Report Number: WUCSE-2006-45

2006-01-01

# Knuth-Bendix Completion with Modern Termination Checking, Master's Thesis, August 2006

Ian Wehrman

Knuth-Bendix completion is a technique for equational automated theorem proving based on term rewriting. This classic procedure is parametrized by an equational theory and a (well-founded) 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... **Read complete abstract on page 2.**

# Knuth-Bendix Completion with Modern Termination Checking, Master's Thesis, August 2006

Ian Wehrman

Complete Abstract:

Knuth-Bendix completion is a technique for equational automated theorem proving based on term rewriting. This classic procedure is parametrized by an equational theory and a (well-founded) 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 thesis, 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.

2006-45

# Knuth-Bendix Completion with Modern Termination Checking, Master's Thesis, August 2006

Authors: Ian Wehrman

Corresponding Author: iwerhman@cse.wustl.edu

Web Page: http://www.cse.wustl.edu/~iwehrman/

Abstract: Knuth-Bendix completion is a technique for equational automated theorem proving based on term rewriting. This classic procedure is parametrized by an equational theory and a (well-founded) 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 thesis, 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.

Type of Report: Other

SEVER INSTITUTE OF TECHNOLOGY

MASTER OF SCIENCE DEGREE

THESIS ACCEPTANCE

(To be the first page of each copy of the thesis)

DATE: July 26, 2006

STUDENT'S NAME: Ian A. Wehrman

This student's thesis, entitled Knuth-Bendix Completion with Modern Termination Checking has been examined by the undersigned committee of three faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Master of Science.

APPROVAL: _____ Chairman

_____

_____

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

---

# KNUTH-BENDIX COMPLETION WITH MODERN TERMINATION CHECKING

by

Ian A. Wehrman, B.Sc.

Prepared under the direction of Aaron Stump

---

Thesis presented to the Henry Edwin Sever Graduate School of
Washington University in partial fulfillment of the
requirements for the degree of

Master of Science

August 2006

Saint Louis, Missouri

WASHINGTON UNIVERSITY
THE HENRY EDWIN SEVER GRADUATE SCHOOL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

---

ABSTRACT

---

# KNUTH-BENDIX COMPLETION WITH MODERN TERMINATION CHECKING

by

Ian A. Wehrman

---

ADVISOR: Aaron Stump

---

August 2006

Saint Louis, Missouri

Knuth-Bendix completion is a technique for equational automated theorem proving based on term rewriting. This classic procedure is parametrized by an equational theory and a (well-founded) 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 thesis, 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.

# Contents

**Figures** . . . . . **v**

**Acknowledgments** . . . . . **vi**

**Typographic Conventions and Software** . . . . . **vii**

**1 Introduction** . . . . . **1**
1.1 Contributions . . . . . 5

**2 Preliminaries** . . . . . **7**
2.1 Logical Foundations . . . . . 7
2.1.1 Syntax: Terms and Identities . . . . . 8
2.1.2 Semantics . . . . . 9
2.2 Term Rewriting . . . . . 10
2.2.1 Proving Confluence with Critical Pairs . . . . . 12
2.2.2 Proving Termination with Well-founded Orders . . . . . 13
2.3 Executions and Simulations . . . . . 14

**3 Knuth-Bendix Completion** . . . . . **16**
3.1 Properties of the System $\mathcal{C}$ . . . . . 19
3.2 Limitations . . . . . 20

**4 The Role of the Order in Knuth-Bendix Completion** . . . . . **22**
4.1 Unordered Completion . . . . . 22
4.1.1 Properties of the System $\mathcal{C}_0$ . . . . . 23
4.2 Many-orderered Completion . . . . . 25
4.2.1 Properties of the System $\mathcal{C}_1$ . . . . . 26

**5 Knuth-Bendix Completion with Modern Termination Checking 29**
5.1 Properties of the System $\mathcal{A}$ . . . 31
5.1.1 Finite Correctness . . . 32
5.1.2 Total Correctness . . . 34
5.1.3 Practical Correctness . . . 37

**6 Implementing Completion with a Termination Checker . . . 38**
6.1 Huet's Completion Strategy . . . 39
6.2 Integration with AProVE . . . 39
6.3 Heuristic Search . . . 40
6.4 Limitations . . . 41

**7 Results and Performance . . . 42**
7.1 New Completions . . . 44

**8 Conclusion . . . 46**
8.1 Related Work . . . 46
8.2 Future Work . . . 48

**References . . . 49**

**Vita . . . 53**

# Figures


3-1 The Theory of Groups (G) . . . . . 17
3-2 A Convergent Completion of G . . . . . 17
3-3 Standard Knuth-Bendix Completion ($\mathcal{C}$) . . . . . 18

4-1 Generalized Knuth-Bendix Completion ($\mathcal{C}_0$) . . . . . 23
4-2 Many-ordered Knuth-Bendix Completion ($\mathcal{C}_1$) . . . . . 26

5-1 Modified Knuth-Bendix Completion ($\mathcal{A}$) . . . . . 30

7-1 The Theory of One Group Endomorphism ($GE_1$) . . . . . 42
7-2 Convergent Completion of $GE_1$ . . . . . 43
7-3 Time in AProVE Completing $GE_1$ . . . . . 43
7-4 Convergent $k$-completions of $GE_1$ . . . . . 44
7-5 The Theory of Two Commuting Group Endomorphisms ($CGE_2$) . . . 45
7-6 Convergent Completion of $CGE_2$ . . . . . 45

# Acknowledgments

Foremost thanks to my adviser Aaron Stump for helping and encouraging me from the beginning, for introducing me to subjects that keep me awake late, and for championing my ambitions. Also to Kenneth Goldman who kindly accepted me in his team and who with apparent fearlessness entrusted me with a few of his great ideas. Thank you both for countering my naiveté with gentle corrections and my frustrations with calm reassurances. I am grateful for your guidance and for helping to make my time at Washington University so enjoyable.

No small gratitude is due to my lab mates and fellow apprentices, whose company has made my study rich. I thank especially Li-Yang Tan and Edwin Westbrook for sharing their ideas and criticizing my own. Also to Joel Brandt, Alan Kwan, Robert Klapper, Sajeeva Pallemulle and Robert Zimmerman for sharing their time and knowledge with me.

Thanks to Robert Pless for graciously submitting to the task of serving on my thesis committee and for teaching me so many ways to compute a convex hull.

I am especially appreciative of Peter Schnieder-Kamp and the rest of the AProVE team at RWTH Aachen University for accommodating integration of their splendid system with our Slothrop project. Thanks also to Stephan Falke at the University of New Mexico for providing feedback on an earlier version of Slothrop.

Thanks to my wonderful girlfriend Britt Caldwell, for her support through hard times, inspiring radiance and inexplicable willingness to let me ramble without end about the termination of rewriting systems. Whatever credit not attributed above is owed to my mother and father. Thank you both for your enduring love and support.

The work described in this thesis was partially supported by NSF grant CCF-0448275.

Ian A. Wehrman

*Washington University in Saint Louis*
*August 2006*

# Typographic Conventions and Software

**Typographic Conventions** We use the following typographic conventions, with some exceptions, throughout the document. Term 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, substitutions and ordinals are all lowercase italic Greek letters; for executions we generally use $\alpha, \beta, \gamma$; for substitutions we use $\sigma, \tau, \lambda$; for ordinals we use $\iota, \kappa, \mu$. Inference systems are named with calligraphic capital letters $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$; their transitions are labeled with words set in SMALL CAPS.

**Software** Software developed during research for this thesis, including source code, is available online at `http://cl.cse.wustl.edu/`. With luck, this will continue to be the case for perpetuity.

The name of the project, SLOTHROP, references the eponymous drifter protagonist of Thomas Pynchon's 1973 novel *Gravity's Rainbow*. Additionally, the quotations that open each chapter are from that and other novels written by Mr. Pynchon.

# Chapter 1

# Introduction

> No, this is not a disentanglement from, but a progressive *knotting into*...
>
> *Gravity's Rainbow*

The field of *automated theorem proving*—which is also known as *automated deduction* and which is considered a subfield of the more general area of *automated reasoning*, itself an area of artificial intelligence—considers algorithms and tools for proving or disproving (e.g., by counterexample) mathematical statements. An automated theorem prover is a program that distinguishes among those formulas that are theorems and those that are not. The field has developed historically along two separate tacks, the first and oldest being general-purpose, uniform methods to search for proofs of any sentence of first-order logic, and the second in which efficient algorithms for solving problems in highly specialized domains are sought.

General methods of proving theorems of first-order logic arose in the 1950s, e.g. with Newell, Shaw and Simon's Logic Theory Machine [27], and today are largely based on variants of the *resolution method* presented by Alan Robinson in 1965 [30]. Recently, however, it has come to be believed that the second approach, involving tools sharpened for specific mathematical domains, has greater promise [32]. After all, a simple, efficient and general method of proof would be almost too good to be true, neatly fulfilling Leibniz' dream of a *calculus ratiocinator* to settle algorithmically disputes among philosophers. Consequently, attention has focused on algorithms for answering questions about relatively small domains: perhaps formulae with limited quantification or without variables, or that consider only simple,

finitely axiomatiziable mathematics such as linear arithmetic (i.e., without multiplication) or other such modest algebraic structures. Individual tools for solving these simpler questions can then be combined to build a robust theorem prover and indeed this is the basic architecture taken by many of the most effective modern systems [5, 7].

In this thesis, we focus on automated techniques for answering questions about an ostensibly humble class of formulas containing only identities[1] between terms, written $t_1 \approx t_2$. All we can ask of a prover for this domain is whether or not a pair of terms are considered identical under the assumption that some other identities hold,

$$u_1 \approx v_1, u_2 \approx v_2, \ldots, u_n \approx v_n \models t_1 \approx t_2.$$

We call these assumptions—the identities left of the turnstile above—a *theory*. The question posed is called the *word problem* because we ask is whether or not two words in a formal language are identified by a given theory. For example, to answer questions about addition of natural numbers, our theory[2] would consist of the two identities $0+x \approx x$ and $x'+y \approx x+y'$. We could then ask whether the terms $0'+0'$ and $0''$ are identified in this theory. Indeed they are, since $0'+0' \approx 0+0''$ by the first assumption and $0+0'' \approx 0''$ by the second assumption.

The preceding two-step chain of reasoning illustrates the concept of *proof*. When a mathematician distinguishes between theorems and non-theorems, he does so by constructing proofs which can then be shared to convince others of the result. Automated theorem provers proceed similarly, searching for proofs and responding positively when such a proof is discovered that settles the question originally set forth. If an algorithm implemented in a theorem prover is guaranteed to eventually produce an affirmative when true mathematical statements are provided, then it is a *semidecision procedure*. If an algorithm has the yet stronger property that it also promises to eventually halt when non-theorems are provided, then it is a *decision procedure*, so called because the algorithm can be trusted to decide any question about a particular theory. Decision procedures are powerful tools, but semidecision procedures somewhat less so — the user of a semidecision procedure

[1] We intentionally distinguish between *identity* $\approx$ and *equality* $=$ throughout, the first relating formulae that are assumed to hold (are universally quantified) and the second relating objects that are one in the same.

[2] We write $x'$ to denote the successor of the natural number $x$.

has no indication of whether a proof that a formula is a theorem will be found after an hour or a year, or if one will never be found because the formula is not a theorem.

There are many areas of mathematics with decision procedures, such as propositional logic, group theory, and arithmetic over the real number line. We call these domains *decidable*. But unfortunately, not all mathematical domains are decidable. For example, there is not and can never be a decision procedure for the seemingly tame setting of integer arithmetic with addition and multiplication [18]. Nor is there a decision procedure for finding solutions to systems of Diophantine equations (polynomials with integral coefficients) [9]. And there are certainly no decision procedures for more sophisticated domains like set theory or Turing machines. Such domains that can not have decision procedures are called *undecidable*.

Somewhat surprisingly, our restricted domain of identities falls into this second class—the word problem is undecidable in general, and so there can be no decision procedure for it. However, over thirty years ago Donald Knuth developed a remarkably effective general technique for solving the word problem for a great many theories, though of course not all. His technique, called *completion*, remains the most effective tool for solving particular instances of the word problem today. Its continual popularity is justified by its relative simplicity and efficiency, by the breadth of questions that can be answered, and also because it is a semidecision procedure. Although it can often disprove false statements, it cannot always do so. It can however be reliably expected to find proofs of true statements regardless of the theory.

Knuth's completion procedure also has the nice property that successful executions result not just in the answer to a question about a single identity in a theory, but the ability to determine whether or not *any* pair of terms are identified by the theory. Completion does this by generating what is called, somewhat confusingly, a *convergent completion*. A convergent completion is a new set of identities, equivalent to the original theory, but with the stronger, more useful property of being oriented, of having a left-to-right sense. The convergent completion can be used to *rewrite* terms by repeatedly replacing instances of a left-hand side in a term with an instance of a right-hand side. This sort of process is of course possible with any set of identities, but when performed with a convergent completion, it has the reassuring quality of reliably terminating with the same result regardless of the manner in which rules are applied. Accordingly, we use the convergent completion to solve the

word problem by simply rewriting both sides of an identity and observing whether or not they are syntactically equal. Using a convergent completion, identities that are consequences of a theory will always rewrite to equal terms, and identities that are not will always rewrite to different terms. In other words, syntactic equality implies identifiability in the theory. (Convergent completions are also interesting objects of study in their own right [34, 38].)

Completion, however, suffers from a fairly major drawback for an automated theorem proving procedure: viz. that it is not quite as automatic as one would like.[3] Ideally, an automated theorem prover would accept as input just the statement to be proved (or disproved) and the definition the the theory (in our case, a set of identities). Knuth's completion procedure fails to live up to this expectation, however, by also requiring that the user provide a well-founded ordering relation (e.g., the usual less-than relation $<$ on the natural numbers) compatible with the given theory (i.e., both sides of each identity are comparable), and with other deductive consequences of the theory. This is a major caveat: in practice, finding and specifying such an order can be extremely challenging, even for experienced users.

All tools that implement Knuth's completion procedure allow the the user to specify orders that fall into a small handful of classes, often only one or two. (Some common classes are defined in Sect. 2.2.) There are a number of problems with this approach. First, there are many theories that cannot be oriented using the few classes of orders usually implemented by these tools. Consequently there are many theorems that cannot be proved in practice with completion simply because the complex orders need to orient the assumptions 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 beyond halting when an incompatibility is detected. And note that the task of finding a compatible order is computationally difficult in its own right—NP-hard at best and undecidable at worst. Furthermore, even if the user discovers an order which is compatible with the identities that define the given theory (which are known at the outset), there is no way to know whether or not the order will be compatible with the deductive consequences of the assumptions generated during execution of the procedure.

[3] Although there are *interactive* tools for constructing proofs [36]—useful and interesting in their own right— for problems in relatively mundane domains like ours we have more lofty expectations.

In this thesis, we address these problems with a new variation on Knuth's original completion procedure. In our revised procedure, the user need not provide a well-founded order as input. Instead, we search for well-founded orders using heuristic search techniques. In the theoretical treatment of our procedure, we rely on an oracle to bound the search by distinguishing among orders that are well-founded and those that contain infinitely decreasing sequences. In practice, this oracle is replaced by a program called a *termination checker* which determines whether a set of identities possesses a property called termination that coincides with the existence of compatible well-founded orderings. Termination checking is an active field which has made great strides in recent years. Many modern termination tools exist, any one of which may be integrated with our procedure, to discover well-founded orders.

By leveraging a modern termination checker, we partially or fully solve the above problems with standard completion. Because termination checkers can detect well-founded orders for a wide variety of theories (as opposed to just a small handful as found in all other completion implementations), implementations of our new procedure have the practical capability of succeeding on a wider variety of theories. Also, because the order is discovered by the tool instead of provided by the user, a burden is removed from the user and could 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.

## 1.1 Contributions

Informally, the major contribution of the thesis is a variation of completion that does not require the user to supply a well-founded order and which provides the same correctness properties as the standard procedure. The algorithm is equally effective as completion in theory, but can be considered more effective in practice by nature of its success on problems for which no implementation of standard completion has yet succeeded.

In more detail, this thesis presents the following contributions to the field of automatic theorem proving:

1. An algorithm for automatically solving the word problem of finite theories is developed. The algorithm is a semidecision procedure for the word problem for

any theory. It also produces convergent completions for many theories, yielding decision procedures for such theories. The new procedure allows convergent completions to be discovered for theories which have never been produced by a fully automatic algorithm.

2. A complete proof of correctness for the algorithm is presented. In a previously published paper [39], we proved the algorithm correct for finite executions only. Here we extend that work by proving that the algorithm is correct for both finite and infinite executions. Our proof implies that the revised algorithm is a semidecision procedure for the word problem. The finite case is proved by *simulation* [25] of the standard Knuth-Bendix completion algorithm. In the infinite case, a more complicated proof technique is used that relies on a different simulation and an argument about the ability to construct such simulations nondeterministically.

3. An implementation of the new algorithm is presented. This tool, which we call SLOTHROP and whose source code is freely available on the Internet, is written in the Ocaml programming language and integrates with powerful termination checker called APROVE. This integration came about as a result of help from members of the APROVE team.

4. SLOTHROP yielded the first fully automatically constructed convergent completion of the theory of two commuting group endomorphisms (described in Chap. 7). Although other tools can produce ground-convergent completions (a less useful object defined in Chap. 2) automatically, and manual methods have been used to discover the convergent completion [33], no other tool has been able to produce such a convergent completion without user intervention.

# Chapter 2

# Preliminaries

> Who can find his way about this lush maze of initials, arrows solid and dotted, boxes big and small, names printed and memorized?
>
> *Gravity's Rainbow*

This chapter briskly develops background necessary for discussion of completion-based automated theorem proving. In Sect. 2.1, we introduce the basic syntactic objects of study required to prove theorems automatically, and describe the connections between these objects and the truths we seek. In Sect. 2.2, we present basic notions of term rewriting, the computational formalism upon which completion-based theorem proving is predicated. The notions are standard for techniques based on rewriting; more detailed presentations, including proofs of well-known theorems and lemmas, can be found in [1, 10, 37, 12].

## 2.1 Logical Foundations

In automated theorem proving, proofs or disproofs are typically sought for statements in the form of logical formulae, which are defined inductively from *atomic formulae* (which are themselves built of terms) and the familiar *logical connectives* ($\vee, \wedge, \neg$, etc). We will instead take a more direct path, suitable for our purposes (the study of completion), and imbue our terms and identities with logical meaning in the form of an algebraic semantics.

### 2.1.1 Syntax: Terms and Identities

A *signature* $\Sigma$ is a finite set of function symbols, each with an associated *arity* $n \in \mathbb{N}$. Those function symbols with arity zero are called *constants*. Let $X$ be a countable set of *variable* symbols, such that $\Sigma \cap X = \emptyset$. The set of *terms* constructed over $\Sigma$ and $X$, written $T(\Sigma, X)$, is defined inductively by:

- $X \subseteq T(\Sigma, X)$.
- If $f \in \Sigma$ with arity $n$ and $t_1, \ldots, t_n \in T(\Sigma, X)$ then $f(t_1, \ldots, t_n) \in T(\Sigma, X)$.

In case a signature contains only single-arity function symbols, we omit parentheses and variables when terms are written. For example, if $\Sigma = \{g, f\}$, both of which are single-arity, then we write $fgf$ as shorthand for $f(g(f(x)))$. In all other signatures, parentheses are omitted only for constants.

The set of *positions* of a term $t$, written $Pos(t)$, is a set of strings defined by induction on $t$:

- If $t = x \in X$, then $Pos(t) = \{\epsilon\}$
- If $t = f(t_1, \ldots, t_n)$, then

$$Pos(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(t_i)\}.$$

The *subterm* of term $t$ at position $p$, written $t|_p$, is defined by simultaneous induction on $p$ and $t$:

- If $p = \epsilon$, then $t|_p = t$.
- If $p = ip'$ and $t = f(t_1, \ldots, t_n)$ with $1 \leq i \leq n$, then $t|_p = t_i|_{p'}$.

If $s$ is a subterm of $t$ at position $p$, we write $s \trianglelefteq t$, and if $p \neq \epsilon$ then $s$ is a *strict* subterm, written $s \triangleleft t$. A *variable position* in a term $t \in T(\Sigma, X)$ is any $p \in Pos(t)$ such that $t|_p = x \in X$. The set $Vars(t)$ contains all $x \in X$ such that there is some variable position $p \in Pos(t)$ with $t|_p = x$. The *size* of a term $t$ is $|Pos(t)|$. For a term $t$ and position $p \in Pos(t)$ the *replacement* of $s$ in $t$ at $p$, written $t[s]_p$, is defined by simultaneous induction on $p$ and $t$:

- If $p = \epsilon$, then $t[s]_p = s$
- If $p = ip'$ and $t = f(t_1, \ldots, t_n)$ with $1 \leq i \leq n$, then $t[s]_p = f(t_1, \ldots, t_i[s]_{p'}, \ldots, t_n)$.

A *substitution* $\sigma$ is a total function from a set of variables $X$ to terms $T(\Sigma, X)$ in which $\sigma(x) \neq x$ only for $x \in X' \subset X$ with $X'$ finite. The *instantiation* of term $t$ with substitution $\sigma$, written $t\sigma$, is defined by induction on $t$:

- If $t = x \in X$, then $t\sigma = \sigma(t)$
- If $t = f(t_1, \ldots, t_n)$, then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.

Instantiation associates to the left: $t\sigma\tau = (t\sigma)\tau$. A substitution is a *unifier* of terms $s, t \in T(\Sigma, X)$ if $s\sigma = t\sigma$ and a *most general unifier* (MGU) if for all unifiers $\theta$ of $s, t$ there exists another unifier $\lambda$ such that $s\theta = s\sigma\lambda = t\sigma\lambda = t\theta$.

For some signature $\Sigma$, a $\Sigma$-*identity* (or simply identity when the signature is clear or irrelevant) is a pair of terms $s, t \in T(\Sigma, X)$, written $s \approx t$. An identity $s \approx t$ is *trivial* if $s = t$. For a set of identities $E$, terms $s$ and $t$ are in the *reduction relation* on $E$, written $s \rightarrow_E t$, if for some identity $(l, r) \in E$, position $p \in Pos(s)$ and substitution $\sigma$, we have $s|_p = l\sigma$ and $t = s[r\sigma]_p$. We write $\overset{=}{\rightarrow}_E$, $\overset{+}{\rightarrow}_E$, $\overset{*}{\rightarrow}_E$ and $\overset{*}{\leftrightarrow}_E$ for the reflexive, transitive, reflexive-transitive and reflexive-transitive-symmetric closures of $\rightarrow_E$, respectively. We write $\leftarrow_E$ for the inverse of $\rightarrow_E$, $\overset{*}{\leftarrow}_E$ for the inverse of $\overset{*}{\rightarrow}_E$, etc. If $s \overset{*}{\leftrightarrow}_E t$, then we say that the identity $s \approx t$ is a *syntactic consequence* of $E$.

### 2.1.2 Semantics

We now briefly discuss semantics for identities and reduction relations, and state the connection between the syntactic and semantic notions.

Let $\Sigma$ be a signature as before. Then a $\Sigma$-*algebra* $\mathcal{A}$ consists of a *carrier set* (or domain) $A$ and a mapping which associates with every $f \in \Sigma$ having arity $n$ a function $f^{\mathcal{A}} : A^n \rightarrow A$. For a set of variables $X$, a $\Sigma$-*term algebra* $\mathcal{T}(\Sigma, X)$ is an algebra with domain $T(\Sigma, X)$ and a mapping such that (informally) $f^{\mathcal{T}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. A $\Sigma$-*homomorphism* is a function $\phi : \mathcal{A} \rightarrow \mathcal{B}$, for any algebras $\mathcal{A}$ and $\mathcal{B}$, such that $\phi(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(\phi(a_1), \ldots, \phi(a_n))$, for $f \in \Sigma$ with arity $n$, $1 \leq i \leq n$ and $a_1, \ldots, a_n \in A$.

An identity $s \approx t$ *holds* in an algebra $\mathcal{A}$ iff for all homomorphisms $\phi$ from the associated term algebra into $\mathcal{A}$, $\phi : \mathcal{T}(\Sigma, X) \to \mathcal{A}$, it is the case that $\phi(s) = \phi(t)$. Intuitively, an identity holds in an algebra if the interpretations of both sides are the same regardless of the interpretations of the variables. The algebra $\mathcal{A}$ is a *model* of a set of identities $E$ if every identity of $E$ holds in $\mathcal{A}$. An identity $s \approx t$ is a *semantic consequence* of $E$ if it holds in every model of $E$. We define the *equational theory* of $E$, written $\approx_E$, as the set of all semantic consequences of $E$; for $s, t \in T(\Sigma, X)$, $s \approx_E t$ iff $s \approx t$ is a semantic consequence of $E$.

We can now state the crucial connection between syntactic objects easily manipulated by a computer and the mathematical objects we wish to reason about. The following theorem states that semantic consequence coincides with syntactic consequence. If an identity between terms is shown to be a syntactic consequence of $E$, then the mathematical objects which the terms represent are also identical in the same theory.

**Theorem 1** (Birkhoff)**.** *For a set of $\Sigma$-identities $E$ and terms $s, t \in T(\Sigma, X)$ for any set of variables $X$, $s \overset{*}{\leftrightarrow}_E t$ iff $s \approx_E t$ [1].*

It follows that the membership in the reduction relation is a necessary and sufficient condition for truth in a mathematical domain which can be axiomatized with a finite set of identities. In particular, an identity $s \approx t$ is valid in $E$ iff $s \approx_E t$. Because of this connection, in the sequel we will abuse terminology somewhat and refer to both $\overset{*}{\leftrightarrow}_E$ and $E$ itself as the equational theory (or simply the theory).

## 2.2 Term Rewriting

For terms $s$ and $t$ and a finite set of identities $E$, deciding whether or not $s \approx_E t$ is called the *word problem* for $E$. The word problem is undecidable in general [6]. However, there are many theories for which the word problem is decidable. For example, there exist efficient solutions for the world problem when the equational theory consists of ground identities only; this problem is called *congruence closure* [4]. We also know that, by Thm. 1, a method for deciding $\overset{*}{\leftrightarrow}_E$ also decides the equational theory of $E$, and hence the word problem. We now give a brief overview of the field of *term rewriting*, in which techniques exist for solving the word problem for many cases of $E$.

A *rewrite rule* $l \to r$ is an identity, with $l$ not a variable, and $Vars(r) \subseteq Vars(l)$. A set of rewrite rules is called a *term rewriting system* (or rewriting system, or TRS). Because a TRS $R$ is simply a set of identities, we use (without abuse) the notation $\to_R$ to denote the reduction relation on $R$. A TRS is *finite* if it has a finite number of rewrite rules. We also assume w.l.o.g. that $Vars(l_1) \cap Vars(l_2) = \emptyset$ and $Vars(r_1) \cap Vars(r_2) = \emptyset$ for any pair of rewrite rules $l_1 \to r_1, l_2 \to r_2$ in a TRS.

We now present terminology useful when discussing rewriting systems, assuming some reduction relation $\to$. A term $t$ is *reducible* if, for some $t'$, $t \to t'$ and a *normal form* otherwise. Terms $t_1$ and $t_2$ are *joinable* if there exists $t'$ such that $t_1 \stackrel{*}{\to} t' \stackrel{*}{\leftarrow} t_2$, in which case we write $t_1 \downarrow t_2$. The reduction relation $\to$ is *globally confluent* (or simply *confluent*) if $t_1 \stackrel{*}{\leftarrow} t \stackrel{*}{\to} t_2$ implies $t_1 \downarrow t_2$, and *locally confluent* if $t_1 \leftarrow t \to t_2$ implies $t_1 \downarrow t_2$. It is *terminating* if there are no sequences of reductions with infinite length, $t_1 \to t_2 \to \cdots$. Note that termination of a relation implies the strictly weaker property that every term has a normal form, but not the other way around.[1] If a rewriting system is both confluent and terminating, then it is called *convergent*; if these properties hold only for ground terms, then it is *ground-convergent*. Note that ground-convergence is a weaker property than convergence. For convenience, we will say a TRS $R$ has one of the above properties when its reduction relation $\to_R$ has it.

By Thm. 1, we know that syntactic consequence implies semantic consequence. Therefore, to solve the word problem for a set of identities $E$ it suffices to decide syntactic consequence of $E$. Since the word problem and semantic consequence is in general undecidable, so then is syntactic consequence. In the special case when $E$ is a finite convergent rewriting system, however, we have an effective test for syntactic consequence. The following theorem states that an identity is joinable in a convergent theory $E$ iff the identity is a syntactic consequence of $E$.

**Lemma 1.** *If $E$ is convergent, then $\stackrel{*}{\leftrightarrow}_E = \downarrow_E$.*

This says that for a convergent TRS, we can determine syntactic consequence by testing joinability. But two terms are joinable if they have the same normal form, and since the system is terminating, these normal forms can be found after only a finite number of reductions. It follows that if the TRS is itself finite, then

[1] E.g., with the TRS $\{f(x) \to f(x), f(x) \to a\}$, every term has a normal form $a$, but there also exist infinite sequences of reductions $f(a) \to f(a) \to \cdots$ .

normal forms can be obtained finitely, joinability can be decided, and hence syntactic consequence and hence semantic consequence. In summary, for finite convergent theories $E$, the word problem for $E$ is decidable.

**Theorem 2.** *If $E$ is finite and convergent, then $\approx_E$ is decidable.*

The difficulty then becomes determining whether a set of identities is convergent. (It is assumed that we can determine whether or not it is finite.) The next two sections consider strategies for proving confluence and termination of rewriting systems.

### 2.2.1 Proving Confluence with Critical Pairs

Demonstrating confluence of a rewriting system is undecidable in general. However, there is a straightforward procedure for testing local confluence in the special case of finite terminating rewrite systems by inspecting only combinations of rewrite rules (instead of all possible terms). For a TRS $R$ with $l_i \to r_i \in R$ for $i = 1, 2$ such that $\sigma$ is an MGU of $l_1|_p, l_2$, and where $l_1$ is not a variable, a *critical pair* of $R$ is a tuple $\langle r_1\sigma, (l_1\sigma)[r_2\sigma]_p\rangle$. We write $\Gamma(R)$ to denote the set of all critical pairs between all rules of $R$. The following lemma states that if a term can be rewritten to two different terms in one step each, then either those two terms are joinable or the two new terms differ by the presence of different elements of a critical pair.

**Lemma 2** (Critical Pairs)**.** *For a TRS $R$, If $t_1 \leftarrow_R t \to_R t_2$, then either $t_1 \downarrow_R t_2$ or $t_1 = s[u_1]_p$ and $t_2 = s[u_2]_p$ for some term $s$, position $p \in Pos(s)$ and critical pair $\langle u_1, u_2\rangle \in \Gamma(R)$.*

It follows that to test local confluence, it suffices to test all critical pairs of a TRS for joinability. In case the TRS is finite and terminating, there are only finitely many critical pairs and testing joinability of each takes only finitely many reduction steps, and so local confluence is decidable. Furthermore, the following lemma states that for such terminating rewriting systems, local confluence coincides with global confluence and, consequently, confluence is decidable for terminating rewriting systems.

**Lemma 3** (Newman)**.** *A terminating TRS is locally confluent iff it is confluent.*

### 2.2.2 Proving Termination with Well-founded Orders

As mentioned above, demonstrating confluence or termination of a TRS is undecidable in general. However, we have seen that confluence is decidable for any finite terminating TRS. Consequently, we now to turn to methods for demonstrating termination of a TRS. To prove termination of a TRS, it suffices to show that its reduction relation is compatible with some well-founded order on the terms; for in this case, there can be no reduction sequences of infinite length.

A *rewrite order* $>$ is a strict order on terms of $T(\Sigma, X)$ which is compatible is $\Sigma$-operations (i.e., $t > t'$ implies $f(t_1, \ldots, t, \ldots, t_n) > f(t_1, \ldots, t', \ldots, t_n)$ and closed under substitutions (i.e., $t > t'$ implies $t\sigma > t'\sigma$). If a rewrite order is well-founded, it is called a *reduction order*. We say a reduction order $>$ is *compatible* with a rewriting system $R$ if, for all $l \to r \in R$ it is the case that $l > r$.

**Theorem 3.** *A TRS $R$ is terminating iff there exists a reduction order compatible with $R$.*

*Sketch.* In one direction, the relation $\xrightarrow{+}_R$ is itself a reduction order. In the other, $t \to t'$ implies $t > t'$ due to compatibility and closure under substitutions, and no infinite sequence of reductions exists because of well-foundedness. □

The *lexicographic extension* of an order $>$, written $>^{\#}$, extends an order on terms to an order on lists of terms. For terms $s_1, \ldots, s_n \in T(\Sigma, X)$ and $t_1, \ldots, t_m \in T(\Sigma, X)$, $\langle s_1, \ldots, s_n \rangle >^{\#} \langle t_1, \ldots, t_m \rangle$ iff $s_1 > t_1$ or both $s_1 = t_1$ and $\langle s_2, \ldots, s_n \rangle >^{\#} \langle t_2, \ldots, t_m \rangle$.

A wide variety of methods exist for proving termination; an excellent survey is found in [37]. One class of reduction orders that is simple to describe are the *lexicographic path orders* (LPOs). An LPO is a complex order parametrized by a more simple order called a *precedence*. The precedence order $>$ is a well-founded partial order (i.e., antisymmetric as well as reflexive and transitive) on the function symbols of $\Sigma$. The LPO $>_{\text{lpo}}$ induced by precedence $>$ is defined by induction on terms. For $s, t \in T(\Sigma, X)$, $s >_{\text{lpo}} t$ iff $s = f(s_1, \ldots, s_n)$ (i.e., $s$ is not a variable) and

1. $s_i = t$ for some $1 \leq i \leq n$; or
2. $s_i >_{\text{lpo}} t$ for some $1 \leq i \leq n$; or

3. $t = g(t_1, \ldots, t_m)$, $s >_{\text{lpo}} t_i$ for all $1 \leq i \leq m$, and either

   (a) $f > g$; or

   (b) $f = g$ and $\langle s_1, \ldots, s_n \rangle >^{\#}_{\text{lpo}} \langle t_1, \ldots, t_m \rangle$.

That the relation defined above is a reduction order is proved in [1, 10, 37]. For a given LPO $>_{\text{lpo}}$, deciding if $s >_{\text{lpo}} t$ holds requires time polynomial in the size of the terms, but deciding if there is some LPO which satisfies the relation is NP-hard [29]

There are a number of orders similar to the lexicographic path order. The *recursive path orders* (RPOs) are orders defined similarly to the LPOs, but where the order is extended to lists of terms in different manners. For example, the multiset path order uses a multiset extension to resolve order between lists. The *Knuth-Bendix orders* (KBOs) use a precedence function as well as a weight function, which maps function symbols to positive integers. Detailed descriptions of these and other generalizations are given in [37, 1].

## 2.3 Executions and Simulations

In this section we describe inference systems, executions of those systems, and simulations between systems briefly and informally. Tempting though it may be (cf. Sect. 2.1), we are not excessively formal here because it is neither the focus of this thesis nor necessary to develop our results.

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. We require *1*) that for every rule the domain of the rule is the same as its range *2*) that all rules share the same domain; and *3*) that the start states are included in that domain.Rules may, however, have preconditions (which we do not specify formally here) that otherwise restrict their application. We take the liberty below of identifying inference systems and their rule sets.

Informally, an *execution* is a sequence of states related to one another by the rules of some inference system. More formally, we think of an execution as a function that maps an index to a state. Since we consider both finite and infinite executions, we will define an execution $\beta$ as a function of type $\beta : \lambda \mapsto \mathscr{U}$, with $\lambda$ a

countable ordinal. We say an execution $\beta : \lambda \mapsto \mathscr{U}$ has *length* $\lambda$, which we write $|\beta|$. In the this thesis, the universe of states will consist of tuples of rewriting systems. For an ordinal $\iota$ and execution $\beta$, we may write either $\beta(\iota)$ or $\beta_\iota$ to denote the state at index $\iota$, assuming $|\beta| > \iota$. State $s \in \mathscr{U}$ is a *deduction* of execution $\beta$ if there is some index $\iota$ 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 $\beta$ extends the function $\beta'$ 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 $\beta$ of system $\mathcal{B}$ is *valid* if it is a well-defined function and

1. $\beta_0$ is a start state of $\mathcal{B}$;
2. if $\beta_{\kappa'}$ (where $\kappa'$ denotes the successor of ordinal $\kappa$) 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 infinite executions.

Throughout this thesis, we rely on proofs by *simulation*. Informally, a simulation from a system $\mathcal{B}_1$ to a system $\mathcal{B}_2$ is the construction of an execution of $\mathcal{B}_2$ given an execution of $\mathcal{B}_1$ that is equivalent in the sense that the new execution is valid in both systems and 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 following lemma (more carefully developed elsewhere [25]):

**Lemma 4** (Simulations)**.** *If all executions of a system $\mathcal{B}_2$ have property $P$ and there exists a simulation from system $\mathcal{B}_1$ to system $\mathcal{B}_2$, then all executions of system $\mathcal{B}_1$ have property $P$.*

# Chapter 3

# Knuth-Bendix Completion

> Things then did not delay in turning curious.
>
> *The Crying of Lot 49*

Thus far we have seen that the word problem is decidable for finite convergent theories, and briefly investigated techniques for demonstrating the convergence of theories. Once we have shown that a theory is convergent, then by Thm. 2, we can solve the word problem by simply rewriting terms to their normal forms and performing a simple equality test; the normal forms are equal iff the identity holds in the theory.

What if we want to solve the word problem for a finite but 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 could solve $\approx_{E'}$ and therefore $\approx_E$. We now investigate a general procedure for discovering such equivalent, convergent theories.

A *Knuth-Bendix completion procedure* [21], invented in 1967 by Donald Knuth,[1] (or simply a *completion procedure*) 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

[1] Peter Bendix, Knuth's student, contributed a Fortran implementation [11].

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 safely (finitely) be used to calculate normal forms during execution.

As an example, the theory of groups (G) is presented in Fig. 3-1. 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 a function symbols as described above.

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

Figure 3-1: The Theory of Groups (G)

One possible completion of G is shown in Fig. 3-2, 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.

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

Figure 3-2: 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 [2], and proved it correct (stated in Sect. 3.1). We refer to this standard inference system as $\mathcal{C}$ because it will serves 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. 3-3. The notation $s \stackrel{\cdot}{\approx} t$ means either $s \approx t$ or $t \approx s$. The notation $s \stackrel{\sqsupset}{\to}_R v$ 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 $s \leftarrow_R u \rightarrow_R t$ |
| 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 $s \rightarrow_R u$ |
| 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 $s \overset{\sqsupset}{\rightarrow}_R v$ |

Figure 3-3: 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 $\gamma$ 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* $E_\omega$ (*persistent rules* $R_\omega$) 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 $j > i$,

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

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 consider as infinite executions: a finite execution $\gamma$ 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.

## 3.1 Properties of the System $\mathcal{C}$

An execution $\gamma$ of $\mathcal{C}$ *succeeds* if $E_\omega = \emptyset$ and $R_\omega$ is a convergent rewriting system equivalent to $E$ as described above. An execution $\gamma$ 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 $\gamma$. An execution 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 all critical pairs of $R_\omega$,

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

In this thesis, 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 4** (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. *$R_\omega$ is equivalent to the set of input identities $E_0$.*
2. *$R_\omega$ is convergent.*
3. *If $R_\omega$ 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 $R_\omega$ 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 proof of Thm. 4 is quite involved [1], and here we will only provide some explanation of the last point: how can an infinite execution of $\mathcal{C}$ that results in an infinite rewriting system be used as a semidecision procedure for the word problem of the input identities?

*Informal argument.* Since the set of persistent rules is an equivalent convergent rewriting system, then it could be used as a decision procedure, but if the execution is infinite, it cannot be explicitly constructed. However, if an identity is in fact a semantic consequence of the input theory, then there would be a finite sequence of reductions using the persistent rules which would show that both terms have the same normal forms. Since the reductions on the terms would be finite, then they only require a finite number of rules. By definition of the persistent sets, then, there is some first intermediate system which includes all of these necessary rules. So infinite executions can be used as a semidecision procedure if, at each step, both sides of the identity are normalized and tested for equality. If the identity holds, then after some finite amount of time the normal forms will be the same. Conversely, if the normal forms are the same, we know the identity holds. This is not a full decision procedure, however, because if the normal forms are not the same, then it may just be the case that we have not generated enough persistent identities to prove the equality — there is no way to tell after a finite number of steps. □

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 5** (Completeness of $\mathcal{C}$)**.** *Every fair completion procedure that does not fail succeeds [2, 1].*

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.

## 3.2 Limitations

An execution of the standard completion procedure $\mathcal{C}$ will fail if not provided a reduction order which is compatible with all of the rules of $R_\omega$. The basic difficulty

with completion is in finding an appropriate reduction order. In some cases no such 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. In other cases, there exist reduction orders which are compatible with the rules in the input set of identities, but not with rules generated later.

To complicate matters further, all known implementations of completions allow the user to specify only a limited class of reduction orders. This is largely out of necessity; in order to provide a suitable order, a user must be able to specify it simply. Nearly all tools allow the user to specify a precedence for generating an LPO or a precedence and weight mapping for a KBO. In fact, we know of no implementations of completions that accept orders other than RPOs or KBO as input.

It is not decidable in general whether a RPO exists which is compatible with a particular theory [8]. 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 [29]. 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.

# Chapter 4

# The Role of the Order in Knuth-Bendix Completion

> Living as he does much of the time in a world of metaphor, the poet is always acutely conscious that metaphor has no value apart from its function; that it is a device, an artifice.
>
> *V.*

Our goal is to develop a variant of Knuth-Bendix completion which solves the problems discussed in the previous chapter; 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. 4-1. 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 4-1: 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 6** (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 $s \rightarrow_R u$. 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 $R_\omega$ be convergent, and indeed it would be quite surprising if $R_\omega$ 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 $\gamma$ of $\mathcal{C}$ we can construct an equivalent execution $\gamma_0$ of $\mathcal{C}_0$ (i.e., with the same deductions). As noted in Lem. 4, 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 all 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 [28] and unfailing completion [3]. However, these techniques provide no means for obtaining a convergent completion. In the next chapter, we consider refinements of $\mathcal{C}_0$ which will solve the problem of distinguishability of successful branch and increase performance.

## 4.2 Many-orderered 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 (or proper class) of reduction orders $\mathcal{O}$. The rules of this system, which we call $\mathcal{C}_1$, are presented in Fig. 4-2. 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 all lexicographic path orders over the signature under consideration. More

generally, we are free here to consider the class 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 class of all orders: $\mathcal{C} \sqsubseteq C_1 \sqsubseteq \mathcal{C}_0$.

| | | |
|---|---|---|
| ORIENT: | $\dfrac{(E \cup \{s \stackrel{\cdot}{\approx} t\}, R)}{(E, R \cup \{s \to t\})}$ | 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: | $\dfrac{(E \cup \{s \stackrel{\cdot}{\approx} t\}, R)}{(E \cup \{u \stackrel{\cdot}{\approx} t\}, R)}$ | 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 $s \stackrel{\sqsupset}{\to}_R v$ |

Figure 4-2: 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. 6.

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) [20] 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 [13, 14]. In 1994, however, Andrea Sattler-Klein closed it with complicated examples that demonstrated incorrectness [31]. In particular, it was shown that there exist theories for which $R_\omega$ 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 $\mathcal{C}_1 - \{\text{COLLAPSE}, \text{COMPOSE}\}$ 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 $R_\omega$ is terminating. Since the execution is fair, all nontrivial critical pairs of $R_\omega$ have been added, and so it is also confluent. Still, infinite executions of the restricted system can result in a non-confluent $R_\omega$ — an important caveat, because it implies that even the system $\mathcal{C}_1 - \{\text{COLLAPSE}, \text{COMPOSE}\}$ 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 $\mathcal{C}_1 - \{\text{COLLAPSE}, \text{COMPOSE}\}$ 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, then 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. And second, it is not clear that such a scheme would be useful as a semidecision procedure. Indeed, the basis of the scheme, $\mathcal{C}_1 - \{\textsc{collapse}, \textsc{compose}\}$, is not a correct semidecision procedure. In the next chapter, we address these issues.

# Chapter 5

# Knuth-Bendix Completion with Modern Termination Checking

> But it was a neat theory, and he was in love with it. The only consolation he drew from the present chaos was that his theory managed to explain it.
>
> *V.*

We now present a modification of the standard Knuth-Bendix completion procedure. The primary difference is that no reduction order is explicitly 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 well-founded 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 Chap. 6, we discuss how this test is accomplished in practice.

Figure 5-1 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 $\alpha$ 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 $\alpha$ of $\mathcal{A}$ is *equivalent* to an execution $\gamma$ of $\mathcal{C}$ when the intermediate equations and rewriting systems are the same at each step. A execution $\alpha$ of system $\mathcal{A}$ succeeds when $E_{|\alpha|} = \emptyset$ and $R_{|\alpha|}$ is a convergent rewriting system equivalent to $E$.

| | | |
|---|---|---|
| ORIENT: | $\dfrac{(E \cup \{s \stackrel{\cdot}{\approx} t\}, R, C)}{(E, R \cup \{s \to t\}, C \cup \{s \to t\})}$ | if $C \cup \{s \to t\}$ terminates |
| DEDUCE: | $\dfrac{(E, R, C)}{(E \cup \{s \approx t\}, R, C)}$ | if $s \leftarrow_R u \rightarrow_R t$ |
| DELETE: | $\dfrac{(E \cup \{s \approx s\}, R, C)}{(E, R, C)}$ | |
| SIMPLIFY: | $\dfrac{(E \cup \{s \stackrel{\cdot}{\approx} t\}, R, C)}{(E \cup \{u \stackrel{\cdot}{\approx} t\}, R, C)}$ | 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 $s \stackrel{\sqsupset}{\to}_R v$ |

Figure 5-1: Modified Knuth-Bendix Completion ($\mathcal{A}$)

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 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. 4; i.e., that it is sound, that the resulting rewriting system is convergent, and finally 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. 6. 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 it here for instructive reasons only, as it illustrates nicely 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 $\alpha$ 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 5.** *Let $\alpha$ 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 $\alpha$ 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 $\alpha$ 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 $\alpha$ 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 $\alpha$ 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 6.** *Let $\alpha$ 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. 5. □

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. This was an open problem in the rewriting community [14], eventually solved by Sattler-Klein [31]. A single reduction order also allows the convenience of proving partial correctness by simulation of a standard completion procedure.

**Theorem 7** ($\mathcal{A} \sqsubseteq \mathcal{C}$)**.** *Let $\alpha$ be a finite execution of the system $\mathcal{A}$. Then there exists an equivalent execution $\gamma$ of $\mathcal{C}$ using reduction order $\succ_{|\alpha|}$.*

*Proof.* By induction on $|\alpha|$. We construct an execution $\gamma$ 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, $\gamma'$ satisfies the claim for $\alpha'$. We show $\gamma$ is a valid execution given the inference rules of $\mathcal{C}$. If the final deduction of $\alpha$ results from any of the rules of $\mathcal{A}$ other than ORIENT, then the deduction is valid in $\gamma$ 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 6 shows that $s \succ_{|\alpha|} t$ for all $s \to t \in R_k$, so ORIENT is a valid deduction in $\gamma$. □

**Theorem 8** (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. 6, the soundness of the unordered system $\mathcal{C}_0$. The second and third parts follow from Thm. 7, an equivalent execution of $\mathcal{C}$ is constructable from $\alpha$ 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 well-founded 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.

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 $R_j = \cup_{0 \leq i \leq j} \{f g^i f \rightarrow f g^{i+1}\}$. 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 section, 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 9** ($\mathcal{C} \sqsubseteq \mathcal{A}$)**.** *For any valid execution $\gamma$ of $\mathcal{C}$ with reduction order $>$, there exists an equivalent valid execution $\alpha$ 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, $\alpha$ is valid, and also $\emptyset \subseteq >$.

- In the step case, $\gamma$ has length $k+1$, $\gamma_k = (E_k, R_k)$ and extends $\gamma'$ of length $k$. By IH there exists an execution $\alpha'$ that satisfies the claim for $\gamma'$, 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 $\alpha$ extend $\alpha'$ such that $\alpha_k = (E_k, R_k, C_k)$. This is clearly equivalent to $\gamma$, 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 gamma is a valid extension of the executions $\gamma_\lambda$ for all $\lambda < |\gamma|$. By IH there exist valid executions $\alpha'$ of $\mathcal{A}$ for all $\lambda < |\gamma|$, and also that $R_\lambda \subseteq >$ and $C_\lambda \subseteq >$. Let $\alpha$ be an extension of all $\alpha'$ as above such that $\alpha_\lambda(\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa \le \iota < |\gamma|} E_\iota, \bigcup_{\kappa<|\gamma|} \bigcap_{\kappa \le \iota < |\gamma|} R_\iota, \bigcup_{\iota<|\gamma|} C_\iota)$. By definition of the persistent sets, $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa \le \iota < |\gamma|} E_\iota := E_\omega$ and $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa \le \iota < |\gamma|} R_\iota := R_\omega$, so $\alpha$ is equivalent to $\gamma$. For $\iota < |\gamma|$ we have $R_\iota \subseteq >$ and $C_\iota \subseteq$ by IH. It follows easily that that $\bigcup_{\kappa<|\gamma|} \bigcap_{\kappa \le \iota < |\gamma|} R_\iota \subseteq >$ 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 by Thm. 3 that the rewriting systems $R$ and constraint systems $C$ 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 of the thesis, correctness of the system $\mathcal{A}$, by leveraging the fact that $\mathcal{C} \sqsubseteq \mathcal{A}$.

**Theorem 10** (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 $\alpha$ 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 $R_\omega$ 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. 6 and that $\mathcal{A} \sqsubseteq \mathcal{C}$. For the other parts, consider that by Thm. 9 for every fair, non-failing execution $\gamma$ of the system $\mathcal{C}$ there exists an equivalent execution $\alpha$ of the system $\mathcal{A}$. Because the identities and rewriting systems of the execution $\alpha$ are the same as those in $\gamma$, it follows that $\alpha$ satisfies that second two properties of the theorem.

Thm. 9 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 $\gamma$ 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 $R_\omega$ of $\gamma$ because, as Thm. 9 demonstrates, some branch of the tree is exactly $\alpha$, and $\alpha$ is equivalent to $\gamma$. □

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 $\alpha$ equivalent to $\gamma$, then we would not have to be concerned about termination, as Thm. 9 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 Chap. 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. 9, and note that an invariant of the execution $\alpha$ 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. 9 is provable and the equivalent execution $\alpha$ 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. 5. 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.

# Chapter 6

# Implementing Completion with a Termination Checker

> A kind of tapestry ... spilled out the slit windows and into a void, seeking hopelessly to fill the void: for all the other buildings and creatures, all the waves, ships and forests of the earth were contained in this tapestry, and the tapestry was the world.
>
> *The Crying of Lot 49*

Our modified Knuth-Bendix completion procedure is implemented in a 7000-line Ocaml program called SLOTHROP.[1] Theories are read from files in the standard format of the TPTP (Thousands of Problems for Theorem Provers) project [35]. The source grew from an older implementation of the standard completion procedure by Franz Baader and Tobias Nipkow [1] and makes use of data structures programmed by Jean-Cristophe Fillâtre [16]. The strategy for executing the completion procedure in SLOTHROP was developed in 1981 by Girard Huet [19], and is commonly referred to as Huet's algorithm. Ten years later, Bachmair used the inference system $\mathcal{C}$ [2] to prove Huet's algorithm correct.

[1] SLOTHROP is available online at `cl.cse.wustl.edu`.

## 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 AProVE

As discussed in Chap. 2, determining whether or not a term rewriting system terminates is undecidable in general. However, modern termination-checking tools, such as AProVE [17], succeed in proving many systems terminating or nonterminating with almost alarming success. In our implementation, we take advantage of this success and use AProVE as an oracle to answer queries about the termination of constraint rewriting systems in each orientation step. If AProVE fails to prove a system terminating or nonterminating, we treat it as a nonterminating system. However, the array of techniques used by AProVE 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 AProVE is able to prove termination of systems that are not compatible with a path order, Slothrop can find convergent completions of theories other modern completion

tools (e.g., WALDMEISTER [24]) cannot. One example of such a theory is given in Chap. 7.

Integrating a separate termination checker also provides separation-of-concerns benefits for theorem proving. As the power and speed of the AProVE tool, 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 AProVE 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 all 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.

## 6.4 Limitations

Even the refined heuristic described above leaves something to be desired in terms of performance. While it has been successful in completing systems of modest size (described in Chap. 7), performance is certainly an issue for larger systems. This is, to some extent, the nature of working on undecidable problems, but other tools have set impressive benchmarks that Slothrop cannot generally match.

We have done some experimentation with different heuristics, most of which show more promise on some theories and less on others. A variation on the above heuristic that factors the size of only non-trivial critical pairs appears occasionally advantageous. We have also 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 space of orientations.

Unfortunately, the heuristics used by Slothrop to search for completions are just that: heuristic. Decisions are made based on semi-educated guesses, and not on solid principals. This limitation remains a roadblock for Slothrop's progress, and any other implementation of the algorithm. An answer to the question of what makes a partial completion similar to a final completion would, we believe, yield great progress.

Another issue is that in some cases AProVE is unable to prove termination of a system with either orientation of a particular identity. Here, we do not discard the system entirely, but attempt to orient other identities in hope that the previously unorientable identity will simplify into an orientable (or trivial) one. In the current implementation of Slothrop, treatment of such systems is delayed until others are explored that can be proved terminating or nonterminating. Again, this heuristic has proved tentatively suitable for simple systems, but a better grasp of the problem is needed to deal with difficult theories efficiently.

# Chapter 7

# Results and Performance

> Not outward, into the simple Mysteries of an open Sea, but inward, — branching, narrowing, compressing towards an Enigma as opaque and perilous as any in my Travels.
>
> *Mason & Dixon*

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 under three seconds on a modern desktop PC. On the way to this completion, it encounters 27 orientations, roughly half of which are not trivially nonterminating and must be verified with APROVE. On the execution branch that leads to a completion, however, only two orientation steps are required. SLOTHROP automatically completes the theory of groups plus a single endomorphism ($GE_1$, shown in Fig. 7-1) in under ten seconds, requiring about a hundred calls to APROVE. The completion discovered is shown in Fig. 7-2.

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

Figure 7-1: The Theory of One Group Endomorphism ($GE_1$)

A large theory with 21 identities corresponding to propositional proof simplification rules [38] is considerably more difficult to complete because of the number of orientations. Nonetheless, SLOTHROP does find a completion without user intervention after about seven hours and three thousand calls to APROVE.

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

Figure 7-2: Convergent Completion of $GE_1$

The majority of SLOTHROP's running time is spent waiting for calls to APROVE. Although we have encountered many examples of rewriting systems which APROVE can show terminating after a prohibitively 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 APROVE that occur on successful branches return in under 2 seconds. Figure 7-3 shows the time for each call to APROVE while completing $GE_1$, in which most calls require fewer than 0.25 seconds and all fewer than 0.5 seconds. Completeness of SLOTHROP can be exchanged for performance enhancements by calling APROVE with a short timeout. The above completions were obtained with a 5-second timeout.

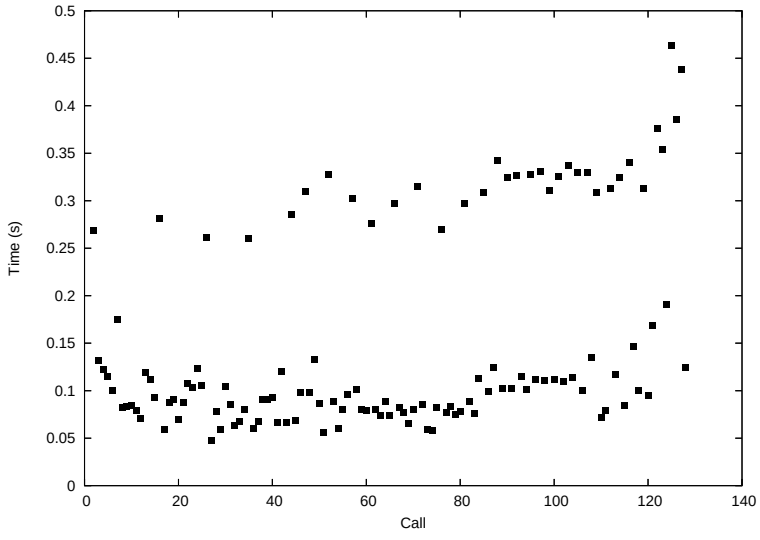


Figure 7-3: Time in APROVE Completing $GE_1$

## 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 are oriented differently depending on the context.

Shown in Fig. 7-4 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. 7-4 are completions of $GE_1$.

| | | |
|---|---|---|
| $x * 1 \to x$ | $x * x^{-1} \to 1$ | $1^{-1} \to 1$ |
| $1 * x \to x$ | $x^{-1} * x \to 1$ | $h(1) \to 1$ |
| $(x^{-1})^{-1} \to x$ | $h(x)^{-1} \to h(x^{-1})$ | $(x * y)^{-1} \to x^{-1} * y^{-1}$ |
| $(x * h^{k+1}(y)) * h^{k+1}(z) \to x * h^{k+1}(y * z)$ | | |
| $(x * h^i(y)) * h^i(y^{-1}) \to x$ | | 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 7-4: 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. 7-5. The convergent completion is shown in Fig. 7-6. 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 for the first time by hand [33] — rules derived from critical pairs were manually oriented, local confluence checked,

and termination of the resulting system verified by AProVE. This is a completely new result and we consider it to be Slothrop's defining achievement.

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

Figure 7-5: 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 7-6: Convergent Completion of $CGE_2$

Using unfailing completion [3], 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 [38]. Furthermore, Waldmeister does not appear to be able to find this ground-convergent completion fully automatically; a carefully selected Knuth-Bendix order (given in [33]) must be provided. Slothrop is able to find the convergent completion with no input from the user other than the theory itself. (This still takes more than an hour, however, even using the heuristic described in Chap. 6.)

# Chapter 8

# Conclusion

> We have to find meters whose scales are unknown in the world, draw our own schematics, getting feedback, making connections, reducing the error, trying to learn the real function ... zeroing in on what incalculable plot?
>
> *Gravity's Rainbow*

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 orderings. 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 Related Work

Another approach to completion, developed by Bachmair along with Nachum Dershowitz and David Plaisted, is called *unfailing completion* [3]. The basis of unfailing completion is that reduction orders are *total* for ground terms (i.e., any two terms without variables are comparable), but not in general (e.g., no reduction order can orient $x \approx y$). Unfailing completion also leverages the observation that, when using the procedure as a semidecision procedure to solve the word problem and decide whether two terms are identified, the two terms in question can always be ground

terms (made so by replacing variables with fresh constants). In an unfailing completion procedure then, the user provides a reduction order, but if at some point in the execution an identity cannot be oriented, the execution does not fail. Instead, the unorientable identity is remained. These identities are used just like rewrite rules in normalizing the terms for which we wish to decide identity. It is possible to use these unoriented identities for normalizing terms because when instantiated for ground terms, the reduction order provided is total, and so is guaranteed to have some 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; confluent and terminating for 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 completions [34, 38]. 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 a special technique for proving termination called *dependency pairs* with Knuth-Bendix completion [26]. This work is limited to the use of dependency pairs technique, and they also mention as future work the difficulty of finding suitable orders:

> We have seen that the use of the dependency pair criterion allows the use of a wider class of term orderings, but it means that it is getting more difficult to build systems that will try to *find* a suitable ordering.

Very recently, Pierre LeScanne suggested by personal communication that some work has been done similar to ours in the early 1980's. We have not yet been able to verify this, but include citations to his suggestions for posterity [15, 22, 23].

## 8.2 Future Work

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. The performance of SLOTHROP also does not approach that of well-tuned equational theorem provers such as WALDMEISTER for most tasks. 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.

# References

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

[2] L. Bachmair. *Canonical Equational Proofs*. Progress in Theoretical Computer Science. Birkhäuser, 1991.

[3] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion Without Failure. In *Resolution of Equations in Algebraic Structures*, volume 2, Rewriting Techniques, pages 1–30. Academic Press, 1989.

[4] Leo Bachmair, Ashish Tiwari, and Laurent Vigneron. Abstract congruence closure. *J. of Automated Reasoning*, 31(2):129–168, 2003.

[5] Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Proceedings of the 16th International Conference on Computer Aided Verification*, 2004.

[6] W. W. Boone. The Word Problem. *Proceedings of the National Academy of Science*, 44:1061–1065, October 1958.

[7] R. S. Boyer and J S. Moore. A theorem prover for a computational logic. In M. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449, pages 1–15. Springer-Verlag, 1990.

[8] Hubert Comon and Ralf Treinen. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1–2):67–87, April 1997.

[9] M. Davis. Hilbert's tenth problem is unsolvable. *American Mathematical Monthly*, 80:233–269, 1973.

[10] N. Dershowitz and D.A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[11] Nachum Dershowitz. Open. Closed. Open. In Jürgen Giesl, editor, *16th International Conference on Rewriting Techniques*, volume 3467 of *Lecture Notes in Computer Science*, Nara, Japan, April 2005. Springer-Verlag.

[12] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. 1990.

[13] Nachum Dershowitz, Jean-Pierre Jouannaud, and Jan Willem Klop. Open problems in rewriting. In R. Book, editor, *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications (Como, Italy)*, volume 488, pages 445–456, Berlin, April 1991. Springer-Verlag.

[14] Nachum Dershowitz and Ralf Treinen. The RTA list of open problems. `http://www.lsv.ens-cachan.fr/rtaloop/`.

[15] David Detlefs and Randy Forgaard. A procedure for automatically proving the termination of a set of rewrite rules. In Jean-Pierre Jouannaud, editor, *Proceedings of the First International Conference on Rewriting Techniques and Applications (University of Dijon, France)*, volume 202, pages 255–270, Berlin, May 1985. Springer-Verlag.

[16] Jean-Christophe Filliâtre. Ocaml data structures. Available at `http://www.lri.fr/~filliatr/software.en.html`.

[17] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated Termination Proofs with AProVE. In V. van Oostrom, editor, *the 15th International Conference on Rewriting Techniques and Applications*, pages 210–220. Springer, 2004.

[18] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

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

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

[21] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[22] P. Lescanne. Implementation of completion by transition rules + control: ORME. In Hélène Kirchner and W. Wechler, editors, *Proceedings 2nd International Conference on Algebraic and Logic Programming, Nancy (France)*, volume 463 of *Lecture Notes in Computer Science*, pages 262–269. Springer-Verlag, 1990.

[23] P. Lescanne. ORME, an implementation of completion procedures as sets of transitions rules. In M. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, pages 661–662. Springer-Verlag, 1990.

[24] B. Löchner and T. Hillenbrand. The Next Waldmeister Loop. In A. Voronkov, editor, *18th International Conference on Automated Deduction*, pages 486–500, 2002.

[25] Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations – part I: untimed systems. In *135*, page 35. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1993.

[26] Claude Marché and Xavier Urbain. Termination of associative-commutative rewriting by dependency pairs. In *RTA*, pages 241–255, 1998.

[27] A. Newell, J. C. Shaw, and H. A. Simon. Empirical Explorations of the Logic Theory Machine. In *Proceedings of the Western Joint Computer Conference*, pages 218–239, 1957.

[28] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[29] Robert Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47(2):65–69, 1993.

[30] J. Alan Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[31] A. Sattler-Klein. About Changing the Ordering During Knuth-Bendix Completion. In P. Enjalbert E. W. Mayr and K. W. Wagner, editors, *Symposium on Theoretical Aspects of Computer Science*, volume 775 of *LNCS*, pages 176–186. Springer, 1994.

[32] N. Shankar. Little engines of proof. In L.-H. Eriksson and P. Lindsay, editors, *Formal Methods - Getting IT Right*, pages 1–20. Springer-Verlag, 2002.

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

[34] A. Stump and L.-Y. Tan. The Algebra of Equality Proofs. In Jürgen Giesl, editor, *16th International Conference on Rewriting Techniques and Applications*, pages 469–483. Springer, 2005.

[35] Christian B. Suttner and Geoff Sutcliffe. The TPTP problem library. Technical Report JCU-CS-96/9, 18 1996.

[36] Li-Yang Tan. The Meta-Theory of $Q_0$ in the Calculus of Inductive Constructions. Technical Report WUCSE-2006-24, Washington University in Saint Louis, May 2006.

[37] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, 2003.

[38] Ian Wehrman and Aaron Stump. Mining Propositional Simplification Proofs for Small Validating Clauses. In A. Armando and A. Cimatti, editors, *3rd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning*, 2005.

[39] Ian Wehrman, Aaron Stump, and Edwin Westbrook. SLOTHROP: Knuth-Bendix Completion with a Modern Termination Checker. In Frank Pfenning, editor, *17th International Conference on Rewriting Techniques and Applications*, 2006. To appear.

# Vita

Ian A. Wehrman

**Date of Birth** June 7, 1980

**Place of Birth** St. Louis, Missouri

**Degrees**

B.Sc. Computer Science, May 2004
Webster University, St. Louis, Missouri

M.Sc. Computer Science, August 2006
Washington University in St. Louis, St. Louis, Missouri

**Honors**

Math & Computer Science Department Honors, 2004
Webster University, St. Louis, Missouri

Distinguished Master's Fellowship, 2004 – 2005
Washington University in St. Louis, St. Louis, Missouri

Student Travel Grant, 2006
The Fourth Federated Logic Conference, Seattle, Washington

Microelectronics and Computer Development Doctoral Fellowship, 2006
The University of Texas, Austin, Texas

College of Natural Sciences Dean's Excellence Award, 2006
The University of Texas, Austin, Texas

**Publications**

Ian Wehrman, Aaron Stump and Edwin Westbrook. *Slothrop: Knuth-Bendix Completion with a Modern Termination Checker.* 17th International Conference on Rewriting Techniques and Applications.

Aaron Stump and Ian Wehrman. *Property Types: Semantic Programming for Java.* 13th International Workshop on Foundations and Developments of Object-Oriented Languages.

Edwin Westbrook, Aaron Stump and Ian Wehrman. *A Language-based Approach to Functionally Correct Imperative Programming.* 10th ACM SIGPLAN International Conference on Functional Programming, pp. 268-279.

Ian Wehrman and Aaron Stump. *Mining Propositional Simplification Proofs for Small Validating Clauses.* Third Workshop on Pragmatics of Decision Procedures in Automated Reasoning. ENTCS, Volume 144, Issue 2, 19 January 2006, pp. 79-91.

**Tech. Reports**

Ian Wehrman, Sajeeva Pallemulle and Kenneth Goldman. *Extending Byzantine Fault Tolerance to Replicated Clients.* WUCSE-2006-7. February, 2006.

Edwin Westbrook, Aaron Stump and Ian Wehrman. *A Language-based Approach to Functionally Correct Imperative Programming.* WUCSE-2005-32. July, 2005.

August 2006

Short Title: Completion with Termination Checking  Wehrman, M.Sc. 2006