the other existing tests (especially when we consider our generalized criterion introduced below). In the composite critical-pair criteria introduced below we will extend the notion of connected below from the individual terms in a proof to the proof itself.

4.4. The Composite Criteria - PCP and GCC

The restricted form of the composite criterion was introduced by Kapur, Musser and Narendran[23].

Definition 4.6

Let R be a rewrite system, and let $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma]$ be a critical overlap between the two rule $s \rightarrow t$ and $l \rightarrow r$ in R. The critical pair $< t\sigma, s\sigma[r\sigma] >$ is considered to be *composite* if some proper subterm of $l\sigma$ is reducible by another rule $l' \rightarrow r'$ in R.

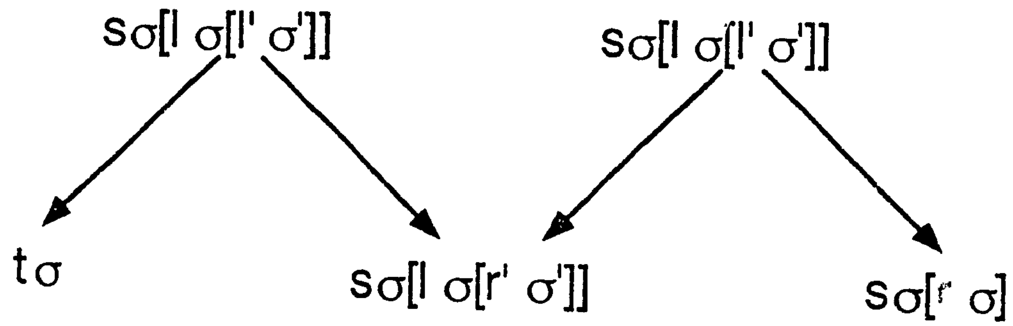

Figure 4.4 - The Composite Criterion PCP

Consider the following example from Bachmair and Dershowitz[4]. Suppose that the set R contains the following rewrite rules: $-(-x + y) \rightarrow -y + -(-x)$, $x + -x \rightarrow 0$ and $-(-x) \rightarrow x$. The first two rules define the following critical overlap.

$$-(-(-x)) + -(-x) \leftarrow_R -(-x + -(-x)) \rightarrow_R -0$$

The above overlap is composite since the subterm $-(-x)$ of $-(-x + -(-x))$ is reducible by the third rule in the set R. Definition 4.6 defines the following set of elimination patterns PCP.

$$t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma] =>$$
$$t\sigma \leftarrow_R s\sigma[l\sigma[l'\ \sigma'\ ]] \rightarrow_R s\sigma[l\sigma[r'\ \sigma'\ /l'\ \sigma'\ ] \leftarrow_R s\sigma[l\sigma[l'\ \sigma'\ ]] \rightarrow_R s\sigma[r\sigma]$$

where l' applies at a position strictly below the position at which l applies in $s\sigma$. The rewrite relation $=>_{PCP}$ can be used to eliminate the set of composite critical-pairs $PCP(E,R)$.

We will prove the correctness of the criterion PCP below in order that we may later highlight the differences between it and our new criterion GCC.

Lemma 4.6

The composite critical-pair criterion PCP is correct[4].

Proof

We will prove the correctness of PCP as a critical-pair criterion by constructing a well-founded ordering $>_{PCP}$ that contains both $=>_{PCP}$ and $=>_C$.

If we let P be the proof $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma]$ and P' be the proof $t\sigma \leftarrow_R s\sigma[l\sigma[l'\ \sigma'\ ]] \rightarrow_R s\sigma[l\sigma[r'\ \sigma'\ /l'\ \sigma'\ ]] \leftarrow_R s\sigma[l\sigma[l'\ \sigma'\ ] \rightarrow_R s\sigma[r\sigma/l\sigma]$ then using the ordering $>_C$ from the previous chapter we cannot show that $P > P'$. In fact since P' contains the term $s\sigma[l\sigma[l'\ \sigma]]$ twice, whereas P contains it only once, and we will actually have $P' > P$. So that we may order the proofs correctly we will introduce a fifth component into the complexity measure $c(s,t)$ that will reflect the amount of overlap between successive proof steps. This will allow us to successfully show that $P >_{PCP} P'$.

Let P be a proof $(s_0, \ldots, s_n)$ and let p_i be the position of the i-th proof step (s_{i-1}, s_1). By $M(P)$ we denote the multiset $[d(s_0,s_1,P), \ldots, d(s_{n-1},s_n,P)]$, where $d(s_{i-1},s_i,P)$ is

$([s_{i-1}], s_{i-1}@p_i, l, s_i, s_{i-1}, s_{i-1}@p_{i-1})$, if $s_{i-1} \rightarrow_R s_i$ by $l \rightarrow r$
(where $s_{i-1}@p_{i-1}$ is $-$, if i is 1);

$([s_i], s_i@p_i, l, s_{i-1}, s_i, s_i@p_{i+1})$, if $s_{i-1} \leftarrow_R s_i$ by $l \rightarrow r$
(where $s_i@p_{i+1}$ is $-$, if i is n);

$([s_i, s_{i-1}], -, -, -, -)$, if $s_{i-1} \leftrightarrow_E s_i$

The first four components are the same as those provided for the complexity measure $c(s,t)$. The fifth component added to the measure $d(s,t)$ reflects the amount of overlap between rewrite rule and its neighbouring proof step. We let the ordering $>_{PCP}{}'$ be the lexicographic combination of the multiset extension $>>$ of the well-founded ordering $>$, the proper subterm ordering, the proper subsumption ordering $>_s$, the ordering $>$, and the proper subterm ordering.

We now consider the ordering $>_{PCP}$ defined in the following fashion $P >_{PCP} P'$ iff $M(P) >>_{PCP}{}' M(P')$. This ordering contains both the orderings $=>_C$ and $=>_{PCP}$ and hence can be used to prove the correctness of the criterion PCP†^{2}.

4.4.1. The Generalized Composite Criterion - GCC

We will now consider how to generalize the ordering $>_{PCP}$ so that we may remove the restriction that the rule $l' \rightarrow r'$ which reduces the critical overlap $s\sigma[l\sigma]$ be a proper subterm of the term $l\sigma$.

[†] The original ordering $>_C$ provided by Bachmair and Dershowitz used the *proper subterm ordering* to order the second component. This is consistent with our definition for $>_C$, since our abstract ordering $>$ (which we use for the second component) has the *subterm property*.

Definition 4.7

Let R be a rewrite system and $t\sigma \leftarrow s\sigma[l\sigma] \rightarrow s\sigma[r\sigma/l\sigma]$ be a critical overlap between rules $s \rightarrow t$ and $l \rightarrow r$ in R. The critical pair $<t\sigma, s\sigma[r\sigma/l\sigma]>$ is called *composite* if the critical overlap $s\sigma[l\sigma]$ is reducible by another rule $l' \rightarrow r'$ also in R, and $l\sigma > l'\sigma'$, $l\sigma = l'\sigma'$ and $l >_s l'$, or $l\sigma = l'\sigma'$, $l = l'$ and $r > r'$.

$$\begin{tikzcd}
 & s\sigma[l\sigma][l'\sigma'] \arrow[dl] \arrow[dr] & & s\sigma[l\sigma][l'\sigma'] \arrow[dl] \arrow[dr] & \\
t\sigma & & s\sigma[l\sigma][r'\sigma'] & & s\sigma[r\sigma]
\end{tikzcd}$$

Figure 4.5 - The Composite Criterion GCC

Definition 4.7 defines the following elimination pattern GCC; note: we indicate the fact that the term $l\sigma'$ need not be a proper subterm of $l\sigma$ (in fact it may occur anywhere in the term $s\sigma$) by writing the critical overlap as $s\sigma[l\sigma][l'\sigma']$ (cf. definition 4.6).

$$t\sigma \leftarrow s\sigma[l\sigma] \rightarrow s\sigma[r\sigma/l\sigma] \Rightarrow$$

$$t\sigma \leftarrow_R s\sigma[l\sigma][l'\sigma'] \rightarrow_R s\sigma[l\sigma][r'\sigma'/l'\sigma'] \leftarrow_R s\sigma[l\sigma][l'\sigma'] \rightarrow_R s\sigma[r\sigma/l\sigma]$$

Lemma 4.7

The composite criterion GCC is correct.

Proof

In a fashion similar to the proof of lemma 4.6 we construct a well-founded ordering $>_{GCC}$ that contains both $=>_{GCC}$ and $=>_C$.

Let P be a proof $(s_0, \ldots, s_n)$, by $M(P)$ we denote the multiset $[g(s_0,s_1), \ldots, g(s_{n-1},s_n)]$ where $g(s_{i-1,}s_i)$ is

$(s_{i-1}, l\sigma, l, s_i, l' \sigma', l', s_{i-2})$ if $s_{i-1} \rightarrow_R s_i$ by $l \rightarrow r$ and $s_{i-1} \rightarrow_R s_{i-2}$ by $l' \not\rightarrow$

(where $l' \sigma'$, l', and s_{i-2} are $-$ if $i=1$)

$(s_i, l\sigma, l, s_{i-1}, l' \sigma', l', s_{i+1})$ if $s_{i-1} \leftarrow_R s_i$ by $l \rightarrow r$ and $s_i \rightarrow_R s_{i+1}$ by $l' \rightarrow r'$

(where $l' \sigma'$, l', and s_{i+1} are $-$ if $i=n$)

$(s_{i-1}\# s_i, -, -, -, -, -, -)$ if $s_{i-1} \leftrightarrow_E s_i$

The first four components are consistent with the complexity measure for $=>_C$ and the last three reflect the amount of overlap between neighbouring rewrite steps. We let the ordering $>_{GCC'}$ be the lexicographic combination of the multiset extension $>>$ of the well-founded ordering $>$, the ordering $>$, the proper subsumption ordering $>_s$, the ordering $>$, the ordering $>$ again, the proper subsumption ordering $>_s$, and finally the ordering $>$.

We consider the ordering $>_{GCC}$ as $P >_{GCC} P'$ iff $M(P) >>_{GCC'} M(P')$. Since $=>_C$ is contained in $=>_{GCC}$ (which is well-founded) our criterion GCC is correct. We explain, in a manner similar to the previous lemma, the reason for the last two components.

Let P be the overlap $t\sigma \leftarrow_R s\sigma[l\sigma] \rightarrow_R s\sigma[r\sigma/l\sigma]$, and let P' be equal to $t\sigma \leftarrow_R s\sigma[l\sigma][l' \sigma'] \rightarrow_R s\sigma[l\sigma][r' \sigma' / l' \sigma'] \leftarrow_R s\sigma[l\sigma][l' \sigma'] \rightarrow_R s\sigma[r\sigma/l\sigma]$ where $l\sigma > l' \sigma'$, or $l\sigma = l' \sigma'$ and $l >_s l'$. The two proof steps $s\sigma[l\sigma][l' \sigma'] \rightarrow s\sigma[l\sigma][r' \sigma' / l' \sigma']$ are both smaller than $s\sigma[l\sigma] \rightarrow t\sigma$ and $s\sigma[l\sigma] \rightarrow s\sigma[r\sigma/l\sigma]$ because they are smaller in the second and third positions; but if we just consider the first four components the we cannot have $P > P'$ since both proofs contain $s\sigma[l\sigma] \rightarrow t\sigma$ and $s\sigma[l\sigma] \rightarrow s\sigma[r\sigma]$. The ordering

> *GCC* correctly handles this problem since when $l\sigma$ is greater than $l' \sigma'$ the fifth component is larger in P than P', when $l\sigma$ and $l' \sigma'$ are equal the requirement that l be greater than l' under the subsumption ordering makes the sixth component larger, and when l and l' are equal the seventh component is larger.

This removes the .equirement (of both Bachmair and Kapur) that $l' \sigma'$ be a proper subterm of $l\sigma$. Kapur et al. gave an example that showed the need for this restriction on their definition of compositeness but we will now show that *GCC* correctly handles their example.

Consider the following system of rewrite rules.

1. $g(f(x,x)) \rightarrow i(x)$
2. $f(x,a(y)) \rightarrow j(x)$
3. $f(a(x),y) \rightarrow j(a(x))$

Kapur et al. argued that if the restriction of $l' \sigma'$ being a proper subterm of $l\sigma$ was lifted the superposition between rule 1 and rule 3, $g(f(a(x),a(x)))$ would be considered composite; yet its critical pairs $g(j(a(x)))$ and $i(a(x))$ are equivalent but not joinable. Under *GCC* this superposition is not considered composite because the left-hand side of rule 3 is not greater than the left-hand side of rule 2 under the subsumption ordering. It is important to note that the position $l' \sigma'$ is not restricted to be a sub-position of $l\sigma$ but may occur anywhere within the term $s\sigma$.

We now consider the following example which shows how our extension of the composite criterion, *GCC*, handles two cases not recognized by the previous composite criterion, *PCP*. of Bachmair and Dershowitz. The first example removes the requirement that l' be a proper subterm of l, where $l \rightarrow r$ and another rule $s \rightarrow t$ form a composite overlap. The second example will remove the restriction that l' be a subterm of l.

Consider the following system of rewrite rules.

1. $h(f(a,y)) \rightarrow g(a,y)$
2. $f(a,b) \rightarrow b$
3. $f(x,b) \rightarrow c$

In this system we have a composite overlap between rule 1 and rule 2, and non-composite overlaps between rules 1 and 3, and rules 2 and 3. The critical pair between rule 1 and rule 2, $g(a,b)=h(b)$ can be considered to be superfluous because it will be joinable once we have oriented the critical pairs between rules 1 and 2, and rules 2 and 3. We would still arrive at the same final system if we had used the *PCP* criterion but the rule derived from the critical pair between rules 1 and 2 would have been added and then removed during the completion process.

Consider now the following system of rewrite rules.

1. $h(g(x),f(x,b)) \rightarrow h(x,b)$
2. $g(a) \rightarrow b$
3. $f(a) \rightarrow c$

We have (by our definition) a composite overlap between rule 1 and rule 2 (rule 2 matches with the overlap $h(g(a),f(a))$ and $f(u)<g(a)$). We can consider the critical pair $h(a)=b$ to be superfluous because it will be joinable once we have computed the critical pair between rule 1 and rule 3.

Thus, our criterion *GCC'* is more general than the test *PCP*. The generality of our test is expected to reduce the number of equations and rewrite rules generated and hence the cost of the completion procedure.

Current Implementation and Future Plans

In this section we will discuss the current stage of our implementation work, and our plans for the future (both theoretical and practical). Currently we have implemented a term rewriting system compiler (TRS compiler) based on the program developed by Yoshihito Toyama of the NTT Basic Research Laboratories in Tokyo[34], his program based, in turn, on the techniques developed by S. Kaplan for a conditional term rewriting system compiler (CTRS compiler)[22].

In systems that interpret a set of rules that make up a term rewriting system, the expression being reduced is usually considered as a tree. The left-hand sides of rules are matched onto subtrees of this tree and the rewriting process is effected by replacing the matched subtree with tree representing the right-hand side of the rule (with the appropriate variable substitutions).

Pattern matching in trees is a slow process (the complexity is $O(\,|pattern| \; x \; \# \; of \; patterns \; x \; |expression|\,))$. It can be speeded up by preprocessing the tree patterns to be matched into some form of tree walking automata[17, 30]. Furthermore, the amount of work in locating redex positions may be reduced by using information from previous searches to restrict the positions within the expression at which we apply the matching process[25]. Unfortunately, often the gains in efficiency from faster matching do not make up for the added cost of processing the rules. This is especially true in situations such as the Knuth-Bendix completion procedure where the rule set is constantly expanding and contracting.

In our system we take a slightly different approach to finding redex positions in the expression we are trying to reduce. Our approach is to turn the operators of an equational theory into functions. The result of applying the function to its arguments (whatever they may be in the expression) is the the appropriate replacement term.

The speed of our system is derived from the fact that matching is achieved by function application; a relatively fast process because it uses the internal mechanisms of the lisp system. Lisp is an obvious language in which to implement our compiler because it is very easy to build and manipulate lisp functions · lisp data objects. Also, the fact that there is a standard order of evaluation for lisp functions allows us to provide more general conditions for confluence (e.g. no overlaps on outermost function symbols) than in the case when the position of rewriting is unrestricted.

An important question we will be examining is how to integrate built in functions and data types into our systems of equations. Consider the example of factorial function in the introduction section. It returns a normal form in terms of the successor function s() (.e.g. **fact(s(s(s(0)))) = s(s(s(s(s(s(0))))))**). Term rewriting systems perform their computations by manipulating the symbols representing a value. If we wish to use actual values and built in functions that use these values we will have to define a suitable conversion process between an object's value and its symbolic representation.

We will continue the ongoing work of implementing the Knuth-Bendix completion procedure for unconditional TRS's. We hope to use our completion procedure as an experimental platform with which to demonstrate the efficacy of our new more general critical-pair criterion in controlling the number of critical-pairs and superfluous rules generated in the completion process.

We have already extended our TRS compiler so that it will compile conditional equations. Our future theoretical work will examine the process of completion as it is applied to conditional equational systems. Some exploratory work[14] has already been done in extending the framework of Bachmair and Dershowitz to conditional systems, and we hope to carry on with this work.

In continuing our interest in the applications of term rewriting theory to the practical problems of programming. We wish to examine method of defining conditional equational systems as programming languages (with special reference to integrating the narrowing technique into the reduction process so that we may unify functional and logic programming). We also wish to pay special attention to the problems of parameterizing and modularizing our equational theories, so that they will be practical systems in which to develop software.

Finally, we will examine the proof normalization process as a framework in which to do equational theorem proving in equational systems that do not complete to a finite set of rewrite rules. As noted previously, this technique will not only provide us with information that may make the process of selecting equations for completion more efficient, but may also allow us to turn the semi-decision procedure for equality in these systems into a decision procedure.

It is my belief, and the general philosophy of our team, that the theory of term rewriting systems has a great number of potential applications to the field of computer science, but its utility can only be realized if the practical systems still remain firmly grounded in the mathematical theory from which they were derived. Maintaining the mathematical purity of the theory not only allows us to integrate our results into the existing framework of mathematical logic; but also allows to achieve gains in efficiency, not by ad hoc measures that restrict the power of our systems, but by providing better theorems, or better proofs of existing theorems.

References

1. Ackermann, W., "Konstruktiver Aufbau eines Abschnitts der zweiten Cantorischen Zahlenklasse," *Math Z.*, vol. 53, pp. 403-413, 1951.

2. Bachmair, L., N. Dershowitz, and J. Hsiang, "Orderings for Equational Proofs," in *Proc. IEEE Symp. Logic in Computer Science*, pp. 346-357, Cambridge, Massachusetts, 1986.

3. Bachmair, L., "Proof Methods for Equational Theories," PhD Thesis, University of Illinois at Urbana-Champaign, 1987.

4. Bachmair, L. and N. Dershowitz, "Critical Pair Criteria for Completion," *J. Symbolic Computation*, vol. 6, pp. 1-18, 1988.

5. Barendregt, H. P., *The Lambda Calculus: Its Syntax and Semantics*, North-Holland, Amsterdam, 1984.

6. Brouwer, L.E.J., in *Collected Works*, ed. A. Heyting, vol. 1, North-Holland, Amsterdam, 1975.

7. Buchberger, B., "A Criterion for Detecting Unnecessary Reductions in the Contruction of Grobner Bases," in *Proc. Eurosam '79*, ed. W. Ng, vol. LNCS 72, Springer-Verlag, Berlin, 1979.

8. Buchberger, B., "A Critical-Pair/Completion Algorithm for Finitely Generated Ideals in Rings," in *Proc. Symp. Rekursive Kombinatoric*, ed. E. Boerger, vol. LNCS 171, Springer-Verlag, Berlin, 1984.

9. Dershowitz, N. and Z. Manna, "Proving Termination with Multiset Orderings," *CACM*, vol. 22, pp. 465-476, 1979.

10. Dershowitz, N., "A Note on Simplification Orderings," *Inf. Proc. Lett.*, vol. 9, pp. 212-215, 1979.

11. Dershowitz, N., "Termination of Rewriting," *Journal of Symbolic Computation*, vol. 3, pp. 69-116, 1987.

12. Dershowitz, N. and M. Okada, "Proof-Theoretic Techniques for Term Rewriting Theory," in *Proc. of the Third Annual Symposium on Logic in Computer Science*, Edinburgh, 1988.

13. Ehrig, H. and B. Mahr, *Fundamentals of Algebraic Specification 1*. Springer-Verlag, Berlin. 1985.

14. Ganzinger, H., "A Completion Procedure for Conditional Equations." in *Conditional Term Rewriting Systems. LNCS 308*, ed. J.-P. Jouannaud. Springer-Verlag, Berlin. 1987.

15. Halmos, P. R., *Naive Set Theory*, Springer-Verlag, New York, 1974.

16. Higman, G., "Ordering by Divisibility in Abstract Algebra." *Proc London Mathematical Society, Series 3*, vol. 2, 1952.

17. Hoffman. C. M. and M. J. O'Donnell. "Pattern Matching in Trees." *JACM*. vol. 29. no. 1, pp. 68-95, 1982.

18. Hoffman, C. M. and M. J. O'Donnell, "Programming with Equations." *ACM Transactions on Programming Languages and Systems*. vol. 4, no. 1. pp. 83-112, 1982.

19. Huet. G.. "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *Journal of the Association for Computing Machinery*, vol. 27, no. 4. pp. 797-821, 1980.

20. Huet, G.. "A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm," *Journal of Computer and System Sciences*, vol. 23. pp. 11-21. 1981.

21. Kamin. S. and J.-J. Levy, "Two Generalizations of the Recursive Path Ordering," Unpublished Report. Dept. of Computer Science, University of Illinois at Urbana-Champaign. 1980.

22. Kaplan. S., "A Compiler for Conditional Term Rewriting Systems." in

LNCS, 256, pp. 24-41, Springer Verlag, Berlin, 1987.

23. Kapur, D., D. R. Musser, and P. Narendran, "Only Prime Superpositions Need to be Considered in the Knuth-Bendix Completion Procedure," *J. Symbolic Computation*, vol. 6, pp. 19-36, 1988.

24. Knuth, D. E. and P. B. Bendix, "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*, ed. John Leech, Pergammon Press, 1970.

25. Kuchlin, W., "Some Reduction Strategies for Algebraic Term Rewriting." *SIGSAM*. vol. 16. no. 4. pp. 13-23, 1982.

26. Kuchlin, W., "A Confluence Criterion Based on the Generalised Newman Lemma," in *Eurocal '85 - LNCS 204*, ed. Bob F. Caviness, Springer-Verlag, Berlin, 1985.

27. Lankford, D. and A. Ballantyne, "The Refutation Completeness of Blocked Permutative Narrowing and Resolution," in *Proc. Fourth Workshop on Automated Deduction*, ed. W. H. Joyner Jr., pp. 168-174, Austin, Texas, 1979.

28. Newman, M. H. A., "On Theories with a Combinatorial Definition of "Equivalence"," *Ann. Math*, vol. 43, no. 2, pp. 223-243, 1942.

29. O'Donnell, M. J., *Equational Logic as a Programming Language*, MIT Press, Cambridge, MA, 1985.

30. Purdom, P. Walton Jr. and C. A. Brown., "Fast Many-to-one Matching Algorithms," in *Rewriting Techniques and Applications '85*, ed. J. P. Jouannaud, Springer-Verlag, Berlin, 1985.

31. Schutte, K., *Proof Theory*, Springer-Verlag, Berlin, 1977.

32. Slagle, J. R., "Automated Theorem Proving for Theories with Simplifiers, Commutativity and Associativity," *JACM*. vol. 21, pp. 622-642, 1974.

33. Thatcher, J. W., E. G. Wagr . and J. B. Wright, "Data Type Specification: Parameterization and the Power of Specification Techniques," in *Tenth Annual ACM Symposium on Theory of Computing.* ACM. New York, 1978.

34. Toyama, Y., "Fast Knuth-Bendix Completion with a Term Rewriting System Compiler," IEICE technical report, pp. 17-26, February, 1989.

35. Winkler, F., "Reducing the Complexity of the Knuth-Bendix Completion Algorithm: A "Unification" of Different Approaches," in *Eurocal '85 - LNCS 204*, ed. B. F. Caviness, pp. 3-38. Springer-Verlag, Berlin, 1985.

Appendix

```
;; TRS compiler - main program
;;
;; This section reads in the rules and functional forms and
;; compiles them to a set of functions that are then made
;; part of the lisp system.
;;
;; The program then sets up a "read - eval" loop using these
;; new functions to reduce and expression to its normal form.
;;
;; The functions below define, and access the data structures
;; in the program.

;; Input rules - ( l1=r1 l2=r2 ...)

(defun get-lhs-ir   (input-rule) (car input-rule))
(defun get-rhs-ir   (input-rule) (caddr input-rule))
(defun remaining-ir (input-rule) (cdddr input-rule))

;; Rules - ((l1 r1) (l2 r2) ...)

(defun mk-rule (lhs rhs) (list lhs rhs))
(defun get-rhs-r  (rule) (cadr rule))
(defun get-fn-r   (rule) (caar rule))
(defun get-args-r (rule) (cdar rule))

;; Forms - ((fn v1 v2 ...) (fn v1 v2 ...) ...)

(defun mk-forms (input-forms) input-forms)
(defun get-fn-f   (form) (car form))
(defun get-vars-f (form) (cdr form))

;; F-rules ((fn v1 v2 ...) ((l1 r1) (l2 r2) ...)

(defun get-form-fr  (f-rule) (car f-rule))
(defun get-rules-fr (f-rule) (cdr f-rule))
(defun get-rule1-fr (f-rule) (cadr f-rule))
```

```
;; Main program
;;
;; The following two functions read in the rules, and set up
;; the read eval loop.

(defun trs ()
      (let ((ax-file) (input) (rules) (forms))
            (terpr)
            (terpr)
            (princ '"Please type an axiom file name (trs) ? ")
            (setq ax-file (concat (read) '.axm))
            (setq input (infile ax-file))
            (setq rules (mk-rules (read input)))
            (setq forms (mk-forms (read input)))
            (trs-comp rules forms)
            (close input)
            (trs-1)
      )
)


(defun trs-1 ()
      (prog (term)
       loop
            (terpr)
            (terpr)
            (princ '"Please enter a term as an s-expression (nil to end)")
            (terpr)
            (terpr)
            (princ '"~> ")
            (setq term (read))
            (cond ((null term) (return nil)))
            (terpr)
            (terpr)
            (princ term)
            (princ '" = ")
            (nf term)
            (go lcop)
      )
)
```

;; This function computes the normal form of the supplied term.

```
(defun nf (term)
      (let ((eval-term))
          (setq eval-term (eval (mk-term term)))
          (princ eval-term)
      )
)
```

;; This function generates the internal representation of the
;; rewrite rules.

```
(defun mk-rules (input-rules)
      (cond ((null input-rules) nil)
           (t (cons (mk-rule  (get-lhs-ir input-rules)
                              (get-rhs-ir input-rules))
                  (mk-rules (remaining-ir input-rules)))
           )
      )
)

(defun terpr () (terpri))
```

```
;; This section takes a set of rules and functional forms and
;; pairs the rules with the corresponding functional form to
;; produce an "f-rule". These f-rules are then compiled into
;; lisp procedures that are then made part of the system using
;; the function "putd".

(defun trs-comp (rules forms)
     (mapcar 'compile (mk-f-rules forms rules))
)

(defun mk-f-rules (forms rules)
     (cond ((null forms) nil)
         (t (cons (mk-f-rule (car forms) rules)
                (mk-f-rules (cdr forms) rules)
           )
         )
     )
)

(defun mk-f-rule (form rules)
     (cons form
         (mapcan
            (quote
            (lambda (rule)
                 (cond ((eq (get-fn-f form)
                         (get-fn-r rule)) (list rule))
                      (t nil)
                 )
            )
            )
            rules
         )
     )
)
```

```
;; The following function takes an f-rule and depending on the
;; arity of the function generates lisp code that implements
;; the function.

(defun compile (f-rule)
     (cond ((null (get-rules-fr f-rule)) (compile-external f-rule))
           (t (cond ((null (get-vars-f (get-form-fr f-rule)))
                          (compile-arity-0 f-rule))
                    (t (compile-arity-n f-rule))
              )
           )
     )
)

(defun compile-external (f-rule)
     (let ((form (get-form-fr f-rule)))
          (mk-func (concat '$ (get-fn-f form))
                   (list 'lambda (get-vars-f form) (mk-default form))
          )
     )
)
```

```
(defun compile-arity-0 (f-rule)
     (mk-func (concat '$ (get-fn-f (get-form-fr f-rule)))
           (list 'lambda () (mk-rhs (get-rhs-r (get-rule1-fr f-rule))))
     )
)

(defun compile-arity-n (f-rule)
     (let ((form (get-form-fr f-rule))
          (vars (get-vars-f (get-form-fr f-rule)))
          (args-l (mapcar 'get-args-r (get-rules-fr f-rule)))
          (rhs-l  (mapcar 'get-rhs-r  (get-rules-fr f-rule)))
     )
        (mk-func (concat '$ (car form))
              (list 'lambda
                   vars
                  (list 'let
                      (mapcar 'ncons (mapcar 'mk-var vars))
                      (append '(cond)
                            (mapcar
                              (quote
                              (lambda (args rhs)
                                   (list
                                       (mk-match args vars)
                                       (mk-rhs rhs)
                                   )
                              )
                              )
                            args-l
                            rhs-l
                            )
                       (list (list 't (mk-default form)))
                       )
                   )
               )
        )
     )
)
```

```
;; The following function converts a term into a form that is
;; suitable for reduction i.e. all operators have a '$' prepended
;; to them so that they may be reduced by the internal functions.

(defun mk-term (term)
     (cond ((null term) nil)
          ((atom term) (list 'quote term))
          (t (cons (concat '$ (car term)) (mapcar 'mk-term (cdr term))))
     )
)

;; The following function makes the lisp code that returns the
;; right-hand side of a rewrite rule.

(defun mk-rhs (term)
     (cond ((null term) nil)
          ((var-p term) term)
          ((atom term) (list 'quote term))
          (t (cons (concat '$ (car term)) (mapcar 'mk-rhs (cdr term))))
     )
)

;; This function returns the code that covers the case when no
;; match has been made.

(defun mk-default (form)
     (cons 'list (cons (list 'quote (get-fn-f form)) (get-vars-f form)))
)
```

```
;; This section takes the argument portion of the left hand sides of
;; rewrite rules and builds a 'cond' term that will recognise it,
;; binding the functions variables in the process.

(defun mk-match (p1 p2)
     (let ((bound-vars))
        (cons 'and (mapcan 'mk-match-1 p1 p2))
     )
)

(defun mk-match-1 (p1 p2)
     (cond ((var-p p1) (cond ((member p1 bound-vars)
                               (list (list 'equal p1 p2))
                             )
                             (t (let ()
                                  (setq bound-vars (cons p1 bound-vars))
                                  (list (list 'setq p1 p2))
                                )
                             )
                       )
          )
          ((atom p1) (list (list 'equal (list 'quote p1) p2)))
          (t (list (append '(and) (mk-match-1 (car p1)
                                       (list 'safe-car p2))
                                   (mk-match-1 (cdr p1) (list 'cdr p2))
                   )
             )
          )
     )
)

(defun safe-car (x) (cond ((listp x) (car x)) (t nil)))
```

```
;; This function puts the built function into the lisp system.

(defun mk-func (name body)
      (let ()
            (putd name body)
            (terpr)
            (print name)
            (terpr)
            (print body)
            (terpr)
      )
)

(defun var-p (term)
      (cond ((atom term) (cond ((eq '? (car (explode term))) t)
                                    (t nil)
                              )
            )
            (t nil)
      )
)

(defun mk-var (name)
      (concat '? name)
)

(trs)
```