Knuth-Bendix Completion with a Termination Checker

Ian Wehrman (iwehrman@cs.utexas.edu)
The University of Texas at Austin

Aaron Stump (stump@cse.wustl.edu) and Edwin Westbrook (emw1@cec.wustl.edu)
Washington University in St. Louis

Abstract. Knuth-Bendix completion takes as input a set of universal equations and attempts to generate a convergent rewriting system with the same equational theory. An essential parameter is a reduction order used at runtime to ensure termination of intermediate rewriting systems. Any reduction order can be used in principle, but modern completion tools typically implement only a few classes of such orders (e.g., recursive path orders and polynomial orders). Consequently, the theories for which completion can possibly succeed are limited to those compatible with an instance of an implemented class of orders. Finding and specifying a compatible order, even among a small number of classes, is challenging in practice and crucial to the success of the method.

In this work, a new variant on the Knuth-Bendix completion procedure is developed in which no order is provided by the user. Modern termination-checking methods are instead used to verify termination of rewriting systems. We prove the new method correct and also present an implementation called SLOTHROP which obtains solutions for theories that do not admit typical orders and that have not previously been solved by a fully automatic tool.

Keywords: automated equational theorem proving, term rewriting, word problem, Knuth-Bendix completion, termination

1. Introduction

Knuth-Bendix completion is a well-known equational automated theorem proving technique for solving the word problem for a finite set of identities (a *theory*) E. This procedure requires the user to provide as input both the theory E and an ordering on terms that orients each identity in E. Furthermore, this input order must satisfy certain properties, including well-foundedness, which make it a *reduction order*. Discovering an appropriate reduction order can be extremely difficult and limits the usefulness of completion techniques in practice.

All past implementations of completion procedures have allowed the user to specify only those orders that fall into a small handful of classes. There are a number of drawbacks with this approach. First, there are many theories that cannot be oriented using the few classes of orders usually implemented by these tools. Consequently the word problem for these theories cannot be solved in practice with completion

because the complex orders needed to orient the given identities are not supported by the tool. Second, even if a theory is compatible with an order in a class supported by the tool, the user is not given help in finding the correct one. And the task of finding a compatible order is computationally difficult in its own right—typically NP-hard or, more often, undecidable.

In this work, we address these problems with a new variation on the Knuth-Bendix completion procedure. In our revised procedure, the user need not provide a reduction order as input. Instead, a heuristic search is performed for an appropriate order. In the theoretical treatment of our procedure, we assume the ability to identify reduction orders, in order to bound the search. In practice, this ability is replaced by an automated *termination checker*, which determines a property that coincides with the existence of compatible reduction orders. There are a variety of mature tools for termination checking, any of which may be integrated with our procedure to discover reduction orders.

By leveraging a termination checker, we solve most of the above problems with standard completion. Because modern termination checkers can detect reduction orders for a wide variety of theories (as opposed to the small handful found in other completion implementations), implementations of our procedure can succeed on a wider variety of theories in practice. Also, because the order is discovered by the tool instead of the user, a burden is removed that can potentially allow completion to succeed for theories which, though they may admit an order implemented by previous tools, would be otherwise difficult to discover manually.

Contributions The primary result of this work is a completion algorithm that uses a termination checker instead of a user-supplied order. In more detail, our major contributions include:

1. A new semidecision procedure for automatically solving the word problem is developed. It produces convergent completions for many theories, yielding decision procedures for such theories. The new procedure allows convergent completions to be discovered for theories that have never been discovered automatically.

2. A complete proof of correctness for the algorithm is presented.

3. An implementation of the new algorithm is presented. This tool, which we call SLOTHROP and whose source code is freely available[1], is written in the Ocaml programming language and integrates with

[1] `http://www.cs.utexas.edu/~iwehrman/slothrop.html`

the automated termination checkers AProVE (Giesl et al., 2004) and TPA (Koprowski, 2006).

4. Slothrop yielded the first fully automatically constructed convergent completion of the theory of two commuting group endomorphisms (described in Sec. 7). Although other tools can automatically discover ground-convergent completions, and manual methods have been used to discover the convergent completion (Stump and Löchner, 2006), no other tool has been able to produce such a convergent completion without user guidance.

A short version of this paper appeared in conference form (Wehrman et al., 2006).

2. Preliminaries

Throughout this paper, we rely on standard terminology and theorems from the field of term rewriting as found in a variety of surveys and textbooks (Baader and Nipkow, 1998; Dershowitz and Plaisted, 2001; TeReSe, 2003; Dershowitz and Jouannaud, 1990).

In Sect. 2.1, we briefly describe our notion of inference systems, their executions and relations among them. In Sect. 2.2, we describe Knuth-Bendix completion, the foundation of our algorithm.

Typographic Conventions We use the following typographic conventions throughout. Function symbols, constants, variables and metavariables are lowercase italic letters. For function symbols we use f, g, h; for constants we use a, b, c; for variables we use x, y, z; for metavariables we use s, t, u, v. Sets, including term rewriting systems, are capitalized italic letters: $A, B, C, \ldots$. Executions and ordinals are all lowercase italic Greek letters. For executions we use α, β, γ; for ordinals we use ι, κ, λ. Inference systems are named with calligraphic capital letters $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$, and their rules are labeled with words set in small caps.

2.1. Executions and Simulations

In this subsection we briefly and informally describe inference systems, executions of those systems, and simulations between systems.

An *inference system* $\mathcal{B}$ is a pair consisting of a set of start states from a universe $\mathscr{U}$, and a set of inference rules, which are partial functions from $\mathscr{U}$ to $\mathscr{U}$. An *execution* is a sequence of states related to one another by the rules of some inference system. More formally, executions are functions that map ordered indexes to a states. Since we

consider both finite and infinite executions, we will define an execution β as a function of type $\lambda \mapsto \mathscr{U}$, with λ a countable ordinal. We say an execution $\beta : \lambda \mapsto \mathscr{U}$ has *length* λ, which we write $|\beta|$. In this work, the universe of states will consist of tuples of rewriting systems. For an ordinal ι and execution β, we may write either $\beta(\iota)$ or β_ι to denote the state at index ι, assuming $|\beta| > \iota$. State $s \in \mathscr{U}$ is a *deduction* of execution β if there is some index ι such that $\beta_\iota = s$. We informally write executions of system $\mathcal{B}$ by

$$s_0 \vdash_{\mathcal{B}} s_1 \vdash_{\mathcal{B}} s_2 \vdash_B \cdots,$$

and also $\beta = \beta' \vdash_{\mathcal{B}} s_{n+1}$ to mean that the function β extends the function β' such that $\beta_{n+1} = s_{n+1}$.

Finite executions are hence constructed by recursion, and infinite executions by transfinite recursion. We say an execution β of system $\mathcal{B}$ is *valid* if it is a well-defined function and

1. β_0 is a start state of $\mathcal{B}$;
2. if $\beta_{\kappa'}$ (where κ' denotes the successor of ordinal κ) is defined, then for some inference rule $\rho \in \mathcal{B}$ it is the case that $\rho(\beta_\kappa) = \beta_{\kappa'}$.

Note that this definition gives some latitude in the definition of infinite executions.

Throughout this work, we rely on proofs by *simulation* (Lynch and Vaandrager, 1993; Milner, 1989). In this context, we consider a simulation from a system $\mathcal{B}_1$ to a system $\mathcal{B}_2$ as a relation that maps executions of $\mathcal{B}_1$ to executions of $\mathcal{B}_2$ that are "equivalent" in the sense that the new execution contains the same sequence of states, possibly modulo extraneous elements irrelevant to the opposite system. When there is a simulation from $\mathcal{B}_1$ to $\mathcal{B}_2$, we say that $\mathcal{B}_2$ *simulates* $\mathcal{B}_1$, denoted $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$. We rely on the fact that if $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$, then for every execution of $\mathcal{B}_1$ there exists an equivalent execution of $\mathcal{B}_2$. Hence, in this case, properties of executions of $\mathcal{B}_2$ hold for executions of $\mathcal{B}_1$.

2.2. Knuth-Bendix Completion

Once an equational theory has been shown to be convergent, we can solve its word problem by simply rewriting terms to normal forms and testing syntactic equality; the normal forms are equal iff the identity holds in the theory. But what about a non-convergent theory E? The word problem is not decidable in general, so we cannot hope to find a general method to solve the problem for all theories. However, in some cases we can find a different set of identities E' which is equivalent to E (i.e., $\overset{*}{\leftrightarrow}_E = \overset{*}{\leftrightarrow}_{E'}$) and convergent. If we can find such a theory, then

we can solve $\approx_{E'}$ and therefore $\approx_E$. We now describe a procedure for discovering such equivalent, convergent theories.

A *Knuth-Bendix completion procedure* (or simply a *completion procedure*) (Knuth and Bendix, 1970), invented in 1967 by Donald Knuth, refers to an instance of a general class of algorithms that take as input a finite set of identities E and a reduction order and which attempts to construct an equivalent convergent rewrite system R. The process is iterative, generating a sequence of intermediate rewriting systems which have theories that are, roughly speaking, successively better approximations of the input theory E. The given reduction order is used to ensure that each intermediate rewriting system is terminating and hence can be used to calculate normal forms during execution in a finite number of steps.

As an example, the theory of groups (G) is presented in Fig. 1. Group theory is particularly well-suited to automated theorem proving procedures such as completion because it is axiomatized by just a few identities. For ease of presentation, we use infix symbols such as $*$ as shorthand for function symbols as described above.

$$1 * x \approx x \quad x^{-1} * x \approx 1 \quad (x * y) * z \approx x * (y * z)$$

Figure 1. The theory of groups (G)

One possible completion of G is shown in Fig. 2, a ten-rule term rewriting system. It is easy to see that all the rewrite rules in the completion are compatible with a lexicographic path order with precedence $-^{-1} > * > 1$. Note however that orders based on the size of the terms or the subterm order are not sufficient to prove that the system is terminating.

$1 * x \to x$	$x * 1 \to x$	$1^{-1} \to 1$
$(x^{-1})^{-1} \to x$	$(x * y)^{-1} \to y^{-1} * x^{-1}$	$(x * y) * z \to x * (y * z)$
$x * x^{-1} \to 1$	$x^{-1} * x \to 1$	
$x * (x^{-1} * y) \to y$	$x^{-1} * (x * y) \to y$	

Figure 2. A convergent completion of G

Leo Bachmair reformulated the original Knuth-Bendix completion procedure, which was originally published as a deterministic algorithm, as a nondeterministic equational inference system (Bachmair, 1991), and proved it correct (stated in Sect. 2.2.1). We refer to this standard

inference system as $\mathcal{C}$ because it will serve as the basis of a correctness condition for a refinement of the procedure that will be developed later. The rules of the inference system $\mathcal{C}$ are shown in Fig. 3. The notation $s \mathrel{\dot{\approx}} t$ means either $s \approx t$ or $t \approx s$. The notation $s \xrightarrow[R]{\sqsupset} v$ means that the term s is reduced with a rule $l \to_R r \in R$ such that l is not reducible by the rule $s \to_R t$. This technical side-condition is a requirement for the proof of correctness, irrelevant to later proofs.

orient:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E, R \cup \{s \to t\})}$	if $s > t$
deduce:	$\dfrac{(E, R)}{(E \cup \{s \approx t\}, R)}$	if $s \leftarrow_R u \to_R t$
delete:	$\dfrac{(E \cup \{s \approx s\}, R)}{(E, R)}$	
simplify:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R)}$	if $s \to_R u$
compose:	$\dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})}$	if $t \to_R u$
collapse:	$\dfrac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)}$	if $s \xrightarrow[R]{\sqsupset} v$

Figure 3. Standard Knuth-Bendix completion ($\mathcal{C}$)

A *deduction* of $\mathcal{C}$, written $(E, R) \vdash_{\mathcal{C}} (E', R')$, consists of finite sets of identities E, E' and rewriting systems R, R'. A execution γ of the system $\mathcal{C}$ is *valid* if it begins with the pair $(E_0, \emptyset)$ and is followed by a possibly infinite sequence of deductions

$$(E_0, \emptyset) \vdash_{\mathcal{C}} (E_1, R_1) \vdash_{\mathcal{C}} (E_2, R_2) \vdash_{\mathcal{C}} \cdots,$$

where E_0 is the finite set of identities provided as input by the user, and each deduction results from an application of exactly one of the inference rules of $\mathcal{C}$.

The *persistent identities* E_ω (*persistent rules* R_ω) are those that appear in some intermediate set of identities E_i (rules R_i) and remain in all future intermediate sets of identities E_j (rules R_j) for $j > i$,

$$E_\omega = \bigcup_{i \in \mathbb{N}} \bigcap_{j \geq i} E_j \qquad \textit{and} \qquad R_\omega = \bigcup_{i \in \mathbb{N}} \bigcap_{j \geq i} R_j.$$

The persistent sets are so defined to construct transfinite executions of completion procedures and to state and prove correctness properties

of $\mathcal{C}$. For consistency, we allow finite executions to be considered as infinite executions: a finite execution γ of length n can be extended to an infinite execution $\hat{\gamma}$ such that $(E_m, R_m) = (E_n, R_n)$ for all $m > n$. In the case of finite executions, the persistent sets are those found in the final deduction.

2.2.1. *Properties of the System $\mathcal{C}$*
An execution γ of $\mathcal{C}$ *succeeds* if $E_\omega = \emptyset$ and R_ω is a convergent rewriting system equivalent to E as described above. An execution γ of $\mathcal{C}$ *fails* if $E_\omega \neq \emptyset$. Failure of an execution occurs if an identity of some intermediate set of identities is never oriented or reduced to a trivial identity. This can occur if either the identity is not compatible with the given reduction order, or else simply that it is never the focus of a rule application in γ. An execution may also neither succeed nor fail if some critical pair is never considered. A *fair strategy* is one that produces executions that eventually consider each critical pair of R_ω,

$$\Gamma(R_\omega) \subseteq \bigcup_{i \in \mathbb{N}} E_i,$$

where $\Gamma(R_\omega)$ denotes the critical pairs of R_ω. In this work, we assume all executions are the result of an arbitrary, fair strategy.

The following theorem states key correctness properties of non-failing and fair executions.

Theorem 1 (Correctness of $\mathcal{C}$)**.** *Let $(E_0, \emptyset) \vdash_{\mathcal{C}} (E_1, R_1) \vdash_{\mathcal{C}} (E_2, R_2) \vdash_{\mathcal{C}} \cdots$ be a non-failing and fair execution of the completion procedure $\mathcal{C}$.*

1. *R_ω is equivalent to the set of input identities E_0.*
2. *R_ω is convergent.*
3. *If R_ω is finite, then the word problem for E_0 is decidable. Otherwise, the execution yields a semidecision procedure for $\approx_{E_0}$.*

The first part of this theorem states soundness of the procedure. The persistent rules are equivalent to the input identities, and since the execution is non-failing by assumption, no identities persist throughout the execution. The second part of the theorem simply states that the resulting rewriting system is confluent and terminating. Note though that the resulting rules may be infinite, so this does not on its own imply that a decision procedure for the input identities results. The third part of the theorem addresses this specifically: if R_ω is finite, then by virtue of our assumption of a non-failing fair execution, the execution is finite and the resulting system can be used to decide

the word problem. However, if the resulting system is infinite, then execution of the procedure yields a semidecision procedure.

The following corollary states that any strategy for applying the rules of system $\mathcal{C}$ that is fair is correct — either an equivalent convergent rewriting system is constructed, or some identities persist.

Corollary 2 (Completeness of $\mathcal{C}$)**.** *Every fair completion procedure that does not fail succeeds (*Bachmair, 1991*).*

It follows that if an execution does not succeed, then it cannot fail and remain fair. A fair execution that does not succeed fails because there is some identity that cannot be oriented at any point during the execution.

3. Related Work

Many classes of reduction orders have been developed for proving termination. Among them are the lexicographic path orders, the multiset path orders, a generalization of these called the recursive path orders with status, the Knuth-Bendix orders, and polynomial orders (Baader and Nipkow, 1998; Dershowitz and Plaisted, 2001; TeReSe, 2003). Common among these orders is that testing compatibility with a TRS is decidable and computationally tractable, but determining the existence of a compatible order is typically not.

For example, it is not decidable in general whether an RPO exists that is compatible with a particular theory (Comon and Treinen, 1997). Even if such an order exists, finding a suitable precedence and weight mapping for a finite set of identities is difficult for experienced users and automated tools alike — for LPO the problem is NP-complete (Nieuwenhuis, 1993). Furthermore, there are many theories which do admit some reduction order, but not an RPO or KBO. Currently, there is no automated theorem proving tool able to complete these theories.

This illustrates the basic limitation of completion procedures. An execution will fail if not provided a reduction order that is compatible with all of the rules of R_ω. Matters are complicated by the fact that implementations of completion allow the user to specify only a limited class of reduction orders: e.g., WALDMEISTER implements LPO and KBO, while CiME additionally implements polynomial orderings (Contejean et al., 2004; Löchner and Hillenbrand, 2002). This is largely out of necessity; in order to provide a suitable order, a user must be able to specify it succinctly. Nearly all tools allow the user to specify a precedence for generating an LPO or a precedence and weight mapping for a KBO.

3.1. Unfailing Completion

In some cases no compatible reduction order exists. For example, any theory which includes the commutativity rule $x * y \approx y * x$ must fail because it is not compatible with any reduction order — if there were, then an arbitrary identity $s \approx t$ could be rewritten to $t \approx s$ and again to $s \approx t$, contradicting strictness of the order.

A related approach to completion, called *unfailing completion* (Bachmair et al., 1989), attempts to cope with such unorientable identities. In an unfailing completion procedure, the user must provide a reduction order, but unorientable identities do not yield failure. The key observations are that *1)* reduction orders, though not total in general, are total on ground terms, and *2)* if a goal identity contains free variables, grounding it (by introducing fresh constants) results in an equisatisfiable problem. It follows that unorientable identities can be used as rewrite rules in normalization because their instantiations will be ground. Since the reduction order provided is total for ground terms, it will always yield an orientation (even if it does not orient the identity uninstantiated).

Because identities are allowed to remain throughout an execution of an unfailing completion procedure, the completions that are produced are not necessarily convergent. However, they are still useful because the completions are ground convergent—i.e., confluent and terminating when used to rewrite ground terms. Unfailing completion is a well understood variant of completion and is widely implemented. Besides its strong correctness properties, it is also efficient in practice.

The primary drawback of unfailing completion is its inability to reliably produce convergent completions. Although the ground convergent completions can be used to solve the word problem for arbitrary terms, the completion itself does not shed further light on the equational theory as a convergent completion does. Convergent completions are necessary for algebraic proof mining, in which fast algorithms are developed for analyzing proofs based on the normal forms of a completion (Stump and Tan, 2005; Wehrman and Stump, 2005). Ground convergent completions do not have easily analyzable normal forms, and hence cannot be used for algebraic proof mining.

There has been some previous interest in combining termination checkers with completion. Claude Marché and Xavier Urbain wrote about combining the *dependency pairs* termination method (Giesl et al., 2002) with Knuth-Bendix completion (Marché and Urbain, 1998). This work is limited to the use of the dependency pairs technique. In contrast, our algorithm can be combined with any termination-checking techniques, and not just those based on dependency pairs.

4. The Role of the Order in Knuth-Bendix Completion

Our goal is to develop a variant of Knuth-Bendix completion which solves the problems discussed in the previous section; namely that it is difficult to find and specify a compatible reduction order. We begin by examining the role of the input reduction order in the completion procedure. This is accomplished by introducing two generalizations of standard Knuth-Bendix completion that relax the requirement that the user provide a single compatible reduction order. These variants will turn out to be over-generalizations. However, by examining their properties and, more specifically, why they fail to be correct, these variants lead us indirectly to the solution we seek.

4.1. Unordered Completion

We first define a generalization of Knuth-Bendix completion that is identical to the system $\mathcal{C}$ but that requires no reduction order whatsoever. This modified system, which we call $\mathcal{C}_0$, is presented in Fig. 4. As an equational system, the only difference between $\mathcal{C}_0$ and $\mathcal{C}$ is in the definition of the orient rule. In system $\mathcal{C}_0$, the side condition only requires that an identity be oriented in such a way that it is a genuine rewrite rule (i.e., that no new variables are introduced in the right-hand side). It is obvious that this will prove to be an over-generalization, but we will see that it shares an important property with the system $\mathcal{C}$; one that any solution must also have.

orient:	$\dfrac{(E \cup \{s \dot{\approx} t\}, R)}{(E, R \cup \{s \to t\})}$	if $Vars(t) \subseteq Vars(s)$
deduce:	$\dfrac{(E, R)}{(E \cup \{s \approx t\}, R)}$	if $s \leftarrow_R u \rightarrow_R t$
delete:	$\dfrac{(E \cup \{s \approx s\}, R)}{(E, R)}$	
simplify:	$\dfrac{(E \cup \{s \dot{\approx} t\}, R)}{(E \cup \{u \dot{\approx} t\}, R)}$	if $s \rightarrow_R u$
compose:	$\dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})}$	if $t \rightarrow_R u$
collapse:	$\dfrac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)}$	if $s \xrightarrow{\sqsupset}_R v$

Figure 4. Generalized Knuth-Bendix completion ($\mathcal{C}_0$)

4.1.1. *Properties of the System $\mathcal{C}_0$*
The first important property to consider is soundness. A deduction $(E, R) \vdash_{\mathcal{C}_0} (E', R')$ is *sound* if the equational theory of $E \cup R$ is equal to the equational theory of $E' \cup R'$. It is easy to see that soundness of deductions implies that the equational theory of any intermediate tuple (E_i, R_i) in an execution is equivalent to that of the input theory E_0, since every execution starts with $(E_0, \emptyset)$.

Theorem 3 (Soundness of $\mathcal{C}_0$)**.** $(E_1, R_1) \vdash_{\mathcal{C}_0} (E_2, R_2)$ *implies* $\overset{*}{\leftrightarrow}_{E_1 \cup R_1} = \overset{*}{\leftrightarrow}_{E_2 \cup R_2}$.

Proof. For rules orient, deduce, delete, the claim follows trivially. For rule simplify, $E_1 = E \cup \{s \doteq t\}$, $E_2 = E \cup \{u \doteq t\}$, $R_1 = R = R_2$ and $s \to_R u$. We have $\overset{*}{\leftrightarrow}_{E_2 \cup R_2} \subseteq \overset{*}{\leftrightarrow}_{E_1 \cup R_1}$ because $u \leftarrow_{R_1} s \overset{*}{\leftrightarrow}_{E_1} t$, and $\overset{*}{\leftrightarrow}_{E_1 \cup R_1} \subseteq \overset{*}{\leftrightarrow}_{E_2 \cup R_2}$ because $s \to_{R_2} u \overset{*}{\leftrightarrow}_{E_2} t$. For rule compose, $E_1 = E = E_2$, $R_1 = R \cup \{s \to t\}$, $R_2 = R \cup \{s \to u\}$ and $t \to_R u$. We have $\overset{*}{\leftrightarrow}_{E_2 \cup R_2} \subseteq \overset{*}{\leftrightarrow}_{E_1 \cup R_1}$ because $s \to_{R_1} t \to_R u$, and $\overset{*}{\leftrightarrow}_{E_1 \cup R_1} \subseteq \overset{*}{\leftrightarrow}_{E_2 \cup R_2}$ because $s \to_{R_2} u \leftarrow_R t$. For rule collapse, $E_1 = E$, $E_2 = E \cup v = t$, $R_1 = R \cup \{s \to t\}$, $R_2 = R$ and $s \to_R v$. We have $\overset{*}{\leftrightarrow}_{E_2 \cup R_2} \subseteq \overset{*}{\leftrightarrow}_{E_1 \cup R_1}$ because $v \leftarrow_R s \to_{R_1} t$, and $\overset{*}{\leftrightarrow}_{E_1 \cup R_1} \subseteq \overset{*}{\leftrightarrow}_{E_2 \cup R_2}$ because $s \to_R v \overset{*}{\leftrightarrow}_{E_2} t$. □

In fact, the proof of soundness of $\mathcal{C}_0$ is exactly the same as a proof of soundness of $\mathcal{C}$; the side-condition on the orient rule is irrelevant. However, the system $\mathcal{C}_0$ is not correct in general because the rewriting systems are not necessarily equivalent to the initial set of identities—soundness is that the *union* of the intermediate rewriting system and identities preserve the equational theory. Consequently $\mathcal{C}_0$ does not generally yield a decision procedure for E_0, and can produce non-failing unsuccessful runs. Success requires that $E_\omega = \emptyset$ and that R_ω be convergent, and indeed it would be quite surprising if R_ω turned out even to be terminating when the only condition for adding new identities is that they be rewrite rules.

It is worth emphasizing that the system $\mathcal{C}_0$ is a strict generalization of the system $\mathcal{C}$—there is a simulation from system $\mathcal{C}$ to the system $\mathcal{C}_0$, written $\mathcal{C} \sqsubseteq \mathcal{C}_0$, meaning that any legal execution of $\mathcal{C}$ is a legal execution of $\mathcal{C}_0$. This is true because, for any execution γ of $\mathcal{C}$ we can construct an equivalent execution γ_0 of $\mathcal{C}_0$ (i.e., with the same deductions). As noted in Lem. 2.1, this implies that any property of $\mathcal{C}_0$, including soundness, also applies to the refined system $\mathcal{C}$.

The system $\mathcal{C}_0$ is not correct in general, but it is useful because it provides us a hint as to how to perform a completion procedure without requiring the user to provide an order (which, as we have discussed, is of considerable difficulty). We start by observing that in either system

$\mathcal{C}$ or $\mathcal{C}_0$ there are only a finite number of possible deductions that can take place at each step of the execution. Both systems have a finite number of rules which can result in different deductions based on the interpretation of the preconditions. But because the intermediate sets of identities and rewriting systems are finite, there are but a finite number of interpretations of the preconditions as well. This observation allows us to consider the set of possible executions as a labeled, finitely branching tree. The initial system $(E, \emptyset)$ is the root, and branches correspond to the states reachable by application of some rule.

This observation is useful because it shows how we can trade time for correctness. Since the tree of executions is finitely branching, if we explore the branches of the tree fairly — for example, with a breadth-first search — then for any execution of $\mathcal{C}$ with identities E_0 and order $>$ there exists some equivalent execution of $\mathcal{C}_0$. By spending exponentially more time searching a tree of executions in the system $\mathcal{C}_0$, the same deductions will eventually occur as in any execution of the system $\mathcal{C}$.

This approach has major drawbacks, however, primarily that there is no way of distinguishing successful branches from unsuccessful branches. In the standard system $\mathcal{C}$, each intermediate TRS is terminating by construction, and confluence can be tested by attempting to join all critical pairs. Therefore when performing standard completion, it is simple to determine whether or not a completion has been found. In the modified system $\mathcal{C}_0$, however, the intermediate term rewriting systems are not terminating in general, so it is not simple to judge whether an intermediate TRS is in fact a convergent completion. Since the convergent completion generated is the decision procedure, we cannot rely on $\mathcal{C}_0$ to ever produce a decision procedure.

Needless to say, this technique would also be exceedingly slow in practice. Indeed there are more elegant and efficient procedures for proof search, such as paramodulation (Nieuwenhuis and Rubio, 2001) and unfailing completion (Bachmair et al., 1989). However, these techniques provide no means for obtaining a convergent completion. In the next section, we consider refinements of $\mathcal{C}_0$ which will solve the problem of distinguishability of successful branches and increase performance.

4.2. Many-ordered Completion

Since we have seen that completely eliding the reduction order in Knuth-Bendix completion breaks correctness, we take a different tack in making it easier to provide orders. Instead of requiring that the user provide a single reduction order compatible with all rules in intermediate rewriting systems, the user provides a set of reduction orders $\mathcal{O}$. The rules of this system, which we call $\mathcal{C}_1$, are presented in Fig. 5. The side-

condition on the orient requires that there is some reduction order $>\in \mathcal{O}$ for which all the rules in $R \cup \{s \to t\}$ are compatible. The idea here is that each intermediate rewriting system is shown terminating by some method, even if it is a different method at each step. The set $\mathcal{O}$ could be, for example, the set of lexicographic path orders over the signature under consideration. More generally, we are free here to consider the set of all reduction orders as $\mathcal{O}$. Also, if $\mathcal{O}$ is the set which contains a single reduction order, then $\mathcal{C}_1$ is equivalent to $\mathcal{C}$. Consequently, we have that there is a simulation from $\mathcal{C}$ to $\mathcal{C}_1$ by choosing $\mathcal{O}$ as a singleton and from $\mathcal{C}_1$ to $\mathcal{C}_0$ by choosing $\mathcal{O}$ as the set of all orders: $\mathcal{C} \sqsubseteq C_1 \sqsubseteq \mathcal{C}_0$.

orient:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E, R \cup \{s \to t\})}$	if $\exists >\in \mathcal{O}.\ \forall l \to r \in R \cup \{s \to t\}.\ l > r$
deduce:	$\dfrac{(E, R)}{(E \cup \{s \approx t\}, R)}$	if $s \leftarrow_R u \to_R t$
delete:	$\dfrac{(E \cup \{s \approx s\}, R)}{(E, R)}$	
simplify:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R)}$	if $s \to_R u$
compose:	$\dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})}$	if $t \to_R u$
collapse:	$\dfrac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)}$	if $s \overset{\sqsupset}{\to}_R v$

Figure 5. Many-ordered Knuth-Bendix completion ($\mathcal{C}_1$)

The intuition behind the system $\mathcal{C}_1$ is that it may be considerably easier to provide a set of orders, one of which is compatible with each intermediate system, than to specify a single order which is compatible with all intermediate systems. Next we investigate whether or not this restriction provides properties we desire.

4.2.1. *Properties of the System* $\mathcal{C}_1$

Since $\mathcal{C}_1 \sqsubseteq C_0$, soundness of $\mathcal{C}_1$ follows immediately from Thm. 3.

Like $\mathcal{C}_0$, the system $\mathcal{C}_1$ is finitely branching. Note that even if $\mathcal{O}$ has infinite cardinality, the rule orient can only be applied to an identity in two ways ($s \to t$ or $t \to s$).

Correctness of this system has been studied previously. In particular, some interactive completion tools, for example the Rewrite Rule Laboratory (RRL) (Kapur and Zhang, 1995) for many years implemented

a variant of this system. In RRL, the user specifies an initial order $>_1$, but is allowed to provide a different order $>_2$ in the middle of the procedure if $>_2$ is compatible with all the rules in the current intermediate rewriting system. Surprisingly, it was unknown for quite a while as to whether or not this variation was correct; Dershowitz listed it in the official Rewriting Techniques and Applications an open problem list in 1991 (Dershowitz et al., 1991; Dershowitz and Treinen,). In 1994, however, Andrea Sattler-Klein closed it with complicated examples that demonstrated incorrectness (Sattler-Klein, 1994). In particular, it was shown that there exist theories for which R_ω was not only non-confluent for infinite executions of $\mathcal{C}_1$, but also non-terminating for infinite executions and non-confluent even for finite executions. However, Sattler-Klein also proved that finite executions of the system $\mathcal{C}_1 - \{\mathsf{collapse}, \mathsf{compose}\}$ are correct; if the system terminates after a finite number of steps without failure, then the resulting TRS is convergent and equivalent to the input theory. This is easy to see, for the system has the property that the intermediate rewriting systems form an increasing chain: $R_1 \subseteq R_2 \subseteq \ldots \subseteq R_n$. First note that since no rule is ever removed, each rule is persistent and $R_\omega = R_n$. Since by definition, there is some reduction order compatible with the last rewriting system, then R_ω is terminating. Since the execution is fair, all nontrivial critical pairs of R_ω have been added, and so it is also confluent. Still, infinite executions of the restricted system can result in a non-confluent R_ω — an important caveat, because it implies that even the system $\mathcal{C}_1 - \{\mathsf{collapse}, \mathsf{compose}\}$ can not be used as a semidecision procedure for some theories.

At first glance, it seems that this system also fails to resolve the problems we presented while remaining usable as a theorem prover. However, because the system $\mathcal{C}_1 - \{\mathsf{collapse}, \mathsf{compose}\}$ is correct for finite execution and is also finitely branching, this allows the following scheme: explore all branches of the execution tree, with branch points at applications of the orient rule. If any path from the root leads to a finite, non-failing system, then because all intermediate rewriting systems are convergent the resulting system is a decision procedure for the input theory.

There are two drawbacks to such a scheme, however. First, without the rules collapse and compose, the rewriting systems produced (the convergent completions) will be large and unwieldy, perhaps with many redundant rules. This is in contrast to the convergent completions produced by the standard procedure in which redundant rules are typically eliminated with the aforementioned rules. While such a completion could still be used as a decision procedure, they do not yield an object of study that reveals as much about the equational theory, mitigating its

usefulness for algebraic proof mining. Second, it is not clear that such a scheme would be useful as a semidecision procedure. Indeed, the basis of the scheme, $\mathcal{C}_1 - \{\mathsf{collapse}, \mathsf{compose}\}$, is not a correct semidecision procedure. In the next section, we address these issues.

5. Knuth-Bendix Completion with Modern Termination Checking

We now present a modification of the standard KB completion procedure in which no reduction order is provided as input, only a finite set of identities. Lacking any specific reduction order to guide the search, we preserve convergence of each intermediate rewriting system R_i by ensuring that some reduction order $\succ_i$ compatible with R_i exists. The orders $\succ_i$ are constructed using terminating rewriting systems C_i, specifically as the transitive closure of the reduction relation on C_i, written $\xrightarrow{+}_{C_i}$. This relation is a reduction order exactly when the system C_i is terminating. While in the standard system $\mathcal{C}$ a rule $s \to t$ is added by orient to R_i only if $s > t$ with the user-specified reduction order, in the modified system the rule is added only if the addition of $s \to t$ to C_i preserves termination. Of course, deciding termination is not possible in general. In Sect. 6, we discuss how this test is accomplished in practice.

Figure 6 provides the inference rules a modification of the standard completion procedure, which we refer to as system $\mathcal{A}$. A deduction of $\mathcal{A}$, written $(E, R, C) \vdash_{\mathcal{A}} (E', R', C')$, consists of identities E, E' and rewriting systems R, R' as in standard completion, and finite *constraint* rewriting systems C, C' unique to $\mathcal{A}$. An execution α of the system $\mathcal{A}$ is valid if it begins with the triple $(E_0, \emptyset, \emptyset)$ and is followed by a sequence of deductions

$$(E_0, \emptyset, \emptyset) \vdash_{\mathcal{A}} (E_1, R_1, C_1) \vdash_{\mathcal{A}} (E_2, R_2, C_2) \vdash_{\mathcal{A}} \cdots,$$

with E_0 the set of input identities and where each deduction results from an application of one inference rule from $\mathcal{A}$. An execution α of $\mathcal{A}$ is *equivalent* to an execution γ of $\mathcal{C}$ when the intermediate equations and rewriting systems are the same at each step. A execution α of system $\mathcal{A}$ succeeds when $E_{|\alpha|} = \emptyset$ and $R_{|\alpha|}$ is a convergent rewriting system equivalent to E.

The rules deduce, delete, simplify, compose and collapse of $\mathcal{A}$ are identical to those of $\mathcal{C}$, except for the presence of the constraint system C which is carried unmodified from antecedent to consequent. The critical difference between $\mathcal{A}$ and $\mathcal{C}$ is in the definition of the orient rule. In the standard system $\mathcal{C}$, an identity $s \doteq t$ of E is added to R as rule $s \to t$ only when $s > t$ for the given reduction order. In

orient:	$\dfrac{(E \cup \{s \dot{\approx} t\}, R, C)}{(E, R \cup \{s \to t\}, C \cup \{s \to t\})}$	if $C \cup \{s \to t\}$ terminates
deduce:	$\dfrac{(E, R, C)}{(E \cup \{s \approx t\}, R, C)}$	if $s \leftarrow_R u \to_R t$
delete:	$\dfrac{(E \cup \{s \approx s\}, R, C)}{(E, R, C)}$	
simplify:	$\dfrac{(E \cup \{s \dot{\approx} t\}, R, C)}{(E \cup \{u \dot{\approx} t\}, R, C)}$	if $s \to_R u$
compose:	$\dfrac{(E, R \cup \{s \to t\}, C)}{(E, R \cup \{s \to u\}, C)}$	if $t \to_R u$
collapse:	$\dfrac{(E, R \cup \{s \to t\}, C)}{(E \cup \{v \approx t\}, R, C)}$	if $s \xrightarrow{\sqsupset}_R v$

Figure 6. Modified Knuth-Bendix Completion ($\mathcal{A}$)

the modified system $\mathcal{A}$, we add the rule $s \to t$ to R only when the augmented constraint system $C \cup \{s \to t\}$ is terminating. The system $\mathcal{A}$ accepts as input only the finite set of identities E; no reduction order is explicitly provided.

In Sect. 5.1, we discuss the properties of the system $\mathcal{A}$. We prove correctness in two parts: first in Sect. 5.1.1 for finite executions; second in Sect. 5.1.2 for all executions, including infinite ones.

5.1. Properties of the System $\mathcal{A}$

We will now show that the system $\mathcal{A}$ is correct in the sense of Thm. 1; i.e., that it is sound, that the resulting rewriting system is convergent, and that finite, non-failing fair executions result in a decision procedure for the word problem of the input identities, and infinite non-failing fair executions a semidecision procedure. Soundness is easy to prove because $\mathcal{A}$ simulates the unordered system $\mathcal{C}_0$ which is sound by Thm. 3. The other two properties require more elaborate arguments. Below we present two separate arguments for the correctness of $\mathcal{A}$.

First we prove in Sect. 5.1.1 that the properties hold *for finite executions only* using a simulation argument, in particular that for every finite execution of $\mathcal{A}$ there exists an equivalent execution of the standard system $\mathcal{C}$. Because every execution of $\mathcal{A}$ is mirrored by an execution of $\mathcal{C}$, and $\mathcal{C}$ is correct, then $\mathcal{A}$ is also correct. This sort of simple argument is exactly the reason Bachmair went to the trouble to formulate the standard completion procedure as an inference system

and prove it correct: most variations can then be proved correct by showing that they simulate the standard system. Unfortunately, as we will see, our modified system is too much of a departure to rely on simulation alone to prove it correct for *infinite* executions.

The second proof of correctness in Sect. 5.1.2, which works for finite or infinite executions, also relies internally on a simulation argument, but is more complicated. In this proof, we perform a simulation opposite to the one used in the proof of correctness for finite executions: we show that for every execution of the original system $\mathcal{C}$ there exists an equivalent execution of the modified system $\mathcal{A}$. We then show that such an execution is actually constructed, so that for any set of identities that can be completed using the system $\mathcal{C}$ an equivalent execution of $\mathcal{A}$ will be made. This proof technique does not rely on the finiteness of executions, however, and so makes the first proof technically superfluous. We include the proof of finite correctness here for instructive reasons only, as it illustrates the limitations of completion correctness arguments via simulation.

5.1.1. *Finite Correctness*

It is easy to see that termination of C_i is invariant over i; no rules are ever simplified or removed, and a rule is only added if it preserves the termination property. The following lemma states that for each intermediate rewriting system in an execution α of $\mathcal{A}$, there exists a reduction order compatible with it. We write $\succ_i$ for $\xrightarrow{+}_{C_i}$, the transitive closure of the reduction relation for C_i.

Lemma 1. *Let α be a finite execution of the system $\mathcal{A}$. Then for all $l \to r \in R_{|\alpha|}$, $l \succ_{|\alpha|} r$.*

Proof. By induction on $|\alpha|$. The claim is vacuously true when $|\alpha| = 0$. For $|\alpha| = k$ with $\alpha = \alpha' \vdash_{\mathcal{A}} (E_k, R_k, C_k)$, we assume $l \succ_{k-1} r$ for all $l \to r \in R_{k-1}$. If the final state of α is due to an instance of the rule deduce, delete, or simplify, then $R_k = R_{k-1}$, $C_k = C_{k-1}$ and $l \succ_k r$ for all $l \to r \in R_k$ by IH. If the final state of α is due to an instance of the rule collapse, then $R_k \subset R_{k-1}$, $C_k = C_{k-1}$ and again $l \succ_k r$ by IH. If the final state of α is due to an instance of the rule compose, then assume for contradiction that for some $l \to r \in R_k = R \cup \{s \to u\}$ we have $l \not\succ_k r$. It cannot be that $l \to r \in R$, because $C_k = C_{k-1}$ and by IH $l \succ_{k-1} r$, so $l \succ_k r$. So $l = s$ and $r = u$, and $s \not\succ_{k-1} u$. But $s \succ_{k-1} t \succ_{k-1} u$, so $s \succ_k u$, a contradiction. Finally, assume the final state of α is due to an instance of the rule orient. Then $R_k = R_{k-1} \cup \{s \to t\}$ and $C_k = C_{k-1} \cup \{s \to t\}$. For all $l \to r \in R_{k-1}$, we have $l \succ_{k-1} r$ and so $l \succ_k r$. Otherwise, $l = s$ and $r = t$, and by definition of the transition, $s \succ_k t$. □

In standard Knuth-Bendix completion, a single reduction order is provided by the user. Therefore, to show correctness of our modified completion procedure, it is necessary to exhibit one reduction order which is compatible with all intermediate rewriting systems. For finite executions, we can use the reduction relation of the final constraint rewriting system, augmented by each application of the rule orient. The constraint systems form an increasing chain, and so the induced reduction order is refined at each step and compatible with all prior orientations.

Lemma 2. *Let α be a finite execution of the system $\mathcal{A}$. Then for all $i \leq |\alpha|$ and $l \to r \in R_i$, $l \succ_{|\alpha|} r$.*

Proof. By induction on $|\alpha|$. The claim is vacuously true for $|\alpha| = 0$. For $\alpha = \alpha' \vdash_{\mathcal{A}} (E_k, R_k, C_k)$, we assume $l \succ_{k-1} r$ for all $l \to r \in \cup_{i<k} R_i$. Since $C_{k-1} \subseteq C_k$, then by definition of the $\succ$ orders, $s \succ_{k-1} t$ implies $s \succ_k t$. So for any $i < k$ and $l \to r$ in R_i, $l \succ_k r$ by IH. Finally, for $l \to r \in R_k$, we have $l \succ_k r$ by Lem. 1. □

The preceding lemma shows that the modified completion procedure can be considered to use only a single reduction order throughout any execution. This is important because completion is not generally correct when reduction orders are changed during execution, even if each is compatible with the immediate intermediate rewrite system (Sattler-Klein, 1994). A single reduction order also allows the convenience of proving partial correctness by simulation of a standard completion procedure.

Theorem 4 ($\mathcal{A} \sqsubseteq \mathcal{C}$)**.** *Let α be a finite execution of the system $\mathcal{A}$. Then there exists an equivalent execution γ of $\mathcal{C}$ using reduction order $\succ_{|\alpha|}$.*

Proof. By induction on $|\alpha|$. We construct an execution γ of $\mathcal{C}$ that uses $\succ_{|\alpha|}$ as the given reduction order. The beginning execution is $\alpha = (E_0, \emptyset, \emptyset)$, which translates trivially to $\gamma = (E_0, \emptyset)$. Otherwise, $\alpha = \alpha' \vdash_{\mathcal{A}} (E_k, R_k, C_k)$ and let $\gamma = \gamma' \vdash_{\mathcal{C}} (E_k, R_k)$. By the IH, γ' satisfies the claim for α'. We show γ is a valid execution given the inference rules of $\mathcal{C}$. If the final deduction of α results from any of the rules of $\mathcal{A}$ other than orient, then the deduction is valid in γ by definition of the corresponding rule of $\mathcal{C}$. Otherwise, orient produces the final deduction with $R_k = R_{k-1} \cup \{s \to t\}$. Lemma 2 shows that $s \succ_{|\alpha|} t$ for all $s \to t \in R_k$, so orient is a valid deduction in γ. □

Theorem 5 (Finite Correctness of $\mathcal{A}$)**.** *Let $\alpha := (E_0, \emptyset, \emptyset) \vdash_{\mathcal{A}} (E_1, R_1, C_1) \vdash_{\mathcal{A}} (E_2, R_2, C_2) \vdash_{\mathcal{A}} \cdots$ be a finite, non-failing, fair execution of the system $\mathcal{A}$.*

1. *$R_{|\alpha|}$ is equivalent to the set of input identities E_0;*
2. *$R_{|\alpha|}$ is convergent; and*
3. *The word problem for E_0 is decidable.*

Proof. The first part of the theorem follows from Thm. 3, the soundness of the unordered system $\mathcal{C}_0$. The second and third parts follow from Thm. 4, an equivalent execution of $\mathcal{C}$ is constructable from α using the reduction order $\succ_{|\alpha|}$. □

We can now see why the preceding simulation argument fails for infinite executions: we must be able to construct the reduction order $\succ_{|\alpha|}$. We have no trouble doing this for finite orders. At each step of the execution, $\succ_{|\alpha|}$ is a reduction order, though it is refined after each application of an ORIENT rule. For finite executions, this incremental refinement provides no trouble; at each step, the order is a reduction order, and remains so until the procedure halts. The problem is that for infinite executions, this unending refinement of the order can prevent it from being well-founded, and hence from being a reduction order.

The particular problem is that termination of the infinite union of the intermediate constraint systems does not follow from termination of the individual systems. This is because in general the union of an infinite number of finite, terminating rewriting systems is not itself terminating. For example, consider the family of rewriting systems $R_j = \cup_{0 \leq i \leq j} \{fg^i f \rightarrow fg^{i+1}f\}$. For any $k \in \mathbb{N}$ it is easy to see that $\cup_{0 \leq j \leq k} R_j$ is terminating. But it is not the case that $\cup_{j \in \mathbb{N}} R_j$ is terminating, for it contains the infinite derivation $ff \rightarrow fgf \rightarrow fggf \rightarrow \cdots$.

Instead, it must be shown in a proof of correctness that includes infinite executions that *some* successful branch of execution always exists, and that it will always be found in the search for a completion. In the next subsection, we make such an argument.

5.1.2. *Total Correctness*

We begin by showing a sort of completeness for our procedure with respect to standard Knuth-Bendix completion. Namely, for any successful execution of the standard completion procedure $\mathcal{C}$ there exists a corresponding execution of the modified procedure $\mathcal{A}$ with the same deductions. This will be used in a later argument, and also shows that our method can, with the proper nondeterministic choices, construct decision procedures for *any theory* that is decidable by the standard method.

Theorem 6 ($\mathcal{C} \sqsubseteq \mathcal{A}$)**.** *For any valid execution γ of $\mathcal{C}$ with reduction order $>$, there exists an equivalent valid execution α of $\mathcal{A}$. Furthermore, $C_{|\alpha|} \subseteq >$ and $R_{|\alpha|} \subseteq >$*

Proof. By transfinite induction on $|\gamma|$.

- In the base case, $\gamma_0 = (E_0, \emptyset)$ by validity, and so $\alpha_0 = (E_0, \emptyset, \emptyset)$. These executions are clearly equivalent, α is valid, and also $\emptyset \subseteq >$.

- In the step case, γ has length $k+1$, $\gamma_k = (E_k, R_k)$ and extends γ' of length k. By IH there exists an execution α' that satisfies the claim for γ', including that $C_{k-1} \subseteq >$. Let $C_k = C_{k-1}$ if the final deduction is the result of any rule except orient, and $C_k = C_{k-1} \cup \{s \to t\}$ otherwise, with $\{s \to t\} = R_k - R_{k-1}$. Now let α extend α' such that $\alpha_k = (E_k, R_k, C_k)$. This is clearly equivalent to γ, and we claim this is a valid execution of $\mathcal{A}$. This is trivial for rules other than orient, since their side conditions do not mention the constraint systems. Otherwise, $s > t$ and $\xrightarrow{+}_{C_{k-1}} \subseteq >$ which implies $C_k \subseteq >$.

- In the limit case, $|\gamma|$ is a limit ordinal, and γ is a valid extension of the executions γ_λ for all $\lambda < |\gamma|$. By IH there exist valid executions α' of $\mathcal{A}$ for all $\lambda < |\gamma|$, and also that $R_\lambda \subseteq >$ and $C_\lambda \subseteq >$. Let α be an extension of all α' as above such that $\alpha = (\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa\leq\iota<|\gamma|} E_\iota, \bigcup_{\kappa<|\gamma|} \bigcap_{\kappa\leq\iota<|\gamma|} R_\iota, \bigcup_{\iota<|\gamma|} C_\iota)$. By definition of the persistent sets, $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa\leq\iota<|\gamma|} E_\iota := E_\omega$ and $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa\leq\iota<|\gamma|} R_\iota := R_\omega$, so α is equivalent to γ. For $\iota < |\gamma|$ we have $R_\iota \subseteq >$ and $C_\iota \subseteq$ by IH. It follows easily that that $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa\leq\iota<|\gamma|} R_\iota \subseteq >$ and $\bigcup_{\iota<|\gamma|} C_\iota \subseteq >$.

□

The above theorem demonstrates the existence of a successful execution of $\mathcal{A}$ for every successful execution of $\mathcal{C}$, including infinite ones. It also implies that the rewriting systems $R_{|\alpha|}$ and constraint systems $C_{|\alpha|}$ are terminating because their rules are compatible with the reduction order $>$. But note that that the rule orient in $\mathcal{A}$ can orient an equation $s \approx t$ in either direction when both $C \cup \{s \to t\}$ and $C \cup \{t \to s\}$ are terminating systems. Consequently, an execution of $\mathcal{A}$ as defined above will fail if a poor decision is made during orientation. The ability to construct a successful execution relies on a non-deterministic orientation choice. Deterministically, an execution of $\mathcal{A}$ becomes a binary tree in which each node is an instance of the rule orient. In practice, we must *search* for a successful execution. We ensure discovery of such an execution (corresponding to a path from the root $(E_0, \emptyset)$) by advancing each of the individual executions in a fair manner.

We now prove the main theorem, correctness of the system $\mathcal{A}$, by leveraging the fact that $\mathcal{C} \sqsubseteq \mathcal{A}$.

Theorem 7 (Correctness of $\mathcal{A}$)**.** *If there exists a non-failing fair execution $\gamma := (E_0, \emptyset) \vdash_{\mathcal{C}} (E_1, R_1) \vdash_{\mathcal{C}} (E_2, R_2) \vdash_{\mathcal{C}} \cdots$ of the system $\mathcal{C}$, then there is a deterministically constructable execution α of the system $\mathcal{A}$ with the following properties.*

1. *$R_{|\alpha|}$ is equivalent to the set of input identities E_0;*
2. *$R_{|\alpha|}$ is convergent; and*
3. *If R_ω is finite, then the word problem for E_0 is decidable. Otherwise, the execution yields a semidecision procedure for $\approx_{E_0}$.*

Proof. The first part follows from the soundness of $\mathcal{C}_0$ in Thm. 3 and that $\mathcal{A} \sqsubseteq \mathcal{C}$. For the other parts, consider that by Thm. 6 for every fair, non-failing execution γ of the system $\mathcal{C}$ there exists an equivalent execution α of the system $\mathcal{A}$. Because the identities and rewriting systems of the execution α are the same as those in γ, it follows that α satisfies that second two properties of the theorem.

Thm. 6 is nonconstructive, however, in that it relies on the ability to make a nondeterministic choice in case a particular identity can be oriented in both directions. If we actually had an execution γ of the system $\mathcal{C}$ on hand, we could choose the correct orientation based on that. We do not, however, and so we simply try both orientations. Because each orientation yields only two choices, if we explore all possible orientations then we can think of the executions as a finitely branching tree. By exploring the branches of this tree fairly, we will eventually encounter the branch that contains the system R_ω of γ because, as Thm. 6 demonstrates, some branch of the tree is exactly α, and α is equivalent to γ. □

It is interesting to note that termination of the intermediate rewrite systems is irrelevant to the proof except that without termination it would not be possible to explore the branches fairly. Because each intermediate rewriting system is terminating as in the standard completion procedure, we can explore the branches of the tree by visiting each execution individually, executing the completion as though we were only performing a single one. If the individual rewriting systems were not terminating, we might encounter an infinite reduction when computing normal forms in critical pairs calculations. Then again, if we could correctly guess which branch contained the execution α equivalent to γ, then we would not have to be concerned about termination, as Thm. 6 proves that the rewriting systems are in fact terminating.

5.1.3. *Practical Correctness*
Given the definition of the system $\mathcal{A}$ and proof of its correctness, one might rightly ask whether or not we have assumed too much. Indeed, in the definition of the orient rule, we assume the ability to determine whether or not an arbitrary rewriting system is encompassed by some reduction order (i.e., is terminating). But as we admitted in Sect. 2, this question is undecidable in general. What then does this new variant of Knuth-Bendix completion practically offer, when no such termination-checking oracle exists?

Consider again Thm. 6, and note that an invariant of the execution α of system $\mathcal{A}$ constructed is that the intermediate rewriting and constraint systems are always included in the input reduction order $>$. Also consider that while termination checking is undecidable in general, there are many decidable classes, and some that are even efficiently decidable. It follows then that as long as the given reduction order $>$ is in some decidable class, then Thm. 6 is provable and the equivalent execution α is constructable without an oracle. This means that the system $\mathcal{A}$ can reliably construct convergent completions for all those theories in which termination is practically checkable.

The same statement holds for Knuth-Bendix completion though. Whenever there is a theory which is provably compatible with some reduction order, a convergent completion can be constructed by Thm. 2. The difference however is not only that these proofs are hard to construct, but that from the user's perspective, it is difficult to guess what should even be proved (i.e., in what reduction order is a theory contained). In the modified algorithm, this difficulty is swept aside by searching for an order and letting a tool, a termination checker, build proofs as the search proceeds. It follows that if, in an implementation of the new procedure, a termination checker is used that can decide termination using some orders $\mathcal{O}$, then that implementation of the new procedure can decide all theories that admit an order in $\mathcal{O}$. A major contribution of the new algorithm is that, by offloading the work of proving termination to a separate tool, the theoretical capability of completion procedures are more readily achieved in reality than with the standard technique.

6. Implementing Completion with a Termination Checker

Our modified Knuth-Bendix completion procedure is implemented in a 3500-line Ocaml program called Slothrop. Theories are read from files in the standard format of the TPTP (Thousands of Problems for Theorem Provers) project (Suttner and Sutcliffe, 1996). The source grew

from an existing implementation of the standard completion procedure by Franz Baader and Tobias Nipkow (Baader and Nipkow, 1998) and makes use of data structures programmed by Jean-Cristophe Filliâtre (Filliâtre, 2003). The strategy for executing the completion procedure in SLOTHROP originated in an algorithm from 1981 by Girard Huet (Huet, 1981). Ten years later, Bachmair used the inference system $\mathcal{C}$ (Bachmair, 1991) to prove Huet's algorithm correct.

6.1. HUET'S COMPLETION STRATEGY

Huet's algorithm proceeds iteratively by choosing identities from the current set E_i, reducing them to normal forms using the current rewriting system R_i, orienting them (if possible), and finally adding the oriented rules to the next set of rewrite rules. A rewrite rule is then chosen from this new set and used to calculate critical pairs against the other rewrite rules. These critical pairs are added to the set of identities, and the chosen rule is *marked*, meaning that it will not again be chosen to generate critical pairs. If during execution all rules become marked and all identities are oriented, then the procedure has succeeded and the current rewrite rules constitute a convergent completion of the input identities.

SLOTHROP implements Huet's algorithm essentially as specified, the major difference being with the orientation step. Huet's algorithm as originally specified assumes the presence of a reduction order $>$ used for orienting identities. As each identity $s \approx t$ is considered, Huet's algorithm checks that $s > t$ and, if so, adds the rule $s \to t$ to the next intermediate set of rewrite rules. If $s \not> t$ but $t > s$, then the rule $t \to s$ is added. If otherwise $s \not> t$ and $t \not> s$, then the execution fails. In the new completion procedure, however, there is no given reduction order, so we turn to a termination checker.

6.2. INTEGRATION WITH A TERMINATION CHECKER

It is well known that determining termination of term rewriting systems is undecidable in general. However, modern termination-checking tools, such as APROVE (Giesl et al., 2004), CiME (Contejean et al., 2004) and TPA (Koprowski, 2006), succeed in proving many systems terminating or nonterminating with great success. Our implementation takes advantage of this success and uses either APROVE or TPA as an oracle to answer queries about the termination of constraint rewriting systems in each orientation step. If the termination checker fails to prove a system terminating or nonterminating, we treat it as a nonterminating system. However, the array of techniques used to show termination includes recursive path orders among many others,

so there is little difficulty recognizing the termination of systems compatible with such an order. Furthermore, since these tools are able to prove termination of systems that are not compatible with any recursive path order, SLOTHROP can find convergent completions of theories other modern completion tools (e.g., WALDMEISTER (Löchner and Hillenbrand, 2002)) cannot. One example of such a theory is given in Sect. 7.

Integrating a separate termination checker also provides separation-of-concerns benefits for theorem proving. As the power and speed of the tools improve, so does SLOTHROP. This also provides the opportunity to leverage other termination checkers with different properties (e.g., one which is faster but less powerful might be useful for simple theories).

6.3. HEURISTIC SEARCH

Another important aspect of our implementation is the manner in which different branches of executions are explored. When the termination checker determines that some identity $s \approx t$ can be oriented either way, both branches are explored. Implementation of this exploration is critical to performance. The binary tree of executions is potentially infinite, and branches whenever orders exist that are compatible with both orientations of an identity.

A breadth-first search of the branches is sufficient for completeness; if there is some successful execution corresponding to a branch on the tree, it will eventually be expanded. In practice, however, this strategy spends too much time in uninteresting areas of the search space, and prevents SLOTHROP from finding completions for any but the most modest theories in a reasonable amount of time. A more effective strategy is a best-first search in which the next execution to advance is chosen based on a cost function defined by

$$\text{cost}(E, R, C) := \text{size}(C) + \text{size}(E) + \text{size}(\Gamma(R)),$$

where $\Gamma(R)$ denotes the set of nontrivial critical pairs of R. With this strategy, $\text{size}(C)$ can be thought of as the cost to reach the current intermediate step in the execution and $\text{size}(E)+\text{size}(\Gamma(R))$ as a heuristic estimate for the cost to find a convergent completion.

While this heuristic has been successful in completing systems of modest size (described in Sect. 7), performance is an issue for larger systems. Better heuristics would greatly help the algorithm's practical effectiveness. Although accurately identifying diverging completions is itself a fundamentally hard problem (Sattler-Klein, 1991), we believe better intuition about what makes a partial completion similar to a final completion could yield great progress.

7. Results and Performance

SLOTHROP is capable of completing a variety of theories fully automatically in a modest amount of time. For example, the standard ten-rule completion of the group axioms is discovered in two seconds on a desktop PC with a 3GHz processor and 1GB of memory. On the way to this completion, it encounters 56 orientations, roughly half of which are not trivially terminating or nonterminating and must be verified with the termination checker. SLOTHROP automatically completes the theory of groups plus a single endomorphism (GE_1, shown in Fig. 7) in seven seconds, requiring about a hundred calls to the termination checker. The completion discovered is shown in Fig. 8. The theory GE_2 which adds another endomorphism symbol is completed in 35 seconds with over 350 calls to the termination checker.

$1 * x \approx x$	$(x * y) * z \approx x * (y * z)$
$x^{-1} * x \approx 1$	$h(x * y) \approx h(x) * h(y)$

Figure 7. The theory of one group endomorphism (GE_1)

$x * 1 \to x$	$x * (y * z) \to (x * y) * z$
$1 * x \to x$	$(x * y)^{-1} \to x^{-1} * y^{-1}$
$x * x^{-1} \to 1$	$(x * y) * y^{-1} \to x$
$x^{-1} * x \to 1$	$(x * y^{-1}) * y \to x$
$1^{-1} \to 1$	$h(x)^{-1} \to h(x^{-1})$
$h(1) \to 1$	$h(x) * h(y) \to h(x * y)$
$(x^{-1})^{-1} \to x$	$(x * h(y)) * h(z) \to x * h(y * z)$

Figure 8. Convergent completion of GE_1

The majority of SLOTHROP's running time is spent waiting for calls to the termination checker. Although we have encountered some examples of rewriting systems which APROVE or TPA can show terminating after a relatively long amount of time, in practice we have found that it is uncommon for such difficult systems to appear on the branch of a successful execution. Most calls to the termination checker that occur on successful branches return in under 2 seconds. Completeness of SLOTHROP can be exchanged for performance enhancements by calling

the termination checker with a short timeout. The above completions were obtained with a 5-second timeout.

7.1. New Completions

Since Slothrop is not restricted to a given reduction order, it can also search for multiple completions of a given theory. In the implementation, once a completion is found in some corner of the space of orientations, the search continues, looking for more in other parts of the space that have not already been proved unworthy of consideration. This has turned out to be an interesting side effect of the algorithm. For example, it finds two completions of the basic group axioms corresponding to both orientations of the associativity rule. It also finds four completions of GE_1 corresponding to the orientations of the associativity endomorphism rules. It also discovers a number of other larger completions of the same theory in which endomorphism rules are oriented differently depending on the context.

Shown in Fig. 9 below is a family of convergent completions for the structure GE_1 discovered by Slothrop. This family is parametrized by the natural number k that controls the size of the system and the shape of the rules in a predictable way. Slothrop discovered the first four completions in this family, from which the pattern was gleaned. It remains to be proved that all members of the family shown in Fig. 9 are completions of GE_1.

$x * 1 \to x$	$x * x^{-1} \to 1$
$1 * x \to x$	$x^{-1} * x \to 1$
$h(1) \to 1$	$1^{-1} \to 1$
$(x^{-1})^{-1} \to x$	$h(x)^{-1} \to h(x^{-1})$
$(x * y)^{-1} \to y^{-1} * x^{-1}$	
$(x * h^{k+1}(y)) * h^{k+1}(z) \to x * h^{k+1}(y * z)$	
$(x * h^i(y)) * h^i(y^{-1}) \to x$	for all $0 \leq i \leq k$
$(x * h^i(y^{-1})) * h^i(y) \to x$	for all $0 \leq i \leq k$
$x * (h^i(y * z)) \to x * (h^i(y) * h^i(z)))$	for all $0 \leq i \leq k$

Figure 9. Convergent k-completions of GE_1

Additionally, a convergent completion can be obtained by Slothrop for the theory of two commuting group endomorphisms (CGE_2), shown in Fig. 10. The convergent completion is shown in Fig. 11. The reader

may verify that no RPO or KBO is compatible with the theory (in particular, the final commutativity rule). A convergent completion was recently obtained by hand (Stump and Löchner, 2006) — rules derived from critical pairs were manually oriented, local confluence checked, and termination of the resulting system verified by AProVE. We consider this to be Slothrop's most significant achievement.

$1 * x \approx x$	$x^{-1} * x \approx 1$
$(x * y) * z \approx x * (y * z)$	$f(x) * g(y) \approx g(y) * f(x)$
$f(x * y) \approx f(x) * f(y)$	$g(x * y) \approx g(x) * g(y)$

Figure 10. The theory of two commuting group endomorphisms (CGE_2)

$(x * y) * z \to x * (y * z)$	$f(1) \to 1$
$x^{-1} * x \to 1$	$(f(x))^{-1} \to f(x^{-1})$
$x * x^{-1} \to 1$	$f(x) * f(y) \to f(x * y)$
$x * (x^{-1} * y) \to y$	$f(x) * (f(y) * z) \to f(x * y) * z$
$x^{-1} * (x * y) \to y$	$g(1) \to 1$
$(x * y)^{-1} \to y^{-1} * x^{-1}$	$(g(x))^{-1} \to g(x^{-1})$
$1 * x \to x$	$g(x) * g(y) \to g(x * y)$
$x * 1 \to x$	$g(x) * (g(y) * z) \to g(x * y) * z$
$1^{-1} \to 1$	$f(x) * g(y) \to g(y) * f(x)$
$(x^{-1})^{-1} \to x$	$f(x) * (g(y) * z) \to g(y) * (f(x) * z)$

Figure 11. Convergent completion of CGE_2

Using unfailing completion (Bachmair et al., 1989), Waldmeister is able to complete CGE_2 as well, but constructs a larger system which is ground-confluent only — i.e, it contains identities as well as rewrite rules. This system is often less helpful than a small convergent completion, for example, in characterizing the normal forms of the system for algebraic proof mining (Wehrman and Stump, 2005). Furthermore, Waldmeister does not appear to be able to find this ground-convergent completion fully automatically; a carefully selected Knuth-Bendix order must be provided. Slothrop is able to find the convergent completion with no input from the user other than the

theory itself. As shown in Fig. 12, the CGE_2 completion is discovered in about ten minutes when integrated with AProVE.

Test	In	Out	Time (s)	Orientations	Calls
G	3	10	2/1	56/56	30/30
GE_1	4	14	7/5	194/194	105/105
GE_2	5	18	35/33	680/680	370/370
CGE_2	6	20	606/1472	2356/2464	1381/2435

Figure 12. Summary of benchmarks (AProVE / TPA), including the number of identities in the theory (In), the number of rewrite rules in the completion (Out), the time in seconds to discover the completion (Time), the total number of orientations encountered (Orientations) and the number of calls to the termination checker (Calls).

A summary of benchmarks is shown in Fig. 12. Note that for smaller theories Slothrop performs comparably with both AProVE and TPA, with TPA slightly faster. For a larger theory like CGE_2, however, AProVE is significantly more efficient. It is believed that this is because AProVE can prove a wider variety of rewriting systems terminating, and hence can find a more direct path to a completion.

8. Conclusion

We have presented a new variant on Knuth-Bendix completion which does not require the user to provide a reduction order to orient identities. The procedure is correct for infinite executions and practically complete for decidable classes of orders. An implementation of the procedure, called Slothrop, can find convergent completions for a number of interesting theories without any input from the user, including one (CGE_2) which cannot be obtained by any existing tool.

8.1. Future Work

The performance of Slothrop does not approach that of well-tuned equational theorem provers such as Waldmeister for most tasks. Consequently, a primary goal of future work is to increase the efficiency of Slothrop. Basic heuristic search techniques have made the algorithm feasible for many theories, but it is still prohibitively slow for large theories — completion of the CGE_3 has not yet been achieved.

Though we have experimented with a variety of other heuristics, none are clearly superior on a wide range of problems than the one

described in Sect. 6. A variation on that heuristic which considers the size of only non-trivial critical pairs is occasionally advantageous, but not always. We have tried heuristics that sacrifice completeness of the algorithm. For example, it seems occasionally useful to note when AProVE needs a particularly long time to prove termination of a rewriting system, and one heuristic tends to avoid such a system. Any heuristic that takes timing information into account is worrisome for a variety of reasons, not the least of which that search is guided mainly toward easy parts of the search space. Modern search and learning techniques, e.g. as developed for SAT, may be applicable to the search for a convergent completion.

Finally, we would like to explore extensions to termination checking techniques to allow proofs to be constructed incrementally. This may significantly decrease the amortized time to prove a series of term rewriting systems terminating, since Slothrop tends to make a number of successive calls on rewriting systems whose rules form increasing chains.

Acknowledgements

The authors are especially appreciative of Peter Schneider-Kamp and the AProVE team for help integrating their system with Slothrop. We thank Stephan Falke for testing Slothrop and providing example theories. Thanks to Bernhard Gramlich for helpful pointers to research on completion divergence phenomena. We also thank Li-Yang Tan as well as the RTA 2006 referees for helpful comments on earlier drafts.

This work was partially supported by National Science Foundation grant CCF-0448275. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Baader, F. and T. Nipkow: 1998, *Term Rewriting and All That.* Cambridge University Press.

Bachmair, L.: 1991, *Canonical Equational Proofs*, Progress in Theoretical Computer Science. Birkhäuser.

Bachmair, L., N. Dershowitz, and D. Plaisted: 1989, 'Completion Without Failure'. In: *Resolution of Equations in Algebraic Structures*, Vol. 2, Rewriting Techniques. Academic Press, pp. 1–30.

Comon, H. and R. Treinen: 1997, 'The First-Order Theory of Lexicographic Path Orderings is Undecidable'. *Theoretical Computer Science* **176**(1–2), 67–87.

Contejean, E., C. Marché, and X. Urbain: 2004, 'CiME3'. Available at `http://cime.lri.fr/`.

Dershowitz, N. and J.-P. Jouannaud: 1990, 'Rewrite Systems'. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. pp. 243–320.

Dershowitz, N., J.-P. Jouannaud, and J. W. Klop: 1991, 'Open Problems in Rewriting'. In: R. Book (ed.): *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications (Como, Italy)*, Vol. 488. Berlin, pp. 445–456, Springer-Verlag.

Dershowitz, N. and D. Plaisted: 2001, 'Rewriting'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning*, Vol. I. Elsevier Science, Chapt. 9, pp. 535–610.

Dershowitz, N. and R. Treinen, 'The RTA list of open problems'. `http://www.lsv.ens-cachan.fr/rtaloop/`.

Filliâtre, J.-C.: 2003, 'Ocaml Data Structures'. Available at `http://www.lri.fr/~filliatr/software.en.html`.

Giesl, J., T. Arts, and E. Ohlebusch: 2002, 'Modular Termination Proofs for Rewriting Using Dependency Pairs'. *Journal of Symbolic Computation* **34**(1), 21–58.

Giesl, J., R. Thiemann, P. Schneider-Kamp, and S. Falke: 2004, 'Automated Termination Proofs with AProVE'. In: V. van Oostrom (ed.): *the 15th International Conference on Rewriting Techniques and Applications*. pp. 210–220, Springer.

Huet, G.: 1981, 'A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm'. *Journal of Computer and System Science* **23**(1), 11–21.

Kapur, D. and H. Zhang: 1995, 'An overview of Rewrite Rule Laboratory (RRL)'. *J. Computer and Mathematics with Applications* **29**(2), 91–114.

Knuth, D. and P. Bendix: 1970, 'Simple Word Problems in Universal Algebras'. In: J. Leech (ed.): *Computational Problems in Abstract Algebra*. pp. 263–297, Pergamon Press.

Koprowski, A.: 2006, 'TPA: Termination Proved Automatically.'. In: F. Pfenning (ed.): *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, Vol. 4098 of *Lecture Notes in Computer Science*. pp. 257–266, Springer.

Löchner, B. and T. Hillenbrand: 2002, 'The Next Waldmeister Loop'. In: A. Voronkov (ed.): *18th International Conference on Automated Deduction*. pp. 486–500.

Lynch, N. A. and F. W. Vaandrager: 1993, 'Forward and backward simulations – Part I: untimed systems.'. In: *135*. ISSN 0169-118X: Centrum voor Wiskunde en Informatica (CWI), p. 35.

Marché, C. and X. Urbain: 1998, 'Termination of Associative-Commutative Rewriting by Dependency Pairs'. In: *RTA*. pp. 241–255.

Milner, R.: 1989, *Communication and Concurrency*, International Series in Computer Science, C.A.R. Hoare, series editor. Prentice-Hall.

Nieuwenhuis, R.: 1993, 'Simple LPO Constraint Solving Methods'. *Information Processing Letters* **47**(2), 65–69.

Nieuwenhuis, R. and A. Rubio: 2001, 'Paramodulation-Based Theorem Proving'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning*, Vol. I. Elsevier Science, Chapt. 7, pp. 371–443.

Sattler-Klein, A.: 1991, 'Divergence Phenomena during Completion.'. In: R. V. Book (ed.): *Rewriting Techniques and Applications, 4th International Confer-*

ence, RTA-91, Como, Italy, April 10-12, 1991, Proceedings, Vol. 488 of *Lecture Notes in Computer Science*. pp. 374–385, Springer.

Sattler-Klein, A.: 1994, 'About Changing the Ordering During Knuth-Bendix Completion'. In: P. E. E. W. Mayr and K. W. Wagner (eds.): *Symposium on Theoretical Aspects of Computer Science*, Vol. 775 of *LNCS*. pp. 176–186, Springer.

Stump, A. and B. Löchner: 2006, 'Knuth-Bendix Completion of Theories of Commuting Group Endomorphisms'. *Information Processing Letters*. To appear.

Stump, A. and L.-Y. Tan: 2005, 'The Algebra of Equality Proofs'. In: J. Giesl (ed.): *16th International Conference on Rewriting Techniques and Applications*. pp. 469–483, Springer.

Suttner, C. B. and G. Sutcliffe: 1996, 'The TPTP Problem Library'. Technical Report JCU-CS-96/9.

TeReSe (ed.): 2003, *Term Rewriting Systems*, Vol. 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Wehrman, I. and A. Stump: 2005, 'Mining Propositional Simplification Proofs for Small Validating Clauses'. In: A. Armando and A. Cimatti (eds.): *3rd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning*.

Wehrman, I., A. Stump, and E. Westbrook: 2006, 'Slothrop: Knuth-Bendix Completion with Modern Termination Checking'. In: F. Pfenning (ed.): *17th International Conference on Term Rewriting and Applications*, Vol. 4098 of *Lecture Notes in Computer Science*. pp. 287–296, Springer.