

PROOF DIAGRAMS AND TERM REWRITING WITH
APPLICATIONS TO COMPUTATIONAL ALGEBRA

Duncan Shand

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



1997

Full metadata for this item is available in
St Andrews Research Repository
at:
<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:
<http://hdl.handle.net/10023/13498>

This item is protected by original copyright

PROOF DIAGRAMS AND TERM REWRITING WITH APPLICATIONS TO COMPUTATIONAL ALGEBRA



A thesis submitted to the
UNIVERSITY OF ST ANDREWS
for the degree of
DOCTOR OF PHILOSOPHY

By
Duncan Shand

School of Mathematical and Computational Sciences
University of St Andrews

October 1996



ProQuest Number: 10167126

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10167126

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

TR C178

I, Duncan Shand, hereby certify that this thesis, which is approximately 70,000 words in length, has been written by me, that it is the record of work carried out by me, and that it has not been submitted in any previous application for a higher degree.

date 14/2/97 signature of candidate _____

I was admitted as a research student in October 1993 and as a candidate for the degree of Doctor of Philosophy in October 1994; the higher study for which this is a record was carried out in the University of St Andrews between 1993 and 1996.

date 14/2/97 signature of candidate _____

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Doctor of Philosophy in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

date _____ signature of supervisor _____

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker.

date 14/2/97 signature of candidate _____

Abstract

In this thesis lessons learned from the use of computer algebra systems and machine assisted theorem provers are developed in order to give an insight into both the problems and their solutions.

Many algorithms in computational algebra and automated deduction (for example Gröbner basis computations and Knuth-Bendix completion) tend to produce redundant facts and can contain more than one proof of any particular fact. This thesis introduces *proof diagrams* in order to compare and contrast the proofs of facts which such procedures generate. Proof diagrams make it possible to analyse the effect of heuristics which can be used to guide implementations of such algorithms.

An extended version of an inference system for Knuth-Bendix completion is introduced. It is possible to see that this extension characterises the applicability of critical pair criteria, which are heuristics used in completion. We investigate a number of executions of a completion procedure by analysing the associated proof diagrams. This leads to a better understanding of the heuristics used to control these examples.

Derived rules of inference are also investigated in this thesis. This is done in the formalism of proof diagrams. Rewrite rules for proof diagrams are defined: this is motivated by the notion of a transformation tactic in the Nuprl proof development system. A method to automatically extract ‘useful’ derived inference rules is also discussed.

‘Off the shelf’ theorem provers, such as the Larch Prover and Otter, are compared to specialised programs from computational group theory. This analysis makes it possible to see where methods from automated deduction can improve on the tools which group theorists currently use. Problems which can be attacked with theorem provers but not with currently used specialised programs are also indicated.

Tietze transformations, from group theory, are discussed. This makes it possible to link ideas used in Knuth-Bendix completion programs and group presentation simplification programs. Tietze transformations provide heuristics for more efficient and effective implementations of these programs.

Acknowledgements

I would especially like to thank my supervisor, Professor Ursula Martin, for her support and encouragement over the last three years, and for providing an inspiring research environment. I am also indebted to Dr Steve Linton and Dr Simon Brock for their advice and patience. I also owe thanks to Dominic Tulley for many interesting discussions.

Contents

1	Introduction	1
1.1	Computational Mathematics	2
1.1.1	Computer Algebra	3
1.1.2	Automated Deduction	4
1.1.3	Search in Automated Deduction	7
1.2	Layout of Thesis	9
2	Homeomorphic Embedding	11
2.1	Introduction	11
2.2	A Proof of Kruskal's Tree Theorem	12
2.2.1	Orderings and Trees	12
2.2.2	The Homeomorphic Embedding Relation	14
2.2.3	Subtrees	17
2.2.4	Sequences	19
2.2.5	Some Topological Arguments	23
2.2.6	Proof of Kruskal's Tree Theorem	26
2.3	Highlights of Embedding Theorems	29
3	An Introduction to Proof Diagrams	33
3.1	Introduction	33
3.2	Graphs and a Version of Kruskal's Theorem	34
3.2.1	Graphs	34
3.2.2	Sources in Graphs	36
3.2.3	Homeomorphic Embedding for Rooted DAGs	38

3.3	Proofs of Facts as Graphs	41
3.3.1	Proof Diagrams	41
3.3.2	A System for Propositional Resolution	42
3.3.3	Proofs of facts	45
3.3.4	Proof Graphs	48
3.3.5	Further Definitions	50
3.4	Properties of Proofs of Facts	52
3.4.1	Characterising Proofs of Facts	52
3.4.2	Direct Proofs	55
3.4.3	Properties of Subproofs	56
3.5	Possible uses of Proof Diagrams	57
3.6	Kruskal's tree theorem Revisited	59
3.7	Conclusions	61
4	Completion and Proof Diagrams	62
4.1	Introduction	62
4.2	Completion	63
4.2.1	Definitions	63
4.2.2	Extended Inference Rules for Completion	67
4.2.3	Critical Pair Criteria	70
4.3	Properties of Proof Diagrams in C-Completion	72
4.3.1	A System for C-Completion and Generating Proof Diagrams	73
4.3.2	Properties of C-Completion	75
4.3.3	Orderings, C-Completion and Proof Diagrams	77

4.3.4	Representations of Proof Diagrams	80
4.4	Examples and Further Work	81
4.5	Conclusions	87
5	Derived Inferences and Proof Diagrams	88
5.1	Introduction	88
5.2	Derived Inferences	90
5.2.1	Definitions for Derived Inferences	90
5.3	Relationships Between Proof Diagrams	93
5.4	Rewriting Proof Diagrams	95
5.4.1	Rewrite Rules and Proof Diagrams	95
5.4.2	Properties of Rewriting Proof Diagrams	98
5.4.3	Finding Useful Derived Rules of Inference	100
5.5	Conclusions	103
6	Group Theory and Automated Deduction	105
6.1	Introduction	105
6.2	Mathematical Background	106
6.2.1	Groups and Presentations	106
6.2.2	Notation	108
6.2.3	The Alternative Formulation of the Problems	108
6.3	Theoretical Results	110
6.4	The Programs	110
6.4.1	Otter	110
6.4.2	Larch	111

6.4.3	String Knuth-Bendix	111
6.4.4	TC	111
6.5	Finite Groups	111
6.5.1	The Examples	112
6.6	Fibonacci groups	117
6.7	Burnside Groups	119
6.8	Nilpotent Groups	122
6.8.1	Sims' Example	122
6.8.2	Engel Groups	126
6.9	Conclusions	127
7	Tietze Transformations	128
7.1	Introduction	128
7.2	Definitions for Tietze Transformations	128
7.3	Tietze Transformations and Completion	129
7.3.1	Related Work	129
7.3.2	Generator Introduction	130
7.4	Tietze Transformations and Group Presentation Simplification	134
8	Conclusions and Future Work	139
8.1	Future Work	141
A	Example Inputs for LP and Otter	153

1 Introduction

In this thesis we formalise an abstract framework for investigating heuristics in proof procedures. To do this we introduce the notion of the proof diagram, which is a graph-like structure. We use a graph-like structure rather than a tree to convey that in some proof procedures certain facts can be generated in more than one way. For instance, in an execution of the Knuth-Bendix completion procedure or a Gröbner basis computation there is usually more than one way that each equation can be generated. We are then able to reason about global and local properties of the proof diagrams to see if there is any information we can learn for future proof attempts.

With this formalism we are able to introduce the notion of a proof P of a fact a being minimal with respect to a proof B of a . There are two criteria we would like such proofs P of a to satisfy:

- the proof of a uses no completely irrelevant information
- there is no ‘shortcut’ to get to ‘useful’ facts which we know from the proof B of a

The second criterion is characterised by the notion of ‘throwing away’ structure. If, given a number of facts, we know about an inference which avoids some complex structure, we wish to use it. Thus if we have a sequence of steps $a_1 a_2 \dots a_6 a_7 a_8$ in a proof of a fact a and we know that $a_2 a_6$ is a valid step, we would wish the proof of a to use the following sequence of steps: $a_1 a_2 a_6 a_7 a_8$.

We show that the concept of a proof P of a being minimal with respect to a proof B of a corresponds in an obvious way to notions which occur in Denzinger and Schulz [35] and Levy [81]. We also prove various properties of such proofs P of a , for example each inference derives only one fact and there are no cycles in the associated graph structure.

We are also able to model derived inferences in the formalism of proof diagrams. It is shown that the notion of a derived inference in this context is a sound extension to the system. We can then define the concept of a proof diagram rewriting rule: the motivation for this comes from the definition of a transformation tactic in Nuprl [29]. We are able to rewrite complex parts of proofs and replace them with syntactically simpler ones. We

1 INTRODUCTION

also discuss the possibility of automatically generating derived inference rules, and give an algorithm for doing so in the setting of proof diagrams.

Knuth-Bendix completion is investigated in this thesis. There are a number of choices which have to be made during any particular execution of the Knuth-Bendix completion procedure. For example, which ordering you choose and which critical pairs are calculated can have a large effect on the efficiency of an execution of the procedure. This leaves a lot of scope for different heuristic approaches. One type of heuristic used in completion is critical pair criteria (see Bachmair and Dershowitz [6]). These heuristics detect the generation of redundant equations. There are intricate methods for proving properties of such criteria, but very little is understood as to when a particular criterion should be applied. We will introduce an extension of the inference rule formalism of Bachmair [5] for completion, which characterises the use of critical pair criteria.

We also look at the application of techniques from completion to group theory. We do this in a number of ways. One area of interest is seeing how general purpose systems compare to the software which group theorists conventionally use. Two 'off the shelf' theorem provers, Otter [93] and the Larch Prover [45], are compared against an implementation of the coset enumeration algorithm and a specialised string completion program. Also, problems out of the scope of conventional group theory programs are mentioned and techniques from automated deduction for solving such problems are discussed. We also investigate how the generality of the systems can be used to represent interesting phenomena in group theory, and use these to guide the search.

Completion is also investigated in terms of Tietze transformations. It will be shown that Tietze transformations are in some sense a generalisation of ideas in completion. This gives rise to a discussion on how completion and group presentation simplification programs can be enhanced through ideas from Tietze transformations.

1.1 Computational Mathematics

Computational techniques have become increasingly important in mathematics: this can be seen by the proliferation of computer programs which address problems in mathematics and engineering. These programs can be fundamentally different in nature: for instance, there

are specialised programs for checking large search spaces (such as the proof of the four colour theorem in graph theory by Appel and Haken) and also programs which rigorously check proofs in a particular logic. This diversity may make it seem as if these programs have little in common, but a theme in most areas of computational mathematics is the feasibility of solving ‘large’ problems.

1.1.1 Computer Algebra

Computer algebra programs (such as Maple [26] and Mathematica [126]) are ubiquitous in many areas of engineering and science. They contain implementations of algorithms for performing routine symbolic computations such as factorising polynomials, solving equations and simplifying algebraic expressions.

One algorithm which is implemented in computer algebra systems is the Gröbner basis algorithm. Suppose we are given a generating set G for an ideal I in a polynomial ring over a field k and a well founded ordering on monomials. The Gröbner basis algorithm computes a generating set G' for I which can be used to reduce any $f \in I$ to the zero polynomial. Gröbner bases have many applications in commutative algebra and algebraic geometry (see, for example, Cox *et al* [33]). Some useful properties can be proved about this algorithm: given any finite generating set for the ideal and any well founded ordering on monomials it can be shown that the algorithm terminates.

The algorithm was shown to be doubly exponentially complex in Mayr and Meyer [92]. This means that even ‘small’ examples become unfeasible. Thus the problem for Gröbner basis computations is not termination but termination in a feasible time. Choosing which ideal generators to calculate and which ordering to use can have a great effect on the viability of computations, and much recent work has investigated these heuristics in a mathematical framework (see Davenport [34]).

Software has also been developed for specific problem areas in group theory. Consequently, computational group theory is a well developed subject in its own right (see, for example, Sims [112]). Specialised group theoretic programs were used in one of the greatest advances in algebra: the classification of finite simple groups. These programs were used to check large search spaces for sporadic groups (see Conway *et al* [32]). Packages such as GAP

1 INTRODUCTION

[109] and Magma [10] have also been developed and are used extensively in the group theory community. They contain implementations of many standard algorithms and provide a language so that users can easily extend the system.

There is also a crossover from work in automated deduction to the sorts of problems investigated by group theorists. The Knuth-Bendix completion procedure can sometimes construct a rewriting system which solves the word problem (i.e. it can identify when a particular word is equal to the identity element) for groups. In certain cases, for instance finite groups, it is known that a finite rewriting system exists: i.e. the completion procedure terminates. In Chapter 6 we give a family of examples of presentations of groups where each group is isomorphic to the trivial group. Computational techniques (including coset enumeration and Knuth-Bendix completion) have great difficulty in verifying this fact for even small instances of the problem.

Many pathological examples exist to show that attractive conjectures about heuristics for Knuth-Bendix completion are false. In Chapter 6 we see empirical evidence that with the use of the recursive path ordering (for the examples presented), the Knuth-Bendix completion procedure is generally quicker than with the length then lexicographic orderings. In Madlener *et al* [88] a family of rewrite systems $K_{m,n}$ is constructed, where $K_{m,n}$ is an n -generator presentation of the trivial monoid with $O(m+n)$ rules. The Knuth-Bendix completion procedure with the lexicographic recursive path ordering is shown to generate $A(m,n)$ intermediate rules (where $A(m,n)$ is Ackerman's function) in order to construct the final rule system which has $O(m+n)$ rules. This example shows that a greater understanding of the heuristics used in completion is needed.

There has been a great deal of work in investigating heuristics and data structures for Knuth-Bendix completion in the context of group theory and also for more standard computational techniques in group theory such as coset enumeration. Sims [112] gives an introduction to many of these issues.

1.1.2 Automated Deduction

The automated deduction community has shown much recent interest in computational mathematics. General purpose 'machine assisted theorem proving' environments have been

1 INTRODUCTION

designed and developed in order to investigate and prove results in this and related areas.

The basis for this is that mathematics can be formalised. This is done by expressing mathematical statements and proofs in a concise language. Any such language must have strict grammatical rules and an unambiguous semantics. The accuracy of formal manipulations of mathematical statements can be tedious for humans to analyse: there is a lot of fine detail which needs to be verified. Fortunately, computers are well suited to such tedious but completely defined tasks.

A *proof procedure* is a mechanical procedure for constructing a proof in a formal system. We paraphrase the following from Gordon and Melham [46]. *Backward* proof procedures organise the search as a tree, beginning with the objective (or goal). The goal is then decomposed, successively if necessary, into what one hopes are more tractable subgoals. The goal is said to be solved if it is reduced to the empty set of subgoals. In this thesis we will investigate a number of *forward* proof procedures which have a particular form. We can think of these proof procedures informally in the following way:

Starting with a set of assumptions, some subset of those assumptions is used to produce a conclusion which is then added to the set of assumptions. This process continues iteratively until the conclusion produced is that required to prove some theorem.

Examples of such techniques are Knuth-Bendix completion and resolution. Proof procedures can also be hybrids of forward and backward procedures.

The resolution method has a simple premise: a single inference rule, the resolution rule, is applied to a set of clauses \mathcal{S} . The resolution rule generates a clause c from \mathcal{S} , c is added to \mathcal{S} and the procedure continues iteratively. The method depends on the fact that \mathcal{S} is satisfiable if and only if $\mathcal{S} \cup \{c\}$ is satisfiable. Thus if the empty clause is generated, \mathcal{S} is unsatisfiable. Haken [51] showed that there is an exponential lower bound complexity for resolution. Restrictions of resolution which cut down the search space have been introduced (see, for example, Jäger [63]). These methods are generally not complete for arbitrary sets of clauses.

There are many areas of automated deduction where search plays an important role, and there are many systems which incorporate complex methods for controlling search. In this

1 INTRODUCTION

thesis the phrase ‘machine assisted theorem prover’ will mean any program which includes an implementation of a proof procedure, or allows the user to define one. We will now see that further distinctions can be drawn between classes of machine assisted theorem provers; in particular a number of existing projects will be discussed. There are many more machine assisted theorem provers which are worthy of mention, but it would be impractical to list them all here.

NQTHM [11] is a *mechanical theorem prover for a logic of recursive functions over finitely generated objects*. It has little user interaction and has complex heuristics which have been built up over a number of years. Proofs often have to be guided by the user suggesting lemmas and high level ‘hints’, but the heuristics in NQTHM are well developed enough so that inductions and the subgoals of inductions can be proved completely automatically. Examples of proofs carried out by NQTHM include a formalisation of Gödel’s incompleteness theorem (Shankar [111]) and the checking of the RSA public-key encryption algorithm (Boyer and Moore [12]).

Proof development systems such as Nuprl [29], Isabelle [100] and HOL [46] need a lot of user interaction, and are based on a set of primitive inference rules; the correctness of each proof is guaranteed as all proofs must eventually be justified in this underlying logic. The primitive inference rules are coded in a meta language: new functions can be written in this language which automate multiple steps in the primitive inferences. In fact, complex proof procedures can be coded in this way. These heuristics are called *tactics*, and as the tactics have primitive inferences as building blocks, it can easily be seen that each tactic is sound with respect to the underlying logic. It can be argued that having a small set of primitive inferences as a basis is inefficient for constructing proofs, but the complexity of some of the results proved with these systems belies this. For example in Nuprl a constructive proof of Higman’s lemma (Murthy [97]) has been formalised and Nipkow [99] formalises a number of Church-Rosser proofs in an implementation of Higher Order Logic in Isabelle.

It should be noted here that Isabelle is a logical framework, meaning that the meta logic of the system has been designed to define new object logics which can be studied further. HOL and Nuprl both have fixed logics.

An expansion of a proof of a theorem into the underlying primitive inferences could increase the number of proof steps, but tactics mean that these large proofs are in some sense

1 INTRODUCTION

hidden from the user. The process of finding new tactics can be seen as a similar process to defining new heuristics for other machine assisted theorem provers: they both involve finding common patterns in proofs.

Of particular interest in this thesis are systems which include an implementation of the Knuth-Bendix completion procedure: two examples are Otter [93] and the Larch Prover [45]. Otter [93] is a *resolution theorem prover*, and has performance as a primary design consideration. Otter uses a weighting system which can be used to prioritise terms and clauses. To a naive user this weighting system seems somewhat *ad hoc*, but very impressive performance results have been achieved by using these methods. Also, an elegant data structure called a *discrimination tree* has been developed. Otter is mainly designed for non-interactive use, but has a primitive interactive mode. The Larch Prover [45] (LP) is a *theorem prover for multi-sorted first order logic*. It has been used primarily for software verification, but has also been used to prove results in algebra (Martin and Lai [89] and Linton *et al* [85]). Although it is not as efficient as programs such as Otter, LP has a greater degree of user interaction and more inference mechanisms which are useful on many examples.

1.1.3 Search in Automated Deduction

It has been shown that search is a major area of interest in computational mathematics. This is obviously a rich and complex area, and there seems little likelihood that any general theory of controlling search exists.

Notwithstanding, there have been a number of recent papers in which new theorems in various areas of mathematics have been proved with the aid of machine assisted theorem provers. For instance, McCune [94] shows that the Otter system can identify and verify new single axiomatisations for groups and in [95] proves new results about cubic curves. Fujita *et al* [42] show that model checking techniques can be used in an area of combinatorics. There has also been interest shown in formalising complex theorems in mathematics. Linton *et al* [85], formalise and verify a theorem about the wreath product of two infinite cyclic groups. These advances have been made possible by both the mathematical insight of the user and the ability of the systems to guide the search.

1 INTRODUCTION

Heuristics are used to control search. Pearl [101], defines heuristics in the following way:

heuristics stand for strategies using readily accessible though loosely applicable information to control problem-solving processes in human beings and machine

Mathematicians develop an ‘armoury’ of useful heuristics as they learn new subject areas. Particular families of examples are generally tackled by specific techniques: for example, a proof of the associativity of plus for the natural numbers would generally be carried out by mathematical induction and not a diagonalisation argument.

It would be advantageous to be able to automatically generate heuristics and be able to analyse the success (and failure) of particular techniques. This has been an area of particular recent interest. Most notably, the DREAM group at Edinburgh University has developed a theory of ‘proof plans’. Bundy [19] informally defines proof plans in the following way:

A proof plan captures the common patterns of reasoning in a family of similar proofs and is used to guide the search for new proofs in this family.

The proof plans developed have been mainly in the area of inductive reasoning and their development involved looking at large numbers of proofs and identifying common patterns of reasoning. These are represented computationally. Proof plans introduce a notion of meta-level control into search. They are implemented through:

- *tactics*, programs from LCF-style theorem provers which automate tedious parts of the proof while guaranteeing soundness (more will be mentioned in Chapter 5),
- *methods* which specify tactics
- *critics* analyse failed proofs and suggest ‘patches’ to partial proofs

Denzinger and Schulz [35] introduce a method which looks for repeated patterns in Knuth-Bendix completion. In an execution of a completion procedure finding certain ‘useful’ equations can be extremely important in guiding search. For example, if we are trying to prove that a group with a presentation which has $n > 1$ generators is isomorphic to a finite cyclic group, we would wish to find equations which showed that $n - 1$ of the

1 INTRODUCTION

generators could be rewritten in terms of the other generator. For any family of examples, finding ‘good’ proofs of ‘useful’ equations can guide a completion procedure effectively. A number of evaluation functions are presented in Denzinger and Schulz [35] which learn how to generate equations which are useful in certain situations. The examples in Denzinger and Schulz [35] show significant speed increases using the methods presented.

In Levy [81] the approach is somewhat different. The irrelevance of certain facts and inferences in logical systems are investigated in an abstract framework. A space of possible definitions for irrelevance is given. Intuitively, a fact (or inference) α is *strongly irrelevant* if α can be removed from any proof of a fact ϕ and a proof of ϕ is retained. A fact (or inference) α is *weakly irrelevant* if there exist proofs of a fact ϕ in which α can be removed and a proof of ϕ is retained. In this setting, a ‘good’ heuristic is one which avoids using irrelevant (for some notion of irrelevance) facts and inferences.

1.2 Layout of Thesis

Chapter 2 discusses a mathematical characterisation of ‘throwing away structure’. The history of the ‘homeomorphic embedding theorems’ and the well quasi-order are discussed, and a proof of a restricted version of Kruskal’s tree theorem is given. This proof follows Klop [71].

In Chapter 3 we introduce proof diagrams to reason about heuristics in automated deduction. We represent proofs of facts through proof diagrams. We also introduce the notion of a proof P of a being minimal with respect to a proof B of a . Such proofs P of a have properties which correspond to ideas of good proofs in Denzinger and Schulz [35] and Levy [81]. In particular it uses the idea of ‘throwing away structure’ which was discussed in Chapter 2. Some of the results presented here appear in Shand and Brock [110].

An extension of the inference system for Knuth-Bendix completion due to Bachmair [5] is introduced in Chapter 4. This inference system holds information which can be thought of as modelling the idea of a critical pair criterion; a complex heuristic in completion. Various other heuristics for completion in the context of the formalism introduced in Chapter 3 will also be discussed.

Chapter 5 introduces a method for reasoning about derived inferences in terms of proof dia-

1 INTRODUCTION

grams. This is done in order to allow us to ‘rewrite proofs’. We also discuss the possibility of automatically finding ‘useful’ derived inferences. The motivation for this work appears in Shand and Brock [110].

A more applied approach to using automated deduction systems is discussed in Chapters 6 and 7. In particular we investigate how machine assisted theorem proving can be applied to group theory.

Chapter 6 identifies areas in which techniques in automated deduction can be applied to group theoretic problems, and compares this to standard approaches in computational group theory. Some areas in which there are no standard tools are also indicated, and possible approaches to these problems are discussed. A version of the work presented here can also be seen in Linton and Shand [83, 84].

The relationship between Tietze transformations, a standard idea in group theory, and Knuth-Bendix completion is investigated in Chapter 7. Possible enhancements to completion programs and group presentation simplification programs are discussed, and some preliminary results in applying these ideas are indicated. This work expands on ideas in Linton *et al* [85].

We conclude in Chapter 8. Some areas of possible future work are also discussed.

2 Homeomorphic Embedding

2.1 Introduction

In this chapter we develop a version of Kruskal's tree theorem. This theorem states that given an infinite sequence of trees $t = (t_1 t_2 \dots)$ there is $i < j$ such that $t_i \sqsubseteq_T t_j$, where \sqsubseteq_T is the homeomorphic embedding relation on trees which will be defined in the chapter. There are many similar definitions for and statements of this theorem. Indeed, the title of the paper (Kruskal [75]) calls it a *frequently rediscovered concept*. Gallier [43] presents a number of equivalent definitions for trees, the homeomorphic embedding relation on trees etc. We prove a restricted version of the theorem which appears in Klop [71]: the restriction is that the trees have a bounded branching order; i.e. each node in a tree has a bounded number of 'successor' nodes. A terse outline of a proof appears in Klop [71], and we expand this to a full proof.

We choose to prove the restricted version of Kruskal's tree theorem which appears in Klop [71] as it most intuitively captures the notion of *throwing away structure* in trees. This relates back to the discussion on heuristics in Chapter 1, as we wish to characterise 'short cuts through proofs'.

In fact, there are many versions of 'homeomorphic embedding theorems' for different structures. Higman [59] proved a version for strings and Robertson and Seymour [105] prove a version for graphs: this is called the *graph minor theorem*. We do not use Robertson and Seymour's work in this thesis as the relation on graphs is different from that on trees (an example of this will be given in Section 2.3) and also because there is no version of the graph minor theorem for labelled graphs.

Section 2.2 will prove Kruskal's tree theorem in the notation presented here, and will fill in the gaps from the proof which occurs in Klop [71]. Necessary notation for the rest of the thesis is also introduced in Section 2.2. In Section 2.3 the history of 'homeomorphic embedding relations' will be investigated and some results from these investigations will be stated.

2.2 A Proof of Kruskal's Tree Theorem

In this section some basic definitions of relations, orderings and trees are given. Also, a version of Kruskal's tree theorem is developed. The definitions presented here are standard, and can be found in, for example, Green [48].

2.2.1 Orderings and Trees

Definition 2.1 Let A be a set. $\preceq \subseteq A \times A$ is said to be a relation on A .

Definition 2.2 A relation \preceq on a set A is said to be:

- reflexive if $(\forall x \in A)(x \preceq x)$
- irreflexive if $(\forall x \in A)(\neg(x \preceq x))$
- transitive if $(\forall x, y, z \in A)((x \preceq y) \wedge (y \preceq z) \Rightarrow (x \preceq z))$
- antisymmetric if $(\forall x, y \in A)((x \preceq y) \wedge (y \preceq x) \Rightarrow (x = y))$
- symmetric if $(\forall x, y \in A)((x \preceq y) \Rightarrow (y \preceq x))$

Definition 2.3 Let A be a set.

- A *preorder* on A is a reflexive, transitive relation on A ,
- A *partial order* on A is a reflexive, antisymmetric, transitive relation on A ,
- A *strict partial order* on A is an irreflexive, transitive relation on A
- A *total order* \preceq on A is a reflexive, transitive, antisymmetric relation on A in which $(\forall x, y \in A)(x \preceq y \vee y \preceq x)$

Definition 2.4 Let \preceq be a preorder on A and suppose $x, y \in A$:

- the relation $\succeq \subseteq A \times A$ is defined by $x \succeq y$ if and only if $y \preceq x$,
- the relation $\prec \subseteq A \times A$ is defined by $x \prec y$ if and only if $x \preceq y$ and $y \not\preceq x$

2 HOMEOMORPHIC EMBEDDING

- the relation $\succ \subseteq A \times A$ is defined by $x \succ y$ if and only if $y \prec x$ and
- x and y are said to be *incomparable* if neither $x \preceq y$ nor $x \succeq y$. This is denoted $x \mid y$

The following definition of tree comes from Klop [71]. We use this definition as we will prove a version of Kruskal's tree theorem which appears in Klop [71]. There are many definitions of tree (see, for instance, Gallier [43]) for which Kruskal's theorem can be proved, but Klop's version makes the notion of 'throwing away structure' intuitive. A version of Kruskal's tree theorem in Gallier [43] is for ordered trees; i.e. trees in which the order of the successor nodes of a node is important (the successor nodes of a node can be thought of as nodes which are minimally less in the partial order on the nodes of the tree). The definition of tree here does not differentiate between successor nodes. If the order of the successor nodes is fixed in some way, then the definition of tree in Gallier [43] and the one below are equivalent.

Definition 2.5 A *tree* is a pair $t = (\langle D, \leq, \rho \rangle, L)$ where D is a non-empty finite set, (called the nodes of t), ρ is a distinguished element of D (called the root), and \leq is a partial order on D satisfying the following conditions:

1. $(\forall b \in D)(b \leq \rho)$
2. $(\forall b, c, d \in D)((b \leq c \wedge b \leq d) \Rightarrow (c \leq d \vee d \leq c))$

L is a map $L : D \rightarrow \mathbb{N}$ which assigns labels (natural numbers) to the nodes of t .

Let \mathcal{T} denote the set of all trees.

The next two definitions and lemma show a necessary consequence of the partial order on the nodes of a tree. This condition will be exploited when a definition of the homeomorphic embedding relation is given.

Definition 2.6 Let $t = (\langle D, \leq, \rho \rangle, L)$ be a tree and suppose that $S \subseteq D$. $b \in D$ is an *upper bound* for S if $\forall s \in S, s \leq b$. $UB(S, t)$ denotes the set of all upper bounds for S in t .

Definition 2.7 Let $t = (\langle D, \leq, \rho \rangle, L)$ be a tree. $b \in D$ is called a *supremum* of a set $S \subseteq D$ if and only if $b \in UB(S, t)$ and $\forall c \in UB(S, t), c \leq b \Rightarrow c = b$.

2 HOMEOMORPHIC EMBEDDING

Lemma 2.8 Let $t = (\langle D, \leq, \rho \rangle, L)$ be a tree. Then for $S \subseteq D$, there is a unique supremum of S .

Proof– There is at least one upper bound of S as the root is defined to have the property that $(\forall d \in D)(d \leq \rho)$. The set $UB(S, t)$ is certainly finite as it is a subset of D , and there is thus at least one smallest element with respect to the partial order \leq . Therefore a supremum exists. Now suppose a supremum was not unique; then there exist $x, y \in UB(S, t)$ for which there is no $z_1, z_2 \in UB(S, t)$ where $(z_1 \neq y) z_1 \leq y$ or $(z_2 \neq x) z_2 \leq x$. As $x, y \in UB(S, t), \forall c \in S c \leq x$ and $c \leq y$; thus by condition 2 of Definition 2.5 $x \leq y$ or $y \leq x$. Either case yields $x = y$. So for $S \subseteq D$, the supremum of S is a unique node. \square

We will now denote the unique supremum of the set $\{d, b\}$ by $d \vee b$.

2.2.2 The Homeomorphic Embedding Relation

In this section the homeomorphic embedding relation on trees will be defined as in Klop [71]. It will also be shown that the relation is a preorder on the set of trees.

Definition 2.9 Let $s = (\langle D_0, \leq_0, \rho_0 \rangle, L_0)$ and $t = (\langle D_1, \leq_1, \rho_1 \rangle, L_1)$ be trees and let $f : D_0 \rightarrow D_1$. f is said to be *sup preserving* if $\forall a, b \in D_0, f(a \vee b) = f(a) \vee f(b)$.

Definition 2.10 Let $s = (\langle D_0, \leq_0, \rho_0 \rangle, L_0)$ and $t = (\langle D_1, \leq_1, \rho_1 \rangle, L_1)$ be trees and let $f : D_0 \rightarrow D_1$. f is said to be *label increasing* if $(\forall a \in D_0) L_0(a) \leq L_1(f(a))$, where \leq is the usual ordering on the natural numbers.

Definition 2.11 Let $s = (\langle D_0, \leq_0, \rho_0 \rangle, L_0), t = (\langle D_1, \leq_1, \rho_1 \rangle, L_1) \in \mathcal{T}$ be finite trees. s is said to be *homeomorphically embedded* in t , ($s \sqsubseteq_{\mathcal{T}} t$), if $\exists \pi : D_0 \rightarrow D_1$ such that:

- π is injective
- π is monotonic
- π is sup preserving
- π is label increasing

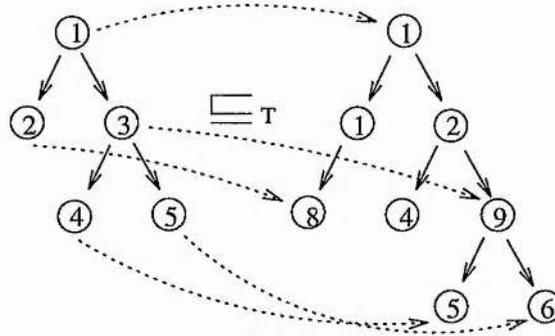


Figure 1: An Example of the Homeomorphic Embedding Relation

The relation $\sqsubseteq_{\mathcal{T}}$ is called *the homeomorphic embedding relation on \mathcal{T}* .

There is a recursive definition of the homeomorphic embedding relation on trees which is given in papers such as Gallier [43]. The above definition is different from a version in Gallier [43] as this definition works for unordered trees; i.e. the order of the successor nodes of a node is unimportant. It should be noted that the notion of homeomorphic embedding for trees in Gallier [43] has similar properties to the definition above. In particular, Gallier [43] has the monotonicity, label increasing and sup preserving properties mentioned in Definition 2.11.

In Figure 1 an example of the homeomorphic embedding relation can be seen. We claim that t_1 can be obtained by a ‘controlled throwing away’ of structure from t_2 . The nodes of t_1 can be thought of as a subset of the nodes of t_2 in which information on the partial order of the nodes is preserved. However, this is certainly not an arbitrary subset of the nodes of t_2 , as the notion of homeomorphic embedding preserves more structure than this. We can see from Figure 1 that the throwing away of structure also preserves information about common ancestors of nodes. This means we can not arbitrarily ‘throw away’ common ancestors of nodes, as the sup of two nodes is preserved in the mapping. For example, as the nodes which are labelled 4 and 5 in t_1 are mapped to the nodes labelled 5 and 6 in t_2 , the sup of $\{4, 5\}$ in t_1 is mapped to the sup of $\{5, 6\}$ in t_2 . The label increasing condition of the mapping also means that the label of the sup of $\{4, 5\}$ in t_1 is less than or equal to the label of the sup of $\{5, 6\}$ in t_2 .

Lemma 2.12 *Homeomorphic embedding is a preorder on \mathcal{T} .*

2 HOMEOMORPHIC EMBEDDING

Proof– To prove that homeomorphic embedding is a preorder on \mathcal{T} , it will be shown that it is a reflexive and transitive relation on \mathcal{T} .

- *homeomorphic embedding is reflexive on \mathcal{T}*

Suppose that there is a tree $t = (\langle D, \leq, \rho \rangle, L)$. Now map $\pi : D \rightarrow D$ such that $\forall d \in D \pi(d) = d$ and assume $x, y \in D$:

- *π is injective*

As $x = \pi(x) = \pi(y) = y$, π is injective.

- *π is monotonic*

Consider $x \leq y$ where $x, y \in D$. The mapping gives $\pi(x) = x$ and $\pi(y) = y$ so $x = \pi(x) \leq \pi(y) = y$ and the mapping is monotonic.

- *π is sup preserving*

The mapping defines $\pi(x \vee y) = x \vee y = \pi(x) \vee \pi(y)$. Thus the mapping is sup preserving.

- *π is label increasing*

As $\pi(x) = x$, $L(x) = L(\pi(x))$ and the mapping is label increasing.

The four conditions for homeomorphic embedding have been satisfied, so homeomorphic embedding is reflexive on \mathcal{T} .

- *homeomorphic embedding is transitive on \mathcal{T}*

Let $a = (\langle D_0, \leq_0, \rho_0 \rangle, L_0)$, $b = (\langle D_1, \leq_1, \rho_1 \rangle, L_1)$ and $c = (\langle D_2, \leq_2, \rho_2 \rangle, L_2)$ be trees. Suppose that there is a mapping $\pi_0 : D_0 \rightarrow D_1$ such that $a \sqsubseteq_T b$ and a mapping $\pi_1 : D_1 \rightarrow D_2$ such that $b \sqsubseteq_T c$. Consider the mapping $\pi_2 : D_0 \rightarrow D_2$ where $\pi_2 = \pi_1 \circ \pi_0$ and assume that $x, y \in D_0$:

- *π_2 is injective*

If $\pi_2(x) = \pi_2(y)$ then $\pi_1(\pi_0(x)) = \pi_1(\pi_0(y))$. As π_1 is injective $\pi_0(x) = \pi_0(y)$. Therefore $x = y$ as π_0 is injective.

- *π_2 is monotonic*

$$\begin{aligned} x \leq_0 y &\Rightarrow \pi_0(x) \leq_1 \pi_0(y) && (\pi_0 \text{ is monotonic}) \\ &\Rightarrow \pi_1(\pi_0(x)) \leq_2 \pi_1(\pi_0(y)) && (\pi_1 \text{ is monotonic}) \\ &\Rightarrow \pi_2(x) \leq_2 \pi_2(y) \end{aligned}$$

2 HOMEOMORPHIC EMBEDDING

Therefore π_2 is monotonic.

– π_2 is sup-preserving

$$\pi_1(\pi_0(x) \vee \pi_0(y)) = \pi_1(\pi_0(x)) \vee \pi_1(\pi_0(y)) \quad (\text{as } \pi_1 \text{ is sup preserving})$$

$$\pi_1(\pi_0(x \vee y)) = \pi_1(\pi_0(x)) \vee \pi_1(\pi_0(y)) \quad (\text{as } \pi_0 \text{ is sup preserving})$$

$$\pi_2(x \vee y) = \pi_2(x) \vee \pi_2(y)$$

The result comes straight from the definition of π_2 .

– π_2 is label increasing

As π_0 and π_1 are label increasing, $L_0(x) \leq L_1(\pi_0(x)) \leq L_2(\pi_1(\pi_0(x))) = L_2(\pi_2(x))$, so π_2 is label increasing.

As $\sqsubseteq_{\mathcal{T}}$ is reflexive and transitive on \mathcal{T} it is a preorder on \mathcal{T} . □

2.2.3 Subtrees

The next task is to prove a version of Kruskal's tree theorem. The version which will be proved is the restriction of the theorem which appears in Exercise 2.3.12 of Klop [71] pp. 34-36, using the structure and notation of the proof sketched there. The proof which is sketched in Klop [71] is rather terse and some of the proof steps omitted, so we present a full proof here.

Definition 2.13 Let $t = (\langle D, \leq, \rho \rangle, L)$ be a tree and suppose $d \in D$.

- The nodes $R(d, t) = \{p \in D \mid p \leq d\}$ are called *the nodes restricted by d in t*.
- The structure $t_1 = (\langle R(d, t), \leq_{T_1}, d \rangle, L_1)$ is defined to be a *subtree* of t , where L_1 is the restriction of L to $R(d, t)$ and $\forall a, b \in R(d, t) (a \leq b \Leftrightarrow a \leq_{T_1} b)$. If $d \neq \rho$ then t_1 is called a *proper subtree* of t .

Now two necessary lemmas about properties of subtrees will be proved.

Lemma 2.14 Let $t = (\langle D, \leq_T, \rho \rangle, L)$ be a tree. Then any subtree of t is a tree.

2 HOMEOMORPHIC EMBEDDING

Proof– Suppose $d \in D$. The subtree $t_1 = (\langle R(d, t), \leq_{T_1}, d \rangle, L_1)$ has a finite set of nodes as $R(d, t)$ is a subset of D and D is a finite set. The partial order \leq_{T_1} on $R(d, t)$ has the following properties:

- $\forall p \in R(d, t)(p \leq_{T_1} d)$ by the definition of $R(d, t)$ and
- $\forall b, c, f \in R(d, t)((b \leq_{T_1} c) \wedge (b \leq_{T_1} f)) \Leftrightarrow (c \leq_{T_1} f) \vee (f \leq_{T_1} c)$ as $((b \leq_T c) \wedge (b \leq_T f)) \Leftrightarrow (c \leq_T f) \vee (f \leq_T c)$.

Thus the subtree t_1 is a tree. □

Lemma 2.15 *Let $s = (\langle D_0, \leq_0, \rho_0 \rangle, L_0)$ be a subtree of the tree $t = (\langle D_1, \leq_1, \rho_1 \rangle, L_1)$. Then $s \sqsubseteq_T t$.*

Proof– As s is a subtree of t , $D_0 \subseteq D_1$, there is a map $\pi : D_0 \rightarrow D_1$ such that $\forall d \in D_0, \pi(d) = d$.

- *The mapping is injective*

If $\pi(x) = \pi(y)$ then $x = y$ from the definition of the mapping.

- *The mapping is monotonic*

$x \leq_0 y \Rightarrow x \leq_1 y$ from the definition of subtree. As $\pi(x) = x$ and $\pi(y) = y$ the result is immediate.

- *The mapping is sup preserving*

this comes straight from the uniqueness of the supremum and the definition of the mapping.

- *The mapping is label increasing*

As L_0 is a restriction of L_1 to D_0 this is immediate.

Thus $s \sqsubseteq_T t$. □

We can now give notation for successor nodes: nodes in a tree which are minimally less (with respect to the partial order on the nodes of the tree) than a given node. We will also be able to refer uniquely to each argument of a node.

2 HOMEOMORPHIC EMBEDDING

Definition 2.16 Let $t = (\langle D, \leq_T, \rho \rangle, L)$ be a tree and suppose $b, d \in D$. b is a *successor* of d in t if $b \leq_T d$ and $\forall c \in D, b \leq_T c \leq_T d \Rightarrow (c = b \vee c = d)$.

Definition 2.17 The *order* of a node $d \in D$ in a tree $t = (\langle D, \leq_T, \rho \rangle, L)$ is the number of nodes b of t such that b is a successor of d . This is denoted by $O(d, t)$.

Definition 2.18 The *branching order* of a tree $t = (\langle D, \leq_T, \rho \rangle, L)$ is

$$br(t) = \max\{O(d, t) \mid d \in D\}.$$

The set $\mathcal{T}_n = \{t \mid br(t) \leq n\}$ is called the *set of trees with maximum branching order n* (for $n \geq 1$).

Note that n is the fixed maximum branching order for the elements of \mathcal{T}_n . \mathcal{T}_n is the set of trees for which the embedding theorem will be proved. In the general case, a version of Kruskal's theorem can be proved for sequences of trees in \mathcal{T} . This proof will be restricted to trees with a fixed branching order, i.e. \mathcal{T}_n . From now on in this section, when a tree is referred to, this will mean an element of \mathcal{T}_n . This restriction is applied to ensure that the set of successors for a node in a tree t is bounded.

Notation 2.19 Let $t = (\langle D, \leq_T, \rho \rangle, L) \in \mathcal{T}_n$. Suppose that $\{d_1, d_2, \dots, d_m\}$ are the successors of $d \in D$ in t ($m \leq n$). Then, for $1 \leq i \leq m$, the subtree $(\langle R(d_i, t), \leq_{T_i}, \rho_i \rangle, L_i)$ is called the *i th argument of d* (denoted $Arg_t(d, i)$),

Notation 2.20 Let $t \in \mathcal{T}_n$. Then $|t|$ is the number of nodes of t .

2.2.4 Sequences

Some elementary definitions for sequences are given, and basic properties of these sequences are proved. In particular, a theorem due to Higman [59] about a set of equivalent notions in the theory of relations will be stated.

Definition 2.21 Let Σ be a set of symbols.

2 HOMEOMORPHIC EMBEDDING

- Let \mathbf{N} denote the set of natural numbers, and \mathbf{N}_i denote the set $\{i, i + 1, \dots\}$. Given an $n \in \mathbf{N}_1$ let $[n]$ denote the finite set $\{1, 2, \dots, n\}$, and let $[0]$ denote \emptyset . A *finite sequence* s of Σ is a function $s : [n] \rightarrow \Sigma$. We denote this by $s = (s_1 s_2 \dots s_n)$. The sequence with domain $[0]$ is called the empty sequence and is denoted by ϵ . The set of all finite sequences of Σ is denoted by Σ^* . An infinite sequence t is a function $t : \mathbf{N}_1 \rightarrow \Sigma$. The set of all infinite sequences of Σ is denoted by Σ^ω . We call the set $Seq(\Sigma) = \Sigma^* \cup \Sigma^\omega$ the *set of all sequences of Σ* . If s is a non-empty sequence, let s_i denote the i th element of that sequence. We denote an arbitrary sequence $t = (t_1 t_2 \dots)$ and will state if finite or infinite,
- Let $x = (x_1 x_2 \dots)$ be a (possibly infinite) sequence. Then a finite sequence y is called a *prefix of x* if and only if y is the empty sequence or there is a j such that $y = (x_1 x_2 \dots x_j)$. Given a sequence $x = (x_1 x_2 \dots)$, $(x)_1$ denotes the empty sequence and for $i \geq 1$, $(x)_{i+1}$ denotes the prefix $(x_1 x_2 \dots x_i)$,
- let $s = (s_1 s_2 \dots), t = (t_1 t_2 \dots)$ be infinite sequences. s is a *subsequence of t* (written $s \leq_{sub} t$) if and only if there are natural numbers $n_1 < n_2 < \dots$ such that for all $k \in \mathbf{N}$, $s_k = t_{n_k}$.
- let $t = (t_1 t_2 \dots)$ be a (possibly infinite) sequence of Σ . Then a finite sequence s is a *contiguous subsequence of t* ($s \leq_{cs} t$) if and only if s is the empty sequence or $s = (s_1 s_2 \dots s_i)$ and there is a $j \in \mathbf{N}$ such that $s_k = t_{k+j}$ for $k = 1, \dots, i$,
- let $s = (s_1 s_2 \dots), t = (t_1 t_2 \dots) \in Seq(\mathcal{T}_n)$ be infinite sequences of trees. s is said to be a *subtree sequence of t* (written $s \leq_{ss} t$) if and only if there are natural numbers $n_1 < n_2 < \dots$ such that for all $k \in \mathbf{N}$, s_k is a proper subtree of t_{n_k} .

The following definitions are taken from Gallier [43].

Definition 2.22 Given a preorder \preceq on A , a non-empty sequence $x = (x_1 x_2 \dots)$ of A is said to be a *decreasing chain* if $x_i \succ x_{i+1}$ for all $i \geq 1$. It is said to be an *antichain* if $x_i \mid x_j$ for all $1 \leq i < j$. If there are no infinite decreasing chains then \succ is said to be *terminating*. If $x_1 \preceq x_2 \preceq \dots$, then x is called a *weakly ascending sequence*. If x is a weakly ascending sequence and $x \leq_{sub} y$, then x is said to be a *weakly ascending subsequence of y* . If there is an element x_m of x such that there is no $n < m$ with $x_m \prec x_n$, then x_m is called a *maximal*

2 HOMEOMORPHIC EMBEDDING

element of x . If there is an element x_p of x such that there is no q with $p < q$ $x_p \succ x_q$, then x_p is called a *final element* of x .

Definition 2.23 Given a preorder \preceq on A , a sequence $(x_1x_2\dots)$ of A is said to be *good* if $\exists i < j : x_i \preceq x_j$. It is said to be *bad* otherwise. A preorder \preceq is said to be a *well quasi-order* (abbreviated *wqo*) if and only if every infinite sequence of A is good.

Now a number of equivalent properties of infinite sequences can be stated. This theorem is essentially Lemma 2.4 and Lemma 2.5 from Gallier [43], but for completeness we present the proof here. These equivalent conditions have been extensively studied (as noted in Kruskal [75]), especially in Higman [59].

Theorem 2.24 *Let \preceq be a preorder on a set A . The following conditions on A are equivalent:*

1. every infinite sequence is good
2. there are no infinite decreasing chains and no infinite antichains
3. for every infinite sequence $x = (x_1x_2\dots)$, there is a weakly ascending subsequence of x

Proof—

(1) \Rightarrow (2) If $x = (x_1x_2\dots)$ is an infinite sequence, then by (1) it is good; i.e. $\exists i < j : x_i \preceq x_j$. Thus x_i and x_j are not incomparable and x is not an antichain. Now suppose that x is an infinite decreasing chain. Then $\forall k \geq 1$ $x_k \succ x_{k+1}$ and $x_{k+1} \not\preceq x_k$. From (1), however, it is known that $\exists i < j : x_i \preceq x_j$. If $j = i + 1$ there is a contradiction immediately. If $j > (i + 1)$ then $x_j \succeq x_i \succ x_{i+1} \succ \dots \succ x_{j-1} \succ x_j$. So $x_j \succeq x_{j-1}$ by the transitivity of \succeq , and there is a contradiction.

(2) \Rightarrow (3) Suppose that there is no infinite weakly ascending subsequence of $x = (x_1x_2\dots)$. We split this argument into cases as follows.

Case 1 An element x_s of x occurs infinitely many times.

Then there exists an infinite subsequence $(x_sx_sx_s\dots)$ of x . This subsequence

is weakly ascending as $x_s \preceq x_s \preceq \dots$ (\preceq is reflexive on A), and this gives a contradiction.

Case 2 Each element of x occurs finitely many times.

We can assume, without loss of generality, that each element of x occurs only once, as we can take a subsequence of x in which we only include the last occurrence of each of the elements.

Case a There is no maximal element of x .

Thus there is an infinite ascending subsequence of x , which is a contradiction.

Case b There is a maximal element x_i of x .

Denote x_i by y_1 . Now consider the sequence $(x_{i+1}x_{i+2}\dots)$. If there is no maximal element of this sequence Case a applies, so a maximal element x_{i+k} will occur in this sequence, and we denote this by $x_{i+k} = y_2$. This construction continues, and an infinite subsequence $y = (y_1y_2\dots)$ of x is constructed. The maximality of the elements used to construct this sequence means that

$$(*) \quad \text{for each } a < b \text{ either } y_a \succ y_b \text{ or } y_a \mid y_b.$$

By hypothesis, there are no infinite decreasing chains, and so, in particular, there is no infinite decreasing chain which is a subsequence of y .

Case b1 There is no final element of y .

There is thus a subsequence of y which is an infinite decreasing chain, and this gives us an immediate contradiction.

Case b2 There is a final element y_p of y .

Note that y_p occurs only once. We denote y_p by z_1 . We now look at the sequence $(y_{p+1}y_{p+2}\dots)$. If there is no final element of this sequence, Case b1 applies, and so we may assume that there will be a final element of this sequence, which we denote by z_2 . We can continue this construction to obtain an infinite sequence $(z_1z_2\dots)$. As this is a subsequence of y , we know from $(*)$ that for each $a < b$ either $z_a \succ z_b$ or $z_a \mid z_b$. As each element of z is a final element of z by construction, we have $z_a \mid z_b$. This gives an immediate contradiction

to the hypotheses of (2), as z is an infinite antichain.

Thus each infinite sequence has a weakly ascending subsequence.

(3) \Rightarrow (1) As there is a weakly ascending subsequence for every sequence $x = (x_1 x_2 \dots)$, there is $i < j : x_i \preceq x_j$. Thus every infinite sequence is good.

Thus the properties of the theorem are equivalent. □

Example 2.25 Now we show an application of this theorem. The underlying set is the natural numbers \mathbf{N} , which are preordered (actually totally ordered) by \leq . It is easy to see that condition 2 of the Theorem 2.24 is satisfied. All members of \mathbf{N} are comparable (as \leq is a total order) and there are no infinite decreasing sequences (as \mathbf{N} has a smallest element, 0). Thus \mathbf{N} , equipped with the preorder \leq has the properties of Theorem 2.24. This fact will be used later.

2.2.5 Some Topological Arguments

The proof of the version of Kruskal's tree theorem for trees with restricted branching order proceeds by looking for a minimal counter-example to the statement of the theorem, which says that in an infinite sequence $t = (t_1 t_2 \dots)$ of trees there is $i < j : t_i \sqsubseteq_T t_j$. The next definitions give some properties of these sequences. The definitions from topology are standard, and can be found in, for example, Sutherland [118].

Definition 2.26 The function $dist : A \times A \rightarrow \mathbf{R}_{\geq 0}$ is a *metric on A* if $\forall s, t, u \in A$:

- $dist(s, t) = 0 \Leftrightarrow s = t$
- $dist(s, t) = dist(t, s)$
- $dist(s, t) \leq dist(s, u) + dist(u, t)$

Definition 2.27 The *distance metric* on \mathcal{T}_n^ω is defined in the following way: suppose $s, t \in \mathcal{T}_n^\omega$, where $s = (s_1 s_2 \dots)$ and $t = (t_1 t_2 \dots)$, then

$$\begin{aligned} dist(s, t) &= 2^{-i} && \text{if } (s)_i = (t)_i \text{ but } s_i \neq t_i, \\ &= 0 && \text{if } s = t. \end{aligned}$$

2 HOMEOMORPHIC EMBEDDING

Lemma 2.28 *The distance metric is a metric on \mathcal{T}_n^ω .*

Proof– Let $x = (x_1 x_2 \dots)$, $y = (y_1 y_2 \dots)$ and $z = (z_1 z_2 \dots)$ be sequences of trees in \mathcal{T}_n^ω .

- $dist(x, y) = 0$ if and only if $x = y$ by definition.
- The distance metric is defined such that $dist(x, y) = 2^{-i}$ if $(x)_i = (y)_i$ and $x_i \neq y_i$. So $(y)_i = (x)_i$ and $y_i \neq x_i$; hence $dist(x, y) = dist(y, x)$. If $dist(x, y) = 0$ then $x = y$ and the symmetry of $=$ gives $y = x$: thus $dist(y, x) = 0$.
- There are five cases to consider:
 - First consider the case when x, y, z are distinct. Suppose
 1. $dist(x, y) = 2^{-i}$,
 2. $dist(y, z) = 2^{-j}$,
 3. $dist(x, z) = 2^{-k}$ and
 4. $2^{-i} > 2^{-j} + 2^{-k}$

Thus the fourth condition gives $i < j$ and $i < k$. So now the definition of the distance metric and the above conditions say that $(x)_i = (y)_i = (z)_i$, $x_i \neq y_i$ and $y_i = z_i$. So $x_i \neq z_i$ and $k \leq i$. Thus there is a contradiction and $2^{-i} \leq 2^{-j} + 2^{-k}$.

- If $x = y, y \neq z$ then $dist(x, y) = 0$. Immediately $dist(x, z) = dist(y, z)$ and $0 \leq dist(x, z) + dist(y, z)$. The cases where $y \neq x, x = z$ and $x \neq y, y = z$ are proved similarly.
- If $x = y = z$ then $dist(x, y) = dist(y, z) = dist(x, z) = 0$ and $0 \leq 0 + 0$.

Therefore the distance metric is a metric on \mathcal{T}_n^ω . □

Definition 2.29 *A metric space is a pair (A, d) where A is a non-empty set and d is a metric on A .*

Thus a metric space of sequences of trees can be defined. The metric space which is of interest is the space $Met = (\mathcal{T}_n^\omega, dist)$, where $dist$ is the distance metric.

2 HOMEOMORPHIC EMBEDDING

Definition 2.30 Given a metric space (A, d) , a point $a \in A$ and a positive real number ϵ , the ϵ -neighbourhood of a in (A, d) is the set

$$N(a, \epsilon) = \{x \in A \mid d(a, x) < \epsilon\}$$

Definition 2.31 Given a metric space (A, d) and a subset H of A , x is a limit point of H if $N(x, \epsilon) \cap (H - \{x\}) \neq \emptyset$ for all $\epsilon > 0$.

Definition 2.32 A subset H of A in a metric space (A, d) is said to be *closed under d* if x is a limit point of H implies that $x \in H$.

Now a definition of a minimal sequence of trees can be given (this definition appears in Klop [71]). It will then be proved that a minimal element for a subset of \mathcal{T}_n^ω exists. This will then be used to show that the set of bad sequences of \mathcal{T}_n^ω has a minimal element and a contradiction will be derived from this to show that the set of bad sequences of \mathcal{T}_n^ω is empty.

Definition 2.33 Let $M \subseteq \mathcal{T}_n^\omega$ and $t = (t_1 t_2 \dots) \in M$. Then t is *minimal in M* if $(\forall s \in M)((\forall k \in \mathbb{N}_1)(s)_k = (t)_k \Rightarrow |t_k| \leq |s_k|)$.

Notation 2.34 Let $M \subseteq \mathcal{T}_n^\omega$ such that $M \neq \emptyset$, and $i \in \mathbb{N}$ then

$$Min_i(M) = \min\{|m_i| \mid (m_1 m_2 \dots) \in M\}$$

Note here that for $M \neq \emptyset$, $Min_i(M)$ will be non empty for any i .

Definition 2.35 A *nest of subsets of $M \subseteq \mathcal{T}_n^\omega$* (where M is non-empty) is defined as follows: $M \supseteq M_1 \supseteq M_2 \supseteq \dots$ where:

$$M_1 = \{(m_1 m_2 \dots) \in M \mid |m_1| = Min_1(M)\}$$

and for $i > 1$:

$$M_i = \{(m_1 m_2 \dots) \in M_{i-1} \mid |m_i| = Min_i(M_{i-1})\}$$

Definition 2.36 Let (A, d) be a metric space and $S \subset A$. The set $cl(S) = S \cup S'$ is called the *closure of S* where S' consists of all the limit points of S .

2 HOMEOMORPHIC EMBEDDING

Note here that a result from standard topology texts (see, for example, Sutherland [118], for the theorem and proof) says that $cl(S)$ is a closed set.

Lemma 2.37 *Let C be the set of bad sequences of \mathcal{T}_n^ω . Then C is closed under the distance metric.*

Proof– Let $x = (x_1x_2\dots)$ be a limit point of C and suppose that $x \notin C$. Then $\exists i < j : x_i \sqsubseteq_T x_j$. So $\forall y \in C$, y does not have the prefix $(x)_{j+1}$ (or else there would be two trees which occur in the sequence y and which are homeomorphically embedded). So $\forall y \in C$ $dist(x, y) > 2^{-(j+1)}$. Now choose $\epsilon > 0$ such that $2^{-(j+1)} > \epsilon$. Thus $\forall y \in C$, $x \notin N(y, \epsilon)$ and there is a contradiction as x is not a limit point of C . Thus C is closed under the distance metric. \square

Lemma 2.38 *Let $M \subseteq \mathcal{T}_n^\omega$ be non-empty and closed under the distance metric. Then there is a minimal element in M .*

Proof– Construct a nest of subsets of M so that $M \supseteq M_1 \supseteq M_2 \supseteq \dots$. Now consider a sequence $p = (p_1p_2\dots)$ in which $|p_1| = Min_1(M)$ and $|p_i| = Min_i(M_{i-1})$ for $i > 1$. Then $p \in \bigcap cl(M_i)$. As M is closed and so are the $cl(M_i)$, we have $\bigcap cl(M_i) \subseteq M$. So $p \in M$ and p is minimal by construction. \square

2.2.6 Proof of Kruskal's Tree Theorem

This section shows that every sequence of trees is good by showing that a minimal counter-example does not exist. The following sequence of lemmas follows the outline of Klop [71], in particular Lemmas 2.39 and 2.40 and Theorem 2.42 appear in that proof. We also fill in the details omitted from that proof.

Lemma 2.39 *Let $t = (t_1t_2\dots)$ be a minimal element of the set C of bad sequences of \mathcal{T}_n^ω . Suppose $s = (s_1s_2\dots) \leq_{ss} t$, then $\exists e < f : s_e \sqsubseteq_T s_f$.*

Proof– As $s \leq_{ss} t$ there is an i such that s_1 is a proper subtree of t_i . Consider the sequence $v = (t_1t_2\dots t_{i-1}s_1s_2\dots) = (v_1v_2\dots)$. This is not in C as t was defined to be a minimal

2 HOMEOMORPHIC EMBEDDING

element in M , $(v)_i = (t)_i$ and $|v_i| < |t_i|$. So as v is not in C there are two elements of the sequence v , $a < b : v_a \sqsubseteq_T v_b$. These two elements can not both occur in the segment $(v)_i$ as there are no two elements of t , $c < d : t_c \sqsubseteq_T t_d$. Now suppose that $t_g \sqsubseteq_T s_j$. But s_j is a proper subtree of some t_k (so by Lemma 2.15 $s_j \sqsubseteq_T t_k$) and thus by the transitivity of the homeomorphic embedding relation on \mathcal{T} , $t_g \sqsubseteq_T t_k$. Again there is a contradiction. Thus there must be some $e < f : s_e \sqsubseteq_T s_f$. \square

Lemma 2.40 *Let $t = (t_1 t_2 \dots)$ be a minimal element of the set C of bad sequences of \mathcal{T}_n^ω . For each infinite sequence $s = (s_1 s_2 \dots)$ such that s is a subtree sequence of t there is a subsequence $r = (r_1 r_2 \dots)$ of s such that $r_1 \sqsubseteq_T r_2 \sqsubseteq_T \dots$*

Proof— Look for a proof by contradiction. Suppose that there is no infinite sequence r such that r is a weakly ascending subsequence of s . There is a maximal element of s as there would be an immediate contradiction if this were not true. Say that the maximal element is s_i . If this did not occur finitely many times, there would be a subsequence $(s_i s_i \dots)$ of s , which is weakly ascending as \sqsubseteq_T is a reflexive relation on \mathcal{T}_n . So we can choose a final occurrence of i which we will denote s_{i_k} . We say $s_{i_k} = r_1$. We now consider the sequence $(s_{i_k+1} s_{i_k+2} \dots)$. Again, as there is no infinite ascending subsequence of this sequence, a maximal element $s_{i_k+j} = r_2$ exists. We continue this construction and form a sequence $(r_1 r_2 \dots)$. Lemma 2.39 says that there is $e < f : r_e \sqsubseteq_T r_f$: this contradicts the maximality of the elements which were chosen to construct r . \square

Lemma 2.41 *Let $t_1 = (\langle D_1, \leq_1, \rho_1 \rangle, L_1)$ and $t_2 = (\langle D_2, \leq_2, \rho_2 \rangle, L_2) \in \mathcal{T}_n$ be trees such that the following conditions hold:*

- $L_1(\rho_1) \leq L_2(\rho_2)$
- $k = O(\rho_1, t_1) \leq O(\rho_2, t_2)$
- *there exist maps $\pi_1, \pi_2, \dots, \pi_k$ such that $\text{Arg}_{t_1}(\rho_1, m)$ is homeomorphically embedded in $\text{Arg}_{t_2}(\rho_2, m)$ via π_m ($1 \leq m \leq k$).*

Then $t_1 \sqsubseteq_T t_2$.

Proof— There is a map $\pi : D_1 \rightarrow D_2$ such that:

2 HOMEOMORPHIC EMBEDDING

1. $\pi(\rho_1) = \rho_2$
2. map all other nodes in exactly the same way as $\pi_1, \pi_2, \dots, \pi_k$

- *The mapping is injective*

immediate from the uniqueness of the root and the fact that the mapping of the arguments is injective.

- *The mapping is monotonic*

from the mapping of the arguments of the root and the definition of the root.

- *The mapping is sup preserving*

There are two cases:

1. the nodes are in the same argument of the root (or at least one is the root itself).
In this case the mapping on the arguments and the definition of the root gives the result.
2. the nodes are in different arguments of the root. $x \vee y = \rho_1$, so it is mapped to ρ_2 . x and y are mapped to separate arguments of the root of t_2 , and so $\pi(x) \vee \pi(y) = \rho_2$.

- *The mapping is label increasing*

this is an immediate property of the mapping.

□

Theorem 2.42 *Let C be the set of bad sequences of \mathcal{T}_n^ω . Then $C = \emptyset$.*

Proof – Suppose the hypothesis is not true. By Lemma 2.37, the set C is closed under the distance metric, and we can then apply Lemma 2.38 to choose a minimal element $t = (t_1 t_2 \dots)$ of C . Suppose the roots and the labelling functions of t_1, t_2, \dots are r_1, r_2, \dots and L_1, L_2, \dots respectively. There is now a sequence of at most $n + 2$ steps which refines the sequence to a subsequence at each stage. These refinements will give us a contradiction:

1. by Example 2.25 there is a subsequence $t' = (t_i t_j \dots)$ of t such that

$$(L_i(r_i) L_j(r_j) \dots) \leq_{was} (L_1(r_1) L_2(r_2) \dots).$$

2 HOMEOMORPHIC EMBEDDING

2. by Example 2.25 a subsequence $t'' = (t_a t_b \dots)$ of t' can be chosen such that we get the following: $(O(r_a, t_a)O(r_b, t_b) \dots) \leq_{was} (O(r_i, t_i)O(r_j, t_j) \dots)$. Thus the number of arguments of each of the trees in the sequence is weakly ascending.
3. now consider the first arguments of the roots of the trees in t'' . Let r_a, r_b, \dots be the roots of t_a, t_b, \dots respectively. The sequence $(Arg_{t_a}(r_a, 1)Arg_{t_b}(r_b, 1) \dots)$ is a subtree sequence of the minimal bad sequence t , and so by Lemma 2.40 there is a weakly ascending subsequence of these arguments. So now consider the subsequence $t''' = (t_v t_w \dots)$ of t'' such that

$$Arg_{t_v}(r_v, 1) \sqsubseteq_T Arg_{t_w}(r_w, 1) \sqsubseteq_T \dots$$

Similarly a subsequence $t'''' = (t_c t_d \dots)$ of t''' can be found such that $Arg_{t_c}(r_c, 2) \sqsubseteq_T Arg_{t_d}(r_d, 2) \sqsubseteq_T \dots$. This ‘refining’ of the sequence can be continued for all arguments of the roots of the remaining trees. There will be at most n of these steps as the branching order for the trees is restricted to n . There is now a contradiction directly from Lemma 2.41 and $C = \emptyset$. □

Corollary 2.43 A Restricted Version of Kruskal’s Tree Theorem

Every infinite sequence of trees of \mathcal{T}_n is good under the preorder \sqsubseteq_T .

Proof – This comes directly from Theorem 2.42 and Theorem 2.24. As the set of bad sequences for \mathcal{T}_n^ω is the empty set, then every infinite sequence is good: this satisfies condition 1 of Theorem 2.24. □

The restriction that we would only work with trees with restricted branching order was used in the proof of Theorem 2.42, and ensured that we only needed to look at finitely many subtrees of a given tree.

2.3 Highlights of Embedding Theorems

In this section we will discuss the history of the well quasi-order and the ‘homeomorphic embedding’ theorems. These concepts appear in many different contexts, as was noted in Kruskal [75], and so appear to be fundamental in many areas of mathematics. The results will be stated, and references to where the proofs can be found will be given. We will

2 HOMEOMORPHIC EMBEDDING

discuss some of the applications of this theory and also give examples of embedding and non-embedding.

The notion of the well quasi-order has been around at least since Janet [64] (as noted by Lescanne [79] and Gallier [43]). The first of the embedding theorems was proved by Higman [59] and is for strings. Higman proves Theorem 2.24 which characterises a number of properties which are equivalent to the wqo. The motivation for this was to simplify some previous work about embedding group algebras in division rings.

Definition 2.44 Let \preceq be a preorder on a set A . The preorder \sqsubseteq , which is called *string embedding*, on A^* is defined as follows:

- $\epsilon \sqsubseteq u$ for each $u \in A^*$ and
- if $u = (u_1 u_2 \dots u_m)$ and $v = (v_1 v_2 \dots v_n)$ with $1 \leq m \leq n$, then $u \sqsubseteq v$ if and only if there are integers j_1, j_2, \dots, j_m such that $1 \leq j_1 < j_2 < \dots < j_m \leq n$ and

$$u_1 \preceq v_{j_1}, u_2 \preceq v_{j_2}, \dots, u_m \preceq v_{j_m}$$

As an example of this, suppose we have the usual ordering on natural numbers \leq . Then $111 \sqsubseteq 2102$ as $1 \leq 2, 1 \leq 1$ and $1 \leq 2$. It is also possible to see that $111 \not\sqsubseteq 202$. The following theorem is due to Higman [59]:

Theorem 2.45 Let \preceq be a wqo on a set A . Then \sqsubseteq is a wqo on A^* .

A restricted form of Kruskal's tree theorem was proved in the last section. The unrestricted version (i.e. where there is no fixed branching order) was proved in Kruskal [74]. The statement of the theorem is as follows:

Theorem 2.46 Let $(t_1 t_2 \dots)$ be an infinite sequence of trees in \mathcal{T} . Then $\exists i \leq j : t_i \sqsubseteq_{\mathcal{T}} t_j$

This result has many applications in logic and computer science. For instance, it is used to show that orderings on terms are well founded, and so can be used to show the termination of sets of rewrite rules in Knuth-Bendix completion (see, for example, Dershowitz and Jouannaud [37]). Gallier [43] looks extensively at the proof theoretic aspects of Kruskal's

2 HOMEOMORPHIC EMBEDDING

tree theorem, and in particular the relationship between the theorem and a countable ordinal, Γ_0 .

The previous section gave examples of homeomorphic embedding for trees, so we will restrict ourselves to an example of non-embedding for trees, which we shall present here in usual term notation. We will show below that the corresponding definition of ordering for graphs does allow the two trees to be related. This example shows the nodes of the trees, and it can easily be checked that $t_1 = a(b, d, e)$ does not homeomorphically embed in $t_2 = a(b, c(d, e))$ for any labelling. This is a direct consequence of the sup preserving condition in the homeomorphic embedding relation.

A similar result to Kruskal's tree theorem has been proved for classes of graphs: these results are called *graph-minor theorems*. Robertson and Seymour have published many papers on this subject (a survey of results and where to find the proofs can be found in Robertson and Seymour [105]). There are, for example, graph minor theorems for undirected graph and hypergraphs. One of the major applications of this work is that any class of graphs which is closed under taking minors is determined by a finite set of 'forbidden minors'. An instance of this is the Kuratowski-Wagner theorem (see e.g. Cameron [20]) which states that a graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor. We will define the relation 'is a minor of' for undirected graphs, where an undirected graph is defined to be $\Gamma = (V, E)$ where V is a set of vertices and E is a set of edges. We will define a (directed) graph later in this thesis (Section 3.2). We start first with the definition of a graph minor:

Definition 2.47 Let $\Gamma = (V, E)$ be an undirected graph with $e = \{x, y\} \in E$ and $z \notin V$. Then we define:

- *deletion* of e to be the construction of the undirected graph $\Gamma - \{e\} = (V, E - \{e\})$
- *contraction* of e to be the construction of the undirected graph $\Gamma / \{e\} = (V_1, E_1)$ where:

$$- V_1 = (V - \{x, y\}) \cup \{z\}$$

- to obtain E_1 , for each edge $f \in E$ replace each occurrence of x or y by z

An undirected graph Γ_0 is a *minor* of an undirected graph Γ if it can be obtained from Γ by

2 HOMEOMORPHIC EMBEDDING

a series of deletions and contractions. The relation *is a minor of* on a class of undirected graphs will be denoted \sqsubseteq_{Γ} .

It can be seen here why there is currently no corresponding theorem for labelled graphs. If the graphs were labelled, the contraction construction in the definition means that there is a ‘choice’ of which label to take for the ‘new’ contracted node. We state the Robertson-Seymour theorem as in Cameron [20].

Theorem 2.48 *Let \mathcal{C} be a class of graphs which is closed under taking minors. Then there is a finite set \mathcal{S} of graphs with the property that $\Gamma \in \mathcal{C}$ if and only if no member of \mathcal{S} is a minor of Γ .*

The following example will show that there is a difference between ‘is a minor of’ on a class of graphs and the way that we have defined the homeomorphic embedding relation for trees. We can construct undirected graphs from the trees we defined in the non-embedding example for Kruskal’s theorem.

$$\begin{aligned}\Gamma_{t_1} &= (\{a, b, d, e\}, \{\{a, b\}, \{a, d\}, \{a, e\}\}) \\ \Gamma_{t_2} &= (\{a, b, c, d, e\}, \{\{a, b\}, \{a, c\}, \{c, d\}, \{c, e\}\})\end{aligned}$$

From the way the relation is defined, it is possible to see that $\Gamma_{t_1} \sqsubseteq_{\Gamma} \Gamma_{t_2}$.

3 An Introduction to Proof Diagrams

3.1 Introduction

Logicians typically use proof trees to represent the proofs of formulae. This representation has the advantage that it is easy to follow. However, proof trees are not so well suited for representing proofs in which a formula occurs more than once. Programs which implement, for instance, Knuth-Bendix completion and Gröbner bases can ‘prove’ formulae in many different ways. This chapter introduces a method to represent the executions of such programs. These execution traces correspond to proofs of given formulae. Proofs of facts here are represented by graphs and structures called proof diagrams, which are like ordered hypergraphs.

Section 3.2 introduces notation and definitions for graphs (as given in Barendregt *et al* [8]). We introduce an embedding theorem, like the one for trees in Section 2.2, for a class of directed acyclic graphs (DAGs). This is different from the graph minor theorem of Robertson and Seymour [105] as it depends on the notion of homeomorphic embedding for trees, which was discussed in Chapter 2. This theory means that local structure can be ‘thrown away’ from this class of DAGs. This notion of throwing away structure characterises the use of a heuristic in this thesis: we wish to ‘quickly’ derive useful facts by missing out complex intermediate substructure in our proofs of facts (see Section 1.1.3 for more details).

The concept of the proof diagram is introduced in Section 3.3.1. Some basic definitions for propositional resolution from a given set of clauses are given in Section 3.3.2. This is used as an example throughout this chapter. A graphical soundness condition is introduced in order to reason about, and represent, the proof of a given formula. A way to map proof diagrams onto graphs is given in Section 3.3.4. By viewing a proof P of a fact a as a graph, one can reason about the local properties of the proof of a ; for instance the ancestry of a particular formula. This dependency is obviously very important when trying to define ‘shortcuts’ through the search space. These two structures are used to introduce the concept of a proof P of a fact a being minimal with respect to a proof B of a in Section 3.3.5. In this case, it is not just a single shortest proof of a which is being looked for, but various shortest proofs of a which have different structures. It is through analysing these different proofs of a that common structure can be found and new heuristics developed. We then prove some

simple consequences of the definitions introduced in this chapter.

Section 3.6 gives a proof that the set of proofs P of a which are minimal with respect to certain proofs B of a is finite. The class of proof diagrams which have this property have two necessary restrictions: the number of facts that can be derived in one step from a given set of facts is finite and any particular fact can only be derived by a finite number of inferences.

3.2 Graphs and a Version of Kruskal's Theorem

This section will develop an embedding theorem for a class of directed acyclic graphs (DAGs) by using Kruskal's tree theorem, which was proved in Chapter 2. Basic definitions for graphs, paths and an *unravelling* function (see Barendregt *et al* [8]) are given and some elementary properties of graphs are proved.

3.2.1 Graphs

The definition from Barendregt *et al* [8] for graphs is first given. Note that in Barendregt *et al* [8] all graphs are rooted. Here all graphs will be rooted as well. We will introduce the concept of a *rooted graph*, in which there are added restrictions to the root, later in this section. DAGs are defined and it is proved that there are finitely many paths in a DAG.

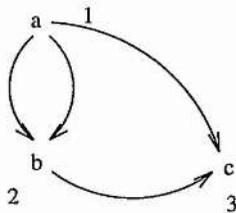
Definition 3.1 A *directed graph* over \mathbb{N} is a 4-tuple $(N, lab, succ, r)$ comprising a finite non-empty set N called the *nodes* of the graph, a function $lab : N \rightarrow \mathbb{N}$, a function $succ : N \rightarrow N^*$ and a distinguished node $r \in N$ called the *root*.

Example 3.2 As an example, consider the graph $G = (\{a, b, c\}, lab, succ, a)$ where:

- $lab(a) = 1, lab(b) = 2$ and $lab(c) = 3$ and
- $succ(a) = (bbc), succ(b) = (c)$ and $succ(c) = \epsilon$.

Figure 2 shows the graphical representation of G .

Note that this definition of graph is more like the standard definition of a multigraph: namely, there can be more than one edge between each pair of nodes; e.g. in Example

Figure 2: Graphical representation of G

3.2 $\text{succ}(a) = (bbc)$ means that there are two edges from a to b . Also the function lab is not necessarily injective: we may have the same label for two different nodes in a graph G . From now on a directed graph will be referred to as a graph.

Definition 3.3 Let $G = (N, \text{lab}, \text{succ}, r)$ be a graph.

1. If $\text{succ}(n) = (n_1 \dots n_m)$ for $n \in N$, then each n_i ($1 \leq i \leq m$) is called a successor of n .
2. A path in G is a sequence $(n_1 \dots n_m)$, ($m \geq 1$), where each n_{k+1} is a successor of n_k ($1 \leq k \leq m - 1$), $n_1, \dots, n_m \in N$. The path is said to be from n_1 to n_m and the length of the path is $m - 1$. The set of all paths in G is denoted $\text{Paths}(G)$.
3. A rooted path is a path which starts at r ,
4. A cycle is a path of length greater than 0 from a node n to itself. G is said to be cyclic if it contains such a path,
5. G is acyclic if no path is a cycle; such a graph is called a DAG.

The rooted paths in G from Example 3.2 are $R = \{(a), (ab), (abc), (ac)\}$, the paths which are not rooted paths are $\bar{R} = \{(b), (c), (bc)\}$, and so the set $\text{Paths}(G) = R \cup \bar{R}$. It is easy to check that G is a DAG.

Now we wish to prove some lemmas about the set $\text{Paths}(G)$ when G is a DAG. These lemmas will be used to show that it is possible to construct, from an arbitrary DAG, a DAG in which the root has the following two properties: there are no nodes of which the root is a successor and there is at least one path from the root to each node.

3 AN INTRODUCTION TO PROOF DIAGRAMS

Lemma 3.4 Let $p = (p_1 p_2 \dots)$ be a path in a graph G . Then every contiguous subsequence $q = (q_1 \dots q_k)$ of p is also a path in G .

Proof– Suppose that q is not a path in G . Then there is an i such that q_{i+1} is not a successor of q_i ($i > 1$). That means there is a node p_{i+j+1} which is not a successor of p_{i+j} in the path p , and there is a contradiction. Thus all contiguous subsequences of a path p are also paths in G . \square

Lemma 3.5 Let $G = (N, lab, succ, r)$ be a DAG. Then there are finitely many distinct paths in G .

Proof– As G is acyclic, there is no path of the form $(n \dots n)$. So each path contains at most one occurrence of each element of N . Thus if N has l elements, there will be at most $\sum_{i=1}^l \binom{l}{i} i!$ paths, where i ranges over the length of the sequence. \square

3.2.2 Sources in Graphs

Now we will define a subset of the nodes of a graph which have desirable properties. These nodes turn out to be very important when defining a rooted DAG.

Definition 3.6 Let $G = (N, lab, succ, r)$ be a graph. A node $n \in N$ is defined to be a *source* if $\forall p \in N, n$ is not a successor of p . The set $S(G) = \{n \in N \mid n \text{ is a source}\}$ is called the *sources of G* .

For instance, in Example 3.2, a is a source and $S(G) = \{a\}$. In fact, it can be shown that, for any DAG, the set of sources is non-empty.

Lemma 3.7 Let $G = (N, lab, succ, r)$ be a DAG. Then $S(G) \neq \emptyset$.

Proof– Lemma 3.5 guarantees that $Paths(G)$ is a finite set. If there was an element p of this set which was of infinite length, then Lemma 3.4 says that each contiguous subsequence of p is also a path in G . So there would be infinitely many paths, which is a contradiction. Thus there will be a longest element of $Paths(G)$; say this is $p = (p_1 \dots p_k)$. Thus p_1 is

not the successor of any node (if it were, the path p could be extended and p would not be the longest path). Hence $p_1 \in S(G)$. \square

We also note here that the set $S(G)$ in a DAG G is finite. This follows immediately from the fact that the set of nodes of G is finite. Next the concept of a rooted DAG is introduced. We will be able to use the definition of the sources of the graph and the above lemma to construct a rooted DAG from an arbitrary DAG.

Definition 3.8 A graph $G = (N, lab, succ, r)$ is called a *rooted graph* if it has the following properties:

1. there is at least one path from r to each node,
2. r is a source

We call a graph G a *rooted DAG* if G is a rooted graph and G is a DAG.

Definition 3.9 Let $G = (N, lab, succ, p)$ be a DAG and $S(G) = \{s_1, s_2, \dots, s_d\}$. If $r \notin N$ the *sourced structure* of G is defined as follows: $Sourced(G) = (N \dot{\cup} \{r\}, lab_1, succ_1, r)$ where N is as in G , lab_1 and $succ_1$ extend lab and $succ$ such that $lab_1(r)$ is any element of N and $succ_1(r) = (s_1 s_2 \dots s_d)$.

Lemma 3.10 Let $G = (N, lab, succ, d)$ be a DAG. Then $Sourced(G)$ is a rooted DAG.

Proof – The set $N \dot{\cup} \{r\}$ is finite as N is finite. As $succ_1$ and lab_1 are defined on all elements of $N \dot{\cup} \{r\}$, the sourced structure is a graph. Suppose that there was a node $l \in N$ such that there was no path from r to l . Look at all paths on which l lies. Suppose that l is a source in G , then there is a path from r to l in $Sourced(G)$ as l is a successor of r in $Sourced(G)$, which is a contradiction. Now there must be a path from a source node to l in G . This source node will be a successor of r in $Sourced(G)$, and so there will be a path from r to l , and again there is a contradiction. Thus there is a path from r to each node. r is a source in the graph as the definition shows that there is no node of which r is the successor. Suppose it was not a DAG. As G is a DAG, any cycle must include the node r , but r is a source and so there is a contradiction. Thus the sourced structure of G is a rooted DAG. \square

Thus a question about an arbitrary DAG can be transformed into a question about a rooted DAG by carrying out the construction of the sourced structure. We will concentrate on proving an embedding theorem for rooted DAGs in the next section.

3.2.3 Homeomorphic Embedding for Rooted DAGs

Now a definition for unravelling a graph will be given. This is a useful construction, as an embedding theorem (like Kruskal's theorem for trees) will be proved. We do this by defining a version of the homeomorphic embedding relation for rooted DAGs in terms of unravelling and the homeomorphic embedding relation for trees. The following definition is taken from Barendregt *et al* [8].

Definition 3.11 Let $G = (N, lab, succ, r)$ be a DAG. The *unravelling of G* , the structure $U(G) = (N_{U(G)}, lab_{U(G)}, succ_{U(G)}, r_{U(G)})$, can be defined in the following way:

- the rooted paths of G are the nodes of $U(G)$
- $r_{U(G)}$ is the path (r) .
- Given a path $p = (rn_1 \dots n_m)$ in G , $lab_{U(G)}(p) = lab(n_m)$ and
- Given a path $p = (rn_1 \dots n_m)$ in G , $succ_{U(G)}(p) = (p_1 \dots p_k)$ where p_i is the result of appending the i th element of $succ(n_m)$ to p .

Note here that the rooted paths of G are not necessarily distinct. In Figure 2, there are two rooted paths of the form (ab) and two of the form (abc) . Both pairs of rooted paths must be considered when unravelling the graph, and each of these paths is a node in the unravelling of the graph. We look back at the graph in Figure 2 as an example.

The rooted paths of G are the sequences of nodes (a) , (ab) , (ab) , (ac) , (abc) and (abc) , and these will be the nodes of the graph $U(G)$. The root of $U(G)$ is (a) .

- $lab_{U(G)}((a)) = lab(a) = 1$, $lab_{U(G)}((ab)) = lab(b) = 2$ (for both occurrences of (ab)), $lab_{U(G)}((ac)) = lab(c) = 3$ and $lab_{U(G)}((abc)) = lab(c) = 3$ (for both occurrences of (abc)).

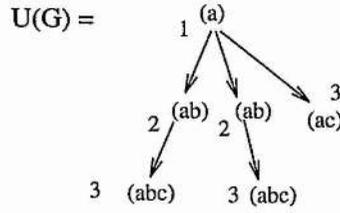


Figure 3: Unravelling a graph

- $\text{succ}_{U(G)}((a)) = ((ab)(ac))$ $\text{succ}_{U(G)}((ab)) = ((abc))$ (for both occurrences of (ab)), $\text{succ}_{U(G)}((ac)) = \epsilon$ and $\text{succ}_{U(G)}((abc)) = \epsilon$ (for both occurrences of (abc)).

It is easy to see that a partial order on the nodes of $U(G)$ is obtained in this example by looking at the prefixes of the nodes of $U(G)$. This will be used in the next lemma to show that an unravelled rooted DAG may be regarded as a tree.

Lemma 3.12 *Let $G = (N_1, \text{lab}_1, \text{succ}_1, r)$ be a rooted DAG, then the unravelling of G , $U(G) = (N, \text{lab}, \text{succ}, r)$, may be regarded as a tree.*

Proof– The structure of $U(G)$ can be translated in the following way to the structure of a tree $t = (\langle D, \leq_T, t_T \rangle, L_T)$. Let $N = D$ and the function lab translate to the function L_T . The root of t will be the path (r) . The set D will be finite as Lemma 3.5 guarantees that a DAG has a finite number of distinct paths: the rooted paths are the nodes of $U(G)$.

Define the partial order \leq_T as follows: $a \leq_T b$ if and only if b is a prefix of a . As all nodes of $U(G)$ are rooted paths every node of $U(G)$ will have (r) as a prefix. From the way the partial order has been defined, $n \leq_T (r)$ for all nodes $n \in D$. Now suppose that $b \leq_T c$ and $b \leq_T d$ for $b, c, d \in D$. So b is a sequence $(rn_1 \dots n_m)$ and both c and d are prefixes of b . As b is a fixed sequence, either c is a prefix of d or d is a prefix of c ; i.e. $c \leq_T d$ or $d \leq_T c$. So D is finite and has a partial order with the properties required by Definition 2.5. Hence $U(G)$ may be regarded as a tree. □

We now use the definition of homeomorphic embedding (for trees) to define the homeomorphic embedding which we are interested in.

Definition 3.13 Let G_1, G_2 be rooted DAGs. G_1 is said to be *homeomorphically embedded* in G_2 (denoted $G_1 \sqsubseteq_G G_2$) if and only if $U(G_1) \sqsubseteq_T U(G_2)$. The relation \sqsubseteq_G is called the

3 AN INTRODUCTION TO PROOF DIAGRAMS

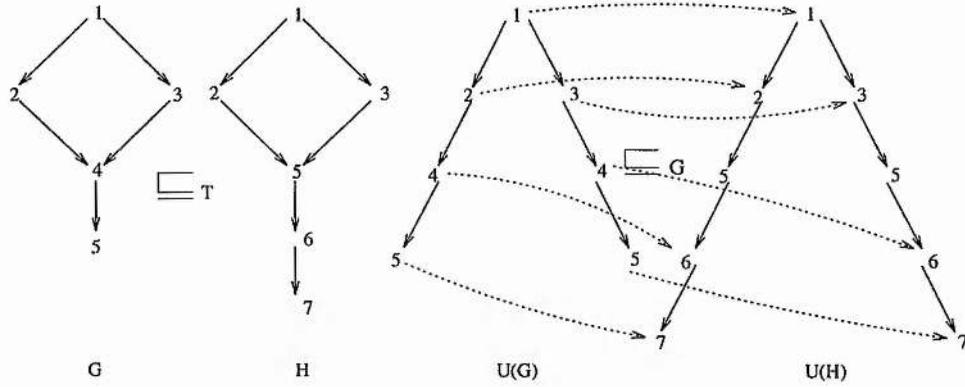


Figure 4: An example of rooted DAG embedding

homeomorphic embedding relation on rooted DAGs.

Figure 4 gives an example of the homeomorphic embedding relation for rooted DAGs by showing how $U(G)$ is homeomorphically embedded as a tree in $U(H)$. The dotted arrows show how the nodes can be mapped (for simplicity only the labels of the graph are shown; each label is associated with a node). Below it is shown that Kruskal's theorem can be extended to rooted DAGs.

Lemma 3.14 *The relation \subseteq_G is a preorder on rooted DAGs.*

Proof– This follows immediately from the fact that \subseteq_T is a preorder on trees. □

Theorem 3.15 An Embedding Theorem for Rooted DAGs

Let $(g_1 g_2 \dots)$ be an infinite sequence of rooted DAGs. Then $\exists i < j \in \mathbb{N}$ such that $g_i \subseteq_G g_j$.

Proof– The finite DAGs can be unravelled to obtain an infinite sequence of trees $(t_1 t_2 \dots)$ (from Lemma 3.12). Kruskal's tree theorem (Corollary 2.46) can be applied to this sequence of trees to say that there exists $i < j$ with $t_i \subseteq_T t_j$. So, by definition, g_i is homeomorphically embedded in g_j as rooted DAGs. □

This theorem states that rooted DAGs have property 1 of Theorem 2.24. Thus rooted DAGs have all the equivalent properties of Theorem 2.24.

3.3 Proofs of Facts as Graphs

This section will develop a theory of ‘proofs of facts as graphs’. The notions of proof diagrams and proof graphs are introduced so that one can reason about the ‘good’ proofs of a fact a which occur in a proof of a which contains many redundancies. Some simple consequences of these definitions will then be proved. We informally define propositional resolution from a given set of clauses in Section 3.3.2, and will use this throughout this section as an extended example.

3.3.1 Proof Diagrams

Some basic definitions are now given which relate inference systems and proof in logic to certain types of graphs.

Definition 3.16 A system $Syst = (Symb, \Phi, V, \approx)$ consists of:

1. An alphabet of symbols $Symb$
2. A set Φ of finite strings of these symbols, called *well-formed formulae* or *facts*
3. A subset V of $\mathbf{P}(\Phi) \times \Phi$, called the *inferences over Φ* . An element $v \in V$ is denoted $v = (A; b)$ where $A \in \mathbf{P}(\Phi)$ and $b \in \Phi$. The *axioms* of V are defined to be the set $In = \{(A; b) \mid A = \emptyset\}$.
4. An equivalence relation \approx on Φ

Informally, an inference system is being modelled (see, for instance, Hamilton [52]) where V is the set of all valid inferences: i.e. instances of an inference rule using appropriate notions of substitution, variable naming and so on. The equivalence relation on ϕ allows us, if necessary, to simplify the structure by calling some facts equivalent, to incorporate, for example, notions of equivalence up to permutation.

It now seems that In is a poor choice of notation for the axioms; it was originally meant to stand for Initial Inferences.

Definition 3.17 Let $Syst = (Symb, \Phi, V, \approx)$ be a system. A *proof diagram* P is a subset of V . A *proof subdiagram of a proof diagram* P is a subset of P . Let $|P|$ denote the cardinality of the proof diagram P .

Note here that P can be empty, and in this case $|P| = 0$. At this stage, there is no notion of ‘proof of a fact’: this will be defined in Section 3.6. From now on, when we mention a proof diagram P , we will be assuming the existence of a system.

3.3.2 A System for Propositional Resolution

In this section, the basic notions for propositional resolution from a given set of clauses will be informally defined. The notation and structure of Jäger [63] will be used. This is then used so that a system (as in Definition 3.16) for propositional resolution from a given set of clauses can be stated, and this will be used in the rest of Section 3.3 as an extended example.

We start our informal definition of propositional resolution from a given set of clauses with some definitions for building logical formulae.

Definition 3.18 Suppose we have the following propositional symbols: the alphabet of propositional variables; the symbol $'$ for forming the complements of the propositional variables; the propositional constants \top (true) and \perp (false); and the propositional connectives \wedge (and) and \vee (or). Then we can inductively generate *formulae* as follows:

- all propositional variables, their complements and propositional constants are formulae
- if A and B are formulae, then so are $A \vee B$ and $A \wedge B$

Negation is now introduced for arbitrary formulae. We do this through the complements of propositional variables, de Morgan’s laws and double negation.

Definition 3.19 If A is a propositional variable then $\neg A = A'$, $\neg A' = A$. We also have $\neg\top = \perp$, $\neg\perp = \top$, $\neg(A \vee B) = \neg A \wedge \neg B$, $\neg(A \wedge B) = \neg A \vee \neg B$.

The semantics of classical propositional logic is based on the two truth values T (true) and F (false).

Definition 3.20 A truth function is a function τ which assigns a truth value to every formula, such that the following conditions are satisfied:

- $\tau(\top) = T$. $\tau(A') = T$ if and only if $\tau(A) = F$
- $\tau(A \vee B) = T$ if and only if $\tau(A) = T$ or $\tau(B) = T$,
- $\tau(A \wedge B) = T$ if and only if $\tau(A) = T$ and $\tau(B) = T$,

A formula A is called satisfiable if $\tau(A) = T$ for some truth function τ .

We now introduce the notion of formulae in conjunctive normal form (CNF). In principle, any formula is equivalent to a formula in CNF. However, the conversion of a formula into CNF is an exponentially complex process, so for the method to be feasible we choose to start with the formula in CNF.

Definition 3.21 A formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction $C_1 \wedge C_2 \wedge \dots$ of disjunctions $C_p = L_{p,1} \vee L_{p,2} \vee \dots$, where each $L_{j,k}$ is a propositional variable or is the complement of a propositional variable. Each of the C_n is called a *clause*. We denote the empty clause by ff .

The inference rule 'pr_r', the propositional resolution rule from a given set of clauses, is stated below.

$$\text{inference rule pr}_r \quad \frac{\bigvee_j B_j \vee A \quad \bigvee_k D_k \vee A'}{\bigvee_j B_j \vee \bigvee_k D_k}$$

where A, A', B_j 's, D_k 's are propositional variables and their complements.

The next definition gives the structure of the inferences which we wish to model with the aid of proof diagrams. This will correspond to the 'edges' in the graphical representation of the proof diagram.

Definition 3.22 The clause $\bigvee_j B_j \vee \bigvee_k D_k$ is said to be a *resolvent* of the clauses $\bigvee_j B_j \vee A$ and $\bigvee_k D_k \vee A'$, and the process of adding a resolvent of two clauses is said to be a *resolution step*.

3 AN INTRODUCTION TO PROOF DIAGRAMS

Suppose we are given an initial formula \mathcal{S} in CNF. Resolvents are calculated from pairs of clauses in \mathcal{S} , these new clauses are then appended to \mathcal{S} and the procedure continues iteratively. We will denote by $Res(\mathcal{S})$ the set which contains each clause which can be obtained by 0 or more applications of pr_r . The next theorem is essential in the use of the resolution method as a proof procedure. A proof can be found in Gallier [44].

Theorem 3.23 *Let \mathcal{S} be a formula in CNF. Then*

$$\mathcal{S} \text{ is satisfiable} \Leftrightarrow ff \notin Res(\mathcal{S})$$

We will now define a system (as in Definition 3.16) for propositional resolution. We are only modelling propositional resolution from a given set of clauses \mathcal{C} : no semantics is given in this definition. Suppose we are given a set of propositional variables, their complements and propositional constants, we can define a system $(Symb, \Phi, V, \approx)$ as follows:

- $Symb$ is the set of propositional symbols, their complements and propositional constants,
- $\Phi = Symb^*$. Each non-empty $\phi = XYZ \dots \in \Phi$ denotes a clause $X \vee Y \vee Z \dots$
- V contains inferences of the form $(\emptyset; \phi)$ for each $\phi \in \mathcal{C}$ and all instances of the resolution step: i.e. instances of the inference $v = (\{XA', YA\}; XY)$. We represent the inference

$$\frac{B_1 \vee \dots \vee B_m \vee A \quad D_1 \vee \dots \vee D_n \vee A'}{B_1 \vee \dots \vee B_m \vee D_1 \vee \dots \vee D_n}$$

by the inference $(\{B_1 \dots B_m A, D_1 \dots D_n A'\}; B_1 \dots B_m D_1 \dots D_n)$

- The equivalence relation defines the structure sharing of the nodes in the graph-like structure. So in the propositional resolution example, an obvious equivalence to use is the equality of the sets of the elements which occur in a clause, e.g. $AB'B'A'BA'A \approx AA'BB'$ (as $\{A, B', B', A', B, A\} = \{A, A', B, B'\}$).

There are, of course, other definitions for system which could be made. For instance, the equivalence relation \approx could be defined as follows: $\forall x, y \in \Phi, x \approx y$ if and only if $x = y$ as finite sequences. In this case $A'X$ and XA' will not be identified and thus the inferences $(\{XA', YA\}; XY)$ and $(\{A'X, YA\}; XY)$ will be different.

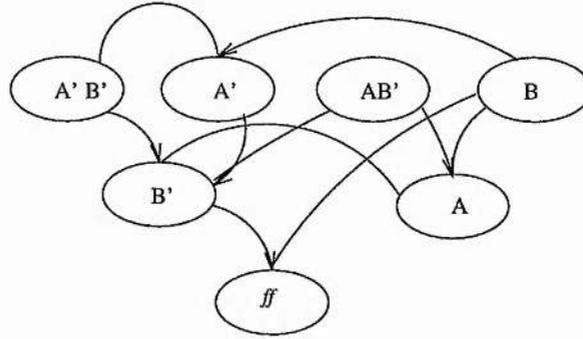


Figure 5: A proof diagram in propositional resolution

Now an example will be given which will be developed to explain the structures in this section. We will use propositional resolution from a given set of clauses, for which the only inference rule is given above, and will have as initial clauses AB' , B , A' , and $A'B'$. A proof diagram such as S can be generated. Figure 5 is a graphical representation of the proof diagram

$$S = \{(\emptyset; AB'), (\emptyset; B), (\emptyset; A'), (\emptyset; A'B'), (\{A', AB'\}; B'),$$

$$(\{AB', B\}; A), (\{A'B', A\}; B'), (\{B', B\}; ff), (\{A'B', B\}; A')\}$$

The edges in a graphical representation of a proof diagram are different to those in most graph-like structures, where an edge is a directed line between two nodes. Here an edge is a set of directed lines from each member of a set of nodes to a single node. This is represented by the set of lines converging on a single arrowhead. For example, there is an edge that connects each of the assumptions $A'B'$, A to the conclusion B' .

3.3.3 Proofs of facts

This section introduces the concept of proofs of facts for proof diagrams. First it will be necessary to refer to the set of facts which occur in the proof diagram.

Definition 3.24 Let P be a proof diagram. The *set of facts which occur in P* is defined to be the set $F(P) = \{f \in \Phi | \exists (R; s) \in P \text{ such that } f \in R \text{ or } f = s\}$.

The next definition gives a way to 'throw away' structure in a proof diagram, and is essential in the definition of proof of a fact below, as it will avoid the possibility of having cycles in

proof graphs.

Definition 3.25 Let P be a proof diagram and $a \in F(P)$. The *proof diagram restricted to a* is defined as follows: $P \setminus \{a\} = \{(C; b) \in P \mid a \neq b \text{ and } a \notin C\}$.

Note here that as we allow a proof diagram P to be empty, this notion is well defined. A graphical condition which will correspond to a notion of proof of a fact will now be given. This is not meant to have a proof theoretic meaning, but to show a global dependency on the proof diagram. The definition is such that there must be some justified ‘route’ back to the axioms from the fact which is proved.

Definition 3.26 Let P be a proof diagram and $a \in F(P)$. The proposition $Proof(P, a)$ is true if and only if $(\emptyset; a) \in P$, or there is an inference $(B; a) \in P$ such that $Proof(P \setminus \{a\}, b)$ is true for all $b \in B$. $Proof(P, a)$ is false otherwise. If $Proof(P, a)$ is true, then we call P a *proof* of a .

For example, in Figure 5 one can check that S is a proof of ff . Since $(\emptyset; ff)$ is not in S , we need to check that there is an inference $(K; ff)$ in S . As $(\{B', B\}; ff) \in S$ we must verify that $S \setminus \{ff\}$ is a proof of B' and B . Here

$$T = S \setminus \{ff\} = \{(\emptyset; AB'), (\emptyset; B), (\emptyset; A'), (\emptyset; A'B'), \\ (\{A', AB'\}; B'), (\{AB', B\}; A), (\{A'B', A\}; B'), (\{B, A'B'\}; A')\}$$

T is certainly a proof of B as $(\emptyset; B) \in T$. But T must also be a proof of B' . As $(\emptyset; B')$ is not in T , then there must be an inference $(L; B')$ in T (which there is: for instance $(\{A', AB'\}; B')$) and $T \setminus \{B'\}$ must be a proof of A' and AB' .

$$U = T \setminus \{B'\} = \{(\emptyset; AB'), (\emptyset; B), (\emptyset; A'), (\emptyset; A'B'), (\{AB', B\}; A), (\{B, A'B'\}; A')\}$$

As $(\emptyset; AB')$ and $(\emptyset; A')$ are in U , U is a proof of the necessary facts and hence S is a proof of ff .

Using different implementations and different heuristics, proof procedures can generate ‘different’ (for some notion of equivalence) proofs of a fact a . Consequently, it would be useful to compare the proofs of a which are generated. One could then build heuristics by

3 AN INTRODUCTION TO PROOF DIAGRAMS

looking for common good proofs a . This means that we would like to have a concept of a proof P of a being minimal with respect to a proof B of a . This is set up in the context of proof diagrams in Section 3.3.5.

Intuitively, we are looking for a proof of a fact a which does not have a smaller proof of a 'sitting inside it'; i.e. a proof diagram in which no inference could be thrown away and a proof of a retained. Proof diagrams enable one to state global properties of what one would want a good proof of a fact to be; i.e. it respects a soundness condition (the definition of a proof P of a) and is as small as possible. This relates back to what was mentioned in Section 1.1.3. This is simply the property of Levy [81] which states that a good proof of a fact a should contain no facts which are strongly irrelevant to the proof of a . We would also like to characterise the properties of good proofs of facts from Denzinger and Schulz [35]. To do this 'useful' facts must be derived quickly. This corresponds to identifying when different facts are generated in a proof of fact a and also when new paths through the search space are taken. This motivates the following definitions.

Definition 3.27 Let P be a proof diagram and suppose $c, d \in F(P)$. The relation *is a parent of* on $F(P)$ is defined such that c is a parent of d in $F(P)$ if and only if there is an inference $(C; d) \in P$, such that $c \in C$. We also say that d is *derived* by the inference $(C; d)$.

So in the above example, looking at the proof diagram S , B is a parent of A . We can also say that B' is derived by the inference $(\{A'B', A\}; B')$.

Definition 3.28 Let P be a proof diagram. The relation *is an ancestor of* on $F(P)$ is defined to be the transitive closure of the relation 'is a parent of' on $F(P)$.

In the proof diagram S it can be said that B is an ancestor of ff . Now some obvious substructures need to be defined so that one can talk about constituent parts of a proof of a fact. It can also be seen what dependencies there are in a proof diagram.

Definition 3.29 A proof P of c is called a *subproof* of a proof Q of d if and only if

1. $c = d$ or c is an ancestor of d in Q and

2. P is a proof subdiagram of Q .

The proposition $oof(P, c, Q, d)$ is true if and only if P is a proof of c which is a subproof of a proof Q of d . It is false otherwise.

As an example we have the proof diagram W . W is a proof of A which is a subproof of the proof S of ff from Section 3.3.2.

$$W = \{(\emptyset; AB'), (\emptyset; B), (\{AB', B\}; A)\}$$

Note that if $Subproof(P, c, Q, d)$ is true it follows immediately that Q is a proof of c .

3.3.4 Proof Graphs

In this section we define a structure so that we will be able to reason about the local properties of a proof diagram. We call this structure a proof graph. We then identify certain facts appearing in the proof diagram which will be necessary in order to add a root to a proof graph.

Definition 3.30 Let P be a proof diagram. The *axiom facts of P* are defined as follows: $Ax(P) = \{a \in F(P) | (\emptyset; a) \in P\}$.

This is obviously related to the axioms In . Here, however, we refer to *facts* rather than inferences (remember: $In = \{(A; b) | A = \emptyset\}$). Now it is possible to map a proof diagram onto a graph. Then we will introduce the concept of a proof P of a being minimal with respect to a proof B of a using the notions of the proof diagram and proof graph. The definition of proof graph is very closely related to the definition of rooted DAG in Section 3.2.

Definition 3.31 A *proof graph* is a structure obtained from a proof diagram P (where $Ax(P) = \{a_1, a_2, \dots\}$) in the following way: $\mathcal{G}(P) = (F(P) \dot{\cup} \{r\}, lab, succ, r)$ such that:

- lab is a labelling function $lab : F(P) \dot{\cup} \{r\} \rightarrow \mathbb{N}$ (where the labels \mathbb{N} are the natural numbers) which assigns a label to each $f \in F(P) \dot{\cup} \{r\}$.

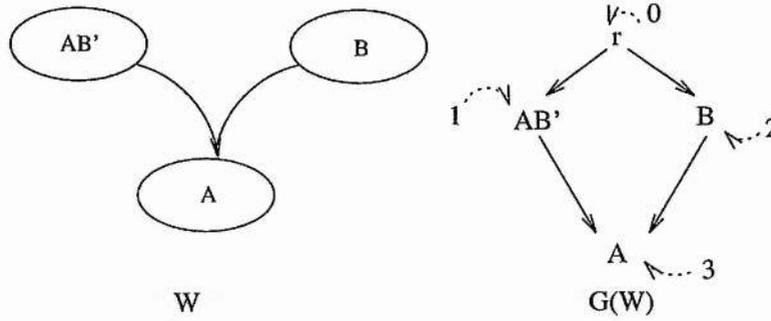


Figure 6: An example of a proof graph

- $succ$ is a function which describes ‘is a parent of’; i.e. $succ : F(P) \dot{\cup} \{r\} \rightarrow (F(P) \dot{\cup} \{r\})^*$ such that $succ(r) = (a_1 a_2 \dots)$, for each element of $Ax(P)$, and for $n \in F(P)$, $succ(n) = (m_1 m_2 \dots)$ where n is a parent of each of the m_i .

This definition may at first sight seem unintuitive as it ‘inverts’ the graph. In the typical representation of a proof tree, the ‘goal’ node is considered to be the root of the tree and so looking at proof diagrams it may seem intuitive to define this as the root node of the proof graph. There may, however, be a proof of a fact which is not an ancestor of the goal node; this happens extensively when trying to derive a certain conclusion from a set of assumptions as numerous redundant facts are generated. In this case unravelling from the root node will not result in a tree. An artificial root is introduced in this formalism as all of the proof procedures in which we are interested have one thing in common: there is a proof of a fact if there is a ‘justified’ route back to the axioms. Figure 6 gives an example of a proof graph (the proof graph of the proof diagram W from the example in Section 3.3.2).

For the purposes of this thesis the labelling is arbitrary. A possible alternative to labelling each node with a natural number would be to label each node with the formula it represents. This suggests the possibility of extending the theory by mapping facts onto facts which subsume them.

Definition 3.32 A proof diagram P is called a *valid proof* if $\forall f \in F(P)$, P is a proof of f .

Theorem 3.33 Let P be a valid proof such that $F(P)$ is a finite set. Then the proof graph $G(P) = (F(P) \dot{\cup} \{r\}, lab, succ, r)$ is a rooted graph.

3 AN INTRODUCTION TO PROOF DIAGRAMS

Proof— $\mathcal{G}(P)$ is a graph as lab is a function from $F(P) \dot{\cup} \{r\}$ to the natural numbers, $succ$ is a function from $F(P) \dot{\cup} \{r\}$ to $(F(P) \dot{\cup} \{r\})^*$ and r is in the node set. Also $F(P) \dot{\cup} \{r\}$ is a finite set as $F(P)$ is a finite set. Now it is necessary to show that it is a rooted graph:

1. Suppose there was not a path from r to some node f . Then there is an $f \in F(P) \dot{\cup} \{r\}$ such that there is no path from r to f . But P is a valid proof, and so P is a proof of f , thus there must be at least one path from f back to the $a_i \in Ax(P)$. There is a contradiction and there is a path from r to each node of $\mathcal{G}(P)$.
2. r is a source as the definition of $\mathcal{G}(P)$ shows there is no p such that there is a path from p to r .

Thus $\mathcal{G}(P)$ is a rooted graph. □

3.3.5 Further Definitions

This section introduces the notion of a proof P of a being minimal with respect to a proof B of a . This concept will use the subgraph, and so this is the next definition.

Definition 3.34 Let $G = (N, lab, succ, r)$ be a graph and let $n \in N$. Then the graph $Sub(G) = (N_1, lab_1, succ_1, n)$ is defined to be a *subgraph of G* if the following conditions hold:

- $N_1 \subseteq N$,
- lab_1 is the restriction of lab to N_1 ,
- $n \in N_1$ and
- Let $a, b \in N_1$. If b is a successor of a in $Sub(G)$ then b is a successor of a in G

The relation 'is a subgraph of' on graphs is denoted \preceq_{sg} .

We wish to be able to define the concept a good proof of a fact a . This proof of a will contain no (obviously) irrelevant information; i.e. if P is a proof of a , then $|P|$ is as small as possible. This characterises the notion of irrelevance which occurs in Levy [81]. We are,

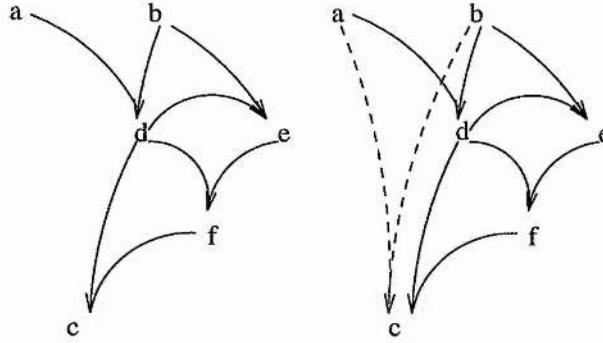


Figure 7: Figure to show the abbreviation of a proof of c

however, also interested in different ways of proving facts and so we introduce a further restriction. We illustrate the further restriction through an example. Figure 7 shows what we mean. In this example, there is a proof diagram

$$P = \{(\emptyset; a), (\emptyset; b), (\{a, b\}; d), (\{b, d\}; e), (\{d, e\}; f), (\{d, f\}; c)\}$$

It is obvious that P is a proof of c and that this proof of c depends on the facts a and b and some intermediate facts. Although these intermediate facts are necessary for the proof of c , we know that they can be generated by a and b . Thus the proof of c has been ‘abbreviated’ by adding in an inference $(\{a, b\}; c)$ (which is shown as an inference with dashed lines), and this gives us a short cut through the proof of c . It is by looking for these short cuts that we are able to build up the more complex heuristics. This notion of a short cut corresponds almost exactly to the ideas of homeomorphic embedding in Section 3.2, and so this too is incorporated into the concept of a proof P of a fact c being minimal with respect to a proof B of c . Indeed, $\{(\emptyset; a), (\emptyset; b), (\{a, b\}; c)\}$ is a proof of c which is minimal with respect to the second of the proofs of c which occur in Figure 7.

Definition 3.35 Let $Subproof(P, a, B, a)$ be true. The proof P of a is said to be *minimal with respect to* the proof B of a if and only if there is no proof Q of a which is a subproof of the proof B of a such that

- $\mathcal{G}(Q)$ is homeomorphically embedded in a subgraph of $\mathcal{G}(P)$ as graphs and
- $|Q| < |P|$.

3 AN INTRODUCTION TO PROOF DIAGRAMS

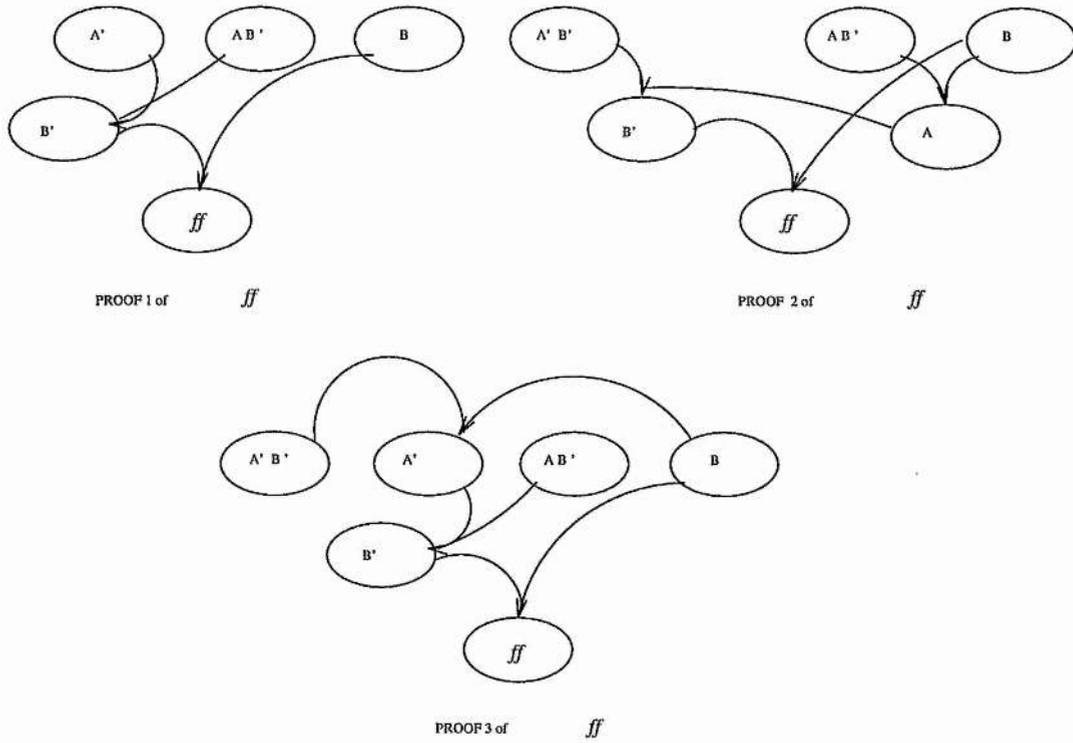


Figure 8: Proofs of ff

The proposition $Min(P, B, a)$ is true if and only if $Subproof(P, a, B, a)$ is true and P is a proof of a which is minimal with respect to a proof B of a . It is false otherwise.

Thus in Figure 8, PROOF 1 and PROOF 2 of ff are minimal with respect to the proof S of ff from Figure 5. PROOF 3 of ff is not minimal with respect to the proof S of ff

3.4 Properties of Proofs of Facts

In this section we show that when $Min(P, B, a)$ is true, the proof P of a has some desirable properties. In particular, it will be shown that there are no irrelevant inferences or facts in such a proof P of a .

3.4.1 Characterising Proofs of Facts

Given a proof B of a , there may be many different proofs P of a such that $Subproof(P, a, B, a)$ is true. The set of all such proofs of a with respect to B is now defined.

Definition 3.36 The *pedigree* of a fact a relative to a proof B of a is defined to be:

$$\mathcal{P}_B(a) = \{ \text{proof diagrams } P \mid \text{Subproof}(P, a, B, a) \text{ is true} \}$$

The next definition will be used in a number of lemmas in this section.

Definition 3.37 Suppose $P \in \mathcal{P}_B(a)$. The proposition $Small(P, B, a)$ is true if there does not exist an $R \in \mathcal{P}_B(a)$ with $|R| < |P|$ and is false otherwise. If $Small(P, B, a)$ is true, then P is called a *smallest proof of a* relative to the proof B of a .

It is obvious that a P with which $Small(P, B, a)$ is true exists. One just needs to choose a $P \in \mathcal{P}_B(a)$ with $|P|$ smallest. Now it is possible to show that there is always a proof P of a fact a which is minimal with respect to any proof of a .

Lemma 3.38 *Let B be a proof of a fact a , then there is a proof P of a such that $Min(P, B, a)$ is true. In particular, a proof P of a is minimal with respect to a proof B of a when $Small(P, B, a)$ is true.*

Proof— Choose $P \in \mathcal{P}_B(a)$ such that $Small(P, B, a)$ is true. Then P is a proof of a and there are no smaller proofs of a in $\mathcal{P}_B(a)$. By definition $Min(P, B, a)$ is true. \square

The converse of Lemma 3.38 is not true; this can be seen directly from Figure 8 in which proofs of ff of different magnitudes are presented. These proofs of ff are minimal with respect to the proof S of ff . It is now possible to show that for any proof B of a , there is a proof P of a such that $Subproof(P, a, B, a)$ is true and $\mathcal{G}(P)$ is a DAG. This will be extended later to show that the proof graphs associated with all proofs P of a when $Min(P, B, a)$ is true are DAGs.

Lemma 3.39 *Let B be a proof of a fact a and suppose that $P \in \mathcal{P}_B(a)$ such that P is a valid proof and $Small(P, B, a)$ is true. Then $\mathcal{G}(P)$ is a DAG.*

Proof— Suppose that $\mathcal{G}(P)$ is not a DAG. Then there is a fact $b \in F(P)$ such that there is a path $(\dots b \dots b \dots)$ in $\mathcal{G}(P)$. There are two cases to consider. Firstly, assume that there is an inference $(E; b) \in P$ with $b \in E$. Then, as E is non-empty, there must be some

justified route back to the axioms from the b which occurs in E . We can remove $(E; b)$ from P and retain a proof of b and a proof of a (as P was a valid proof). Thus there is a contradiction and this case cannot occur. In the proof diagram P there are thus the distinct inferences $(C; b)$ and $(D; b)$ ($C \neq D$). Let $Q = P - \{(C; b)\}$. All $f \in F(Q)$ are derived by inferences in Q , in particular b is derived by $(D; b)$. As P is a valid proof, so is Q . Q is a proof of b because Q is a proof of all $d \in D$. In particular Q is a proof of a . Thus there is a contradiction as it was supposed that there was no smaller proof of a in $\mathcal{P}_B(a)$ than the proof P of a . \square

The next lemma shows a property of ‘is a subgraph of’ which will be needed later.

Lemma 3.40 *Let P and Q be proof diagrams. If $\mathcal{G}(P) = (F(P) \dot{\cup} \{r\}, lab_0, succ_0, r)$, $P \subseteq Q$, and $\mathcal{G}(Q) = (F(Q) \dot{\cup} \{r\}, lab_1, succ_1, r)$ then $\mathcal{G}(P) \preceq_{sg} \mathcal{G}(Q)$.*

Proof– As $P \subseteq Q$, $F(P) \dot{\cup} \{r\} \subseteq F(Q) \dot{\cup} \{r\}$. It is also immediate that $r \in F(P) \dot{\cup} \{r\} \subseteq F(Q) \dot{\cup} \{r\}$. a is a successor of b in $\mathcal{G}(P)$ means that there is an inference $(B; a) \in P$ with $b \in B$. As $P \subseteq Q$, $(B; a) \in Q$ and a is a successor of b in $\mathcal{G}(Q)$. Thus $\mathcal{G}(P) \preceq_{sg} \mathcal{G}(Q)$. \square

The next theorem shows a property one would want to characterise a proof P of a when $Min(P, B, a)$ is true: one can not throw away any of the information which P contains.

Theorem 3.41 *Let $Subproof(P \dot{\cup} \{(A; b)\}, a, Q, a)$ be true. If P is also a proof of a , then the proof $P \dot{\cup} \{(A; b)\}$ of a is not minimal with respect to the proof Q of a .*

Proof– Choose a valid proof R of a such that $Small(R, P \dot{\cup} \{(A; b)\}, a)$ is true. Thus R is a proof of a which is a subproof of the proof Q of a . So $|R| < |P \dot{\cup} \{(A; b)\}|$ ($|R|$ must have at most the same magnitude as $|P|$), and $\mathcal{G}(R)$ homeomorphically embeds in a subgraph of $\mathcal{G}(P \dot{\cup} \{(A; b)\})$ (namely $\mathcal{G}(R)$ itself as $\sqsubseteq_{\mathcal{G}}$ is reflexive). Thus the proof $P \dot{\cup} \{(A; b)\}$ of a is not minimal with respect to the proof Q of a . \square

Corollary 3.42 *Let $Min(P, B, a)$ be true. Then P is a valid proof.*

Proof– Suppose that P was not a valid proof. Then there is an inference $(C; d) \in P$ where P is not a proof of d . We can therefore remove $(C; d)$ from P and retain a proof of a . Theorem 3.41 gives the result immediately. \square

3.4.2 Direct Proofs

The next definition and lemma show another desirable property of a proof P of a fact a when $Min(P, B, a)$ is true. We introduce the concept of a direct proof of a fact a ; a proof of a in which only ancestors of a are proved. If facts which are not ancestors of a were proved, there would be an obvious redundancy in the proof of a . Although these facts may be of use when building more complex proofs of facts, they have no direct relevance to the proof of a , so we wish to exclude them from what we would consider to be a good proof of a . The motivation for this is the notion of strong irrelevance from Levy [81] which was discussed in Section 1.1.3.

Definition 3.43 Let P be a proof of a fact a . P is a direct proof of a when for all $(A; b) \in P$:

1. $A = \emptyset$ and $b = a$ or
2. $b = a'$ for some a' which is an ancestor of a

Lemma 3.44 Let $Min(P, B, a)$ be true. Then P is a direct proof of a .

Proof— If P was not a direct proof of a , then there would be a fact b such that $b \in F(P)$, b is not an ancestor of a . Then there is an inference $(C; b) \in P$ and $P = Q \dot{\cup} \{(C; b)\}$. Q is a proof of every ancestor of a which occurs in the proof P of a , hence Q is a proof of a . Thus by Lemma 3.41, the proof P of a is not minimal with respect to the proof B of a and there is a contradiction. \square

The next lemma characterises the proofs of axiom facts. This lemma will be used in a number of further lemmas later in this section.

Lemma 3.45 Let $Min(P, B, a)$ be true. If $(\emptyset; a) \in P$ then $P = \{(\emptyset; a)\}$.

Proof— Suppose there is an inference $(B; c) \in P$, $B \neq \emptyset$ or $c \neq a$. This inference could be removed from P and there would still be a proof of a which is minimal with respect to the proof B of a . Hence Lemma 3.41 gives a contradiction. \square

The next lemma gives another characterisation of a proof P of a when $Min(P, B, a)$ is true: each fact is derived by exactly one inference. This will be used to help to show that

'is an ancestor of' is a strict partial order on the facts which occur in the proof diagram associated with such a proof P of a .

Lemma 3.46 *Let $Min(P, H, a)$ be true. Then each fact $c \in F(P)$ is derived by exactly one inference $(D; c)$ in P .*

Proof— If $(\emptyset; a) \in P$ then $P = \{(\emptyset; a)\}$ by Lemma 3.45. Hence there is only one inference of the form $(C; a)$ in P . Suppose that there are inferences $(A; b), (C; b) \in P$ (A or $C \neq \emptyset$) for some fact $b \in F(P)$. Now remove $(A; b)$ from the proof diagram P . There is still, however, a proof of every fact which occurs in the proof of a ; in particular b is derived by $(C; b)$. This means there is still a proof of a . Thus the proof P of a is not minimal with respect to the proof H of a by Lemma 3.41 and there is a contradiction. \square

3.4.3 Properties of Subproofs

We can now show that when $Min(P, Q, a)$ is true, the proof P of a is made up of valid, direct proofs R of b which are subproofs of the proof P of a . We show that in this case $Min(R, P, b)$ is true.

Lemma 3.47 *Let $Min(P, Q, a)$ be true. Each valid direct proof R of a fact b such that $Subproof(R, b, P, a)$ is true is minimal with respect to the proof P of b .*

Proof— By Lemma 3.46, each $f \in F(P)$ is derived by a unique inference $(C; f) \in P$. As $Subproof(R, b, P, a)$ is true, $R \subseteq P$, and clearly each $r \in F(R)$ is derived by a unique inference $(S; r) \in R$. If $(\emptyset; a) \in R$, then $P = R = \{(\emptyset; a)\}$ by Lemma 3.45, and the result follows immediately. Now look at proofs of a where $(\emptyset; a) \notin P$. As the proof is direct, there are no proofs of facts which are not ancestors of b and so if any inference is removed from R , there will be no proof of b (if this was not true, we could remove the inference from P and we would have an immediate contradiction from Lemma 3.41). Hence $Small(R, P, b)$ is true and the proof R of b is minimal with respect to the proof P of b . \square

The next theorem and corollary show that there are no cycles in $\mathcal{G}(P)$ when $Min(P, B, a)$ is true.

Theorem 3.48 *Let $Min(P, B, a)$ be true. Then a is not a parent of any $b \in F(P)$.*

Proof – If $(\emptyset; a) \in P$, then $P = \{(\emptyset; a)\}$ by Lemma 3.45. Thus a is not the ancestor of any b in $F(P)$. If $(\emptyset; a) \notin P$ and the hypothesis was not true, then there would be an inference $(A; b) \in P$ with $a \in A$. By Lemma 3.47, if $Min(P, B, a)$ is true, every valid direct proof Q of b such that $Subproof(Q, b, P, a)$ is true is minimal with respect to the proof P of b . Hence any valid direct proof of a parent of b (in particular a), is minimal with respect to the proof P of a . This is of smaller magnitude than the proof P of a and $\mathcal{G}(Q)$ is homeomorphically embedded in a subgraph of $\mathcal{G}(P)$ ($\mathcal{G}(Q)$ itself) and is a proof of a . Thus the proof P of a is not minimal with respect to the proof B of a and there is a contradiction. \square

This theorem enables us to show the following property of P when $Min(P, B, a)$ is true:

Corollary 3.49 *Let $Min(P, B, a)$ be true. Then ‘is an ancestor of’ is a strict partial order on $F(P)$.*

Proof – ‘is an ancestor of’ is defined to be transitive on $F(P)$, and the fact that it is irreflexive on $F(P)$ comes from Theorem 3.48. A fact can not appear in the proof of one of its ancestors. Hence ‘is an ancestor of’ is irreflexive on $F(P)$. \square

Corollary 3.50 *Let $Min(P, B, a)$ be true. Then $\mathcal{G}(P)$ is a proof DAG.*

Proof – $\mathcal{G}(P)$ is a proof graph with no cycles in it; thus it is a proof DAG. \square

3.5 Possible uses of Proof Diagrams

There are many ways of analysing proofs of facts through proof diagrams, and this section will discuss a number of possible approaches. In particular, we will investigate the development of useful heuristics for families of similar examples through analysing proofs of facts.

A simplistic approach to analysing proofs of a fact a is to analyse the cardinality of each proof diagram P when $Proof(P, a)$ is true. One could then argue that the ‘usefulness’

3 AN INTRODUCTION TO PROOF DIAGRAMS

of heuristics used to generate the proof diagram P can be described by comparing $|P|$ to a proof diagram $|Q|$ where $Small(Q, P, a)$ is true. This is a rather unsatisfactory method as it seems unlikely that heuristics 'learned' by this method will generalise. Choosing the smallest element of $\mathcal{P}_P(a)$ gives no intuition into the structure of the problem itself. Given a proof P of a , the proof of a chosen may be 'useful' for only this one particular example. Also, the heuristics used to generate a proof Q of a such that $Small(Q, P, a)$ is true may be extremely costly to implement.

A more interesting approach is to analyse the irrelevance of facts and inferences which occur in a proof P of a . There is already a body of work in this area, see Levy [81]. If a fact or inference is never necessary in a proof P of a , then Levy calls the fact or inference *strongly irrelevant*. If the cost of avoiding the calculation of these facts or inferences is also small, then a useful heuristic would be one in which these facts and inferences were not generated. There may be some facts and inferences which are only necessary in *some* proofs of a fact a . These facts and inferences are called *weakly irrelevant* in Levy [81]. It is these facts and inferences which may contain 'useful' information, as they may be of use in proofs of other 'useful' facts. Of course, this notion of usefulness will probably be specific to a family of examples, but if we know some information about the usefulness of weakly irrelevant facts and inferences, it may be possible to analyse the proofs of facts more meaningfully. Therefore, it may be possible to use the theory of irrelevance to analyse proofs of facts and develop new heuristics for families of proofs of facts.

One could analyse a proof P of a by analysing the proofs Q of a such that $Min(Q, P, a)$ is true. This is similar to looking at irrelevance, which was mentioned above, as it is shown in Lemma 3.44 that no strongly irrelevant information is used in such a proof Q of a . Each such proof Q of a also describes different proofs of possibly useful weakly irrelevant facts. It seems very likely that the heuristics needed to generate such a proof Q of a would be extremely costly to implement, and so this may seem an unlikely way to build useful heuristics. This approach does, however, suggest a method whereby one would look for repeated facts and patterns in a family of examples. This approach will be discussed in more detail in Section 5.4.3, and seems a likely method to generate useful heuristics for future attempts to prove facts.

3.6 Kruskal's tree theorem Revisited

This section presents a result which will show that certain classes of proofs B of a fact a have an interesting property: the set of proofs of a which are minimal with respect to a proof B of a is finite.

Note here that B can be infinite. We will look at an example in propositional resolution. Suppose we have an infinite number of propositional variables L_0, L_1, L_2, \dots . Let

$$P = \{(\emptyset; L_0), (\emptyset; L'_0), (\emptyset; L_1), (\emptyset; L'_1), \dots\}$$

In this case it is easy to see that there are going to be infinitely many proofs of ff which are minimal with respect to the proof P of ff . The restriction we need to prove this theorem is that there is no 'infinite branching' either up or down. This corresponds to restricting the number of inferences which can derive a particular fact and the number of facts a specific set of facts can derive to be finite. We will show in Chapter 4 that implementations of Knuth-Bendix completion have this property.

Lemma 3.51 *Let P be a valid proof of a such that $|P|$ is finite. Then $\mathcal{P}_P(a)$ is a finite set.*

Proof— Look at proof subdiagrams of P . As P is finite, there are a finite number of possible subsets of P and the result is immediate. \square

The following lemma shows that, with certain finiteness restrictions, there are only finitely many valid direct proofs of a fact of a given magnitude.

Lemma 3.52 *Let P be a proof of a such that:*

- *each $b \in F(P)$ is derived by a finite number of inferences $(A_i; b)$ ($i = 1, \dots, k$),*
- *for each $A \in \mathbf{P}(F(P))$ there are a finite number of b_j ($j = 1, \dots, l$) such that b_j is derived by $(A; b_j)$.*

The set of all valid direct proofs of a of cardinality m is finite.

Proof— As there are a finite number of $b_j \in F(P)$, which are derived by a particular $A \in \mathbf{P}(F(P))$ there are, in particular, a finite number of axioms (say there are q of them)

3 AN INTRODUCTION TO PROOF DIAGRAMS

$In = \{(\emptyset; b_j) | (1 \leq j \leq q)\}$. $Q = \{(\emptyset; b_n)\}$ is a valid proof of b_n of magnitude 1. As only valid direct proofs of a fact a are of interest, it is possible to constrain the proofs of facts which are of interest to ones which in the proof graph has a justified route from the axioms.

Consider what can be proved in one step, i.e. say there is a proof diagram $B = \{(\emptyset; b_p) | p = 1, \dots, q\}$; an inference $(C; d)$ will be added to B if $(C; d) \in P$ and $C \subseteq Ax(P)$. The set of such inferences will be finite: each set of parents C can occur in only a finite number of inferences, and the set $Ax(P)$ is finite. Further iterations can also be made, and at each stage only a finite number of inferences can possibly be added to the proof diagram. Thus after $m - q$ such steps the proof diagram will still be finite. No further steps need to be considered as the magnitude of the proof diagrams which are of interest is bounded by m . Lemma 3.51 gives the result immediately. \square

Theorem 3.53 *Let B be a proof of a such that*

- *each $b \in F(B)$ is derived by a finite number of inferences $(A_i; b)$ ($i = 1, \dots, k$),*
- *for each $A \in P(F(B))$ there are a finite number of b_j ($j = 1, \dots, l$) such that b_j is derived by $(A; b_j)$.*

Then the set $D \subseteq \mathcal{P}_B(a)$, such that for each $d \in D$, $Min(d, B, a)$ is true, is finite.

Proof— Consider disjoint subsets $P_1, P_2, \dots \subseteq \mathcal{P}_B(a)$ such that each $Q \in P_i$ is a proof of a (when P_i is non-empty) which is minimal with respect to the proof B of a and $|Q| = i$. So, from Corollary 3.42 and Lemma 3.44 each element of a P_i is a valid direct proof of a . Infinitely many of these subsets of $\mathcal{P}_B(a)$ are non-empty or else the theorem follows immediately from Lemma 3.52. Choose an arbitrary element from each of the non-empty sets P_i so that $p_1 \in P_1, p_2 \in P_2, \dots$ and form a sequence of the proof DAGs (these proof graphs are DAGs by Corollary 3.50). $(\mathcal{G}(p_1)\mathcal{G}(p_2)\dots)$. Suppose that this sequence is infinite. Theorem 3.15 says that $\exists i < j : \mathcal{G}(p_i) \sqsubseteq_{\mathcal{G}} \mathcal{G}(p_j)$. But now there is a contradiction: p_i and p_j are proofs of a which are subproofs of the proof B of a , $|p_i| < |p_j|$ and $\mathcal{G}(p_i)$ is homeomorphically embedded in a subgraph of $\mathcal{G}(p_j)$ (namely $\mathcal{G}(p_j)$ itself). This contradicts the fact that the proof p_j of a is minimal with respect to the proof B of a . Thus there can only be a finite number of sets of proofs of a , $P_1, P_2, \dots, P_l \subseteq \mathcal{P}_B(a)$, such that each

element of these sets is a proof of a which is minimal with respect to the proof B of a . So there are a finite number of finite sets (by Lemma 3.52) of such proofs of a and the result is immediate. \square

In practice, there are not many proofs of facts generated by a computational procedure which will have the necessary finiteness properties. In Chapter 4, though, we will show that proofs of facts in Knuth-Bendix completion can be seen to have these properties.

3.7 Conclusions

This section has given an introduction to proof diagrams. This will be developed in Chapter 5 so that a rewriting system on proofs of facts is defined. This will also enable us to search for the common substructures which correspond to heuristics and derived inferences in automated deduction. Chapter 4 looks at Knuth-Bendix completion in the setting of proof diagrams, in particular the heuristics which are called *critical pair criteria* (see, for example, Bachmair and Dershowitz [6]).

4 Completion and Proof Diagrams

4.1 Introduction

Knuth and Bendix [72] introduced a completion procedure as a method to construct a confluent and terminating rewrite system from a given set of equations and a reduction ordering. This has a number of useful applications in mathematics and computer science; see, for example, Dershowitz and Jouannaud [37]. Equations are oriented into rewrite rules, the procedure then works by checking a local confluence property: an equation is generated from a pair of rewrite rules, and if it is not 'locally confluent', the equation is added to the set of existing equations and the procedure continues iteratively. The procedure aims to construct a convergent rewrite system, and so terminates if a completeness criterion is satisfied. However, the procedure may not terminate (i.e. the procedure 'diverges') or it may fail if an equation which can not be oriented is generated. Thus Knuth-Bendix completion fits the description of forward proof procedure which were stated to be of interest in Chapter 1.

It is important to be able to construct confluent and terminating rewrite systems efficiently, if they exist at all. In any particular execution of a completion procedure there are a number of choices which have to be made, and these choices can have an effect on the efficiency of the procedure. More will be mentioned about this in Chapter 6. The choice of which critical pairs to calculate has attracted much attention in the rewriting community. Critical pair criteria are heuristics for choosing which critical pairs to calculate (see, for example, Bachmair and Dershowitz [6] and Bündgen [17]). They have been introduced as a method to check the redundancy of the critical pairs generated in a completion procedure. Omitting a redundant critical pair e is supposed to increase the efficiency of the completion procedure as no critical pairs need be calculated with an oriented version of e .

In this chapter we investigate Knuth-Bendix completion and critical pair criteria in the framework of proof diagrams in order to illustrate notions of heuristic. It is possible to generate a proof diagram from the inference rule formalism of Bachmair [5] for Knuth-Bendix completion. However, there are cycles in the associated proof graphs when this formalism is used, as an arbitrary chain of inferences may calculate the same critical pair many times. We therefore introduce a new inference rule formalism for completion which keeps track of which critical pairs are calculated. The main theorem in this chapter shows

that there are no cycles in the associated proof graphs in our new formalism. Thus repeated application of the same inference rule on the same fact is not allowed. We are also able to investigate other heuristics for completion in this framework.

The chapter is organised as follows: standard definitions for equations, rewrite rules and completion are given in Section 4.2.1, and then a modification of the inference rules of Bachmair [5] is introduced in Section 4.2.2. Critical pair criteria are investigated in Section 4.2.3, and it is shown that the modified inference system characterises them. Section 4.3 looks at properties of the proof diagrams which represent completion. A system for the set of inference rules is presented in Section 4.3.1. The main result in this section, Theorem 4.2.2 shows that, in our formulation, the relation ‘is an ancestor of’ is a strict partial order on the facts of a proof diagram which represents a completion procedure. Finally, Section 4.4 shows a number of examples of the completion procedure, and heuristics for completion are investigated in terms of proof diagrams.

4.2 Completion

Knuth-Bendix completion is introduced in the inference rule formalism of Bachmair [5]. Elementary definitions for terms, equations and rewrite rules will be given to make this chapter self contained. We introduce an extension of Bachmair’s inference rule formalism and show that this has a number of important properties: in particular, it characterises critical pair criteria.

4.2.1 Definitions

Some standard definitions of completion from Bachmair [5] are now given. Some theorems essential to the completion process, from Knuth and Bendix [72], are then stated.

Let \mathcal{F} and \mathcal{V} be two disjoint (countable) sets. We call \mathcal{F} the set of *function symbols* and \mathcal{V} the set of *variables*. Each $f \in \mathcal{F}$ is associated with a natural number, called its *arity*. Symbols $f \in \mathcal{F}$ with arity 0 are called *constants*. A *term* is a variable or an expression $f(t_1, \dots, t_n)$ where f is a function symbol of arity n and t_1, \dots, t_n are terms. The set of all terms built from function symbols in \mathcal{F} and variables \mathcal{V} is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Terms containing no variables are called *ground terms*. A term s is said to be a *subterm* of a term

t if $s = t$ or $t = f(r_1, \dots, r_n)$ and s is a subterm of one of the r_i .

A subterm of a term is referred to by its *position* (a sequence of natural numbers). The empty sequence ϵ is a position in any term, and ip is a position of a term $f(t_1, \dots, t_n)$ if and only if $1 \leq i \leq n$ and p is a position of t_i . If p is a position in a term t then the subterm $t|_p$ at position p is defined to be t if $p = \epsilon$, and $t_i|_q$ if $t = f(t_1, \dots, t_n)$ and $p = iq$ for some $1 \leq i \leq n$. $t[s]_p$ denotes the result of replacing a subterm of a term t at a position p by a term s . If $p = \epsilon$, then $t[s]_p = s$ and if $p = iq$ ($1 \leq i \leq n$) and $t = f(t_1, \dots, t_n)$ then $t[s]_p = f(t_1, \dots, t_{i-1}, t_i[s]_q, t_{i+1}, \dots, t_n)$.

A *substitution* is a mapping from variables to terms. The value of a substitution σ for a variable x is denoted $x\sigma$. The mapping σ on variables can be extended to a mapping on terms uniquely: $(f(t_1, \dots, t_n))\sigma = f(t_1\sigma, \dots, t_n\sigma)$ for all terms $f(t_1, \dots, t_n)$. The *composition* of two substitutions is defined by $t(\sigma\tau) = (t\sigma)\tau$ for all terms t . Two terms s, t are said to be *unifiable* if there is a substitution σ (called a *unifier*) such that $t\sigma = s\sigma$. A unifier σ of s and t is said to be most general (denoted by m.g.u. for *most general unifier*) if for every unifier τ of s and t there exists a substitution τ' such that $(x\sigma)\tau' = x\tau$ for all variables x . If two terms are unifiable, then they have a most general unifier, which is unique up to a renaming of variables (see Robinson [107]): we will assume this henceforth.

An equation is a pair of terms, written $s = t$. Henceforth $s = t$ and $t = s$ will be considered to be the same equation. The set of equations built from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted $Eq(\mathcal{T}(\mathcal{F}, \mathcal{V}))$. A binary relation \rightarrow on terms is called *monotonic* if $s \rightarrow t \Rightarrow u[s]_p \rightarrow u[t]_p$, for all terms s, t and u and positions p in u . It is *stable* under substitution if $s \rightarrow t \Rightarrow s\sigma \rightarrow t\sigma$ for any substitution σ . \rightarrow^+ denotes the transitive closure of \rightarrow , \rightarrow^* the transitive, reflexive closure and \leftrightarrow the symmetric closure. The symmetric, reflexive, transitive closure of \rightarrow is denoted by \leftrightarrow^* . For any set of equations E we denote by \leftrightarrow_E the smallest symmetric relation that contains E and is stable and monotonic.

Directed equations are called *rewrite rules* and are written $s \rightarrow t$. The set of rewrite rules built from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted $\overline{RR}(\mathcal{T}(\mathcal{F}, \mathcal{V}))$. A *rewrite system* is any set of rewrite rules R in which the variables on the right hand side also appear on the left. The *rewrite relation* \rightarrow_R on terms is the smallest stable and monotonic relation which contains R ; i.e. $s \rightarrow_R t$ (s rewrites to t) means that there exists a term w , a position p in w , a substitution σ and $u \rightarrow v \in R$ such that $s = w[u\sigma]_p$ and $t = w[v\sigma]_p$. A term s is said to be in *normal form*

4 COMPLETION AND PROOF DIAGRAMS

with respect to R if there is no term t such that $s \rightarrow_R t$. Rewriting a term t into a term u using a rewrite rule $l \rightarrow r$ at position p with substitution σ is denoted $t \rightarrow_{l \rightarrow r}^{p, \sigma} u$.

A relation \rightarrow on terms is said to be *Church-Rosser* (or *confluent*) if, for any two elements s and t such that $s \leftrightarrow^* t$, there is an element v such that $s \rightarrow^* v$ and $t \rightarrow^* v$. It is said to be *weakly Church-Rosser* (or *weakly confluent*) if for any term t such that $s \leftarrow t \rightarrow u$, then there is a term v such that $s \rightarrow^* v \leftarrow^* u$. The relation is said to be *terminating* if there are no infinite sequences $t \rightarrow t_1 \rightarrow \dots$. The relation \rightarrow on terms is said to be *convergent* if it is both terminating and Church-Rosser. A rewrite system R is said to be *terminating* if and only if the associated rewrite relation \rightarrow_R is terminating and is said to be *convergent* if and only if the associated rewrite relation \rightarrow_R is convergent.

Definition 4.1 Let $u = v$ be an equation, and R be a rewriting system. Then $u = v$ is said to be *redundant* with respect to R if there is a term t such that $u \rightarrow_R^* t \leftarrow_R^* v$.

A transitive terminating relation is said to be *well-founded*. A reduction ordering on terms is a well-founded ordering on terms which is stable and monotonic.

Now the inference system of Bachmair [5] in the form of Comon [28] will be presented. An inference rule here is a binary relation on tuples of equations and rewrite rules. In the remainder of Section 4.2.1, it will be assumed that all equations are built from terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$. In what follows it will be assumed that E is a set of equations, R is a set of rewrite rules and $>$ is a reduction ordering on $\mathcal{T}(\mathcal{F}, \mathcal{V})$. (E, R) denotes the pair of sets E and R .

Orient	$\frac{E \dot{\cup} \{l = r\}, R}{E, R \cup \{l \rightarrow r\}}$	If $l > r$
Deduce	$\frac{E, R}{E \cup \{l[d]_p \sigma = r \sigma\}, R}$	If $l \rightarrow r, g \rightarrow d \in R, p$ is a non-variable position of l, σ is the m.g.u. of $l _p$ and g ,
Compose	$\frac{E, R \dot{\cup} \{l \rightarrow r\}}{E, R \cup \{l \rightarrow r'\}}$	If $r \rightarrow_R^+ r'$
Simplify	$\frac{E \dot{\cup} \{s = t\}, R}{E \cup \{s = t'\}, R}$	If $t \rightarrow_R^+ t'$
Delete	$\frac{E \dot{\cup} \{s = s\}, R}{E, R}$	
Collapse	$\frac{E, R \dot{\cup} \{l \rightarrow r\}}{E \cup \{l' = r\}, R}$	If $l \rightarrow_{g \rightarrow d}^{p, \sigma} l'$ with $g \rightarrow d \in R$, and either $p \neq \epsilon, g \neq d$ up to renaming, or $r > d$

Given an inference system \mathcal{J} , $A \vdash_{\mathcal{J}} B$ denotes that B can be obtained from A by an application of one or more of the inference rules in \mathcal{J} . A sequence $A_0 \vdash_{\mathcal{J}} A_1 \vdash_{\mathcal{J}} \dots$ is called a *derivation* in \mathcal{J} from A_0 .

Let E be a set of equations, R be a rewrite system and $>$ be a reduction ordering. A *proof* of an equation $s = t$ from (E, R) and $>$ is a sequence $P = (s_1 s_2 \dots s_n)$ such that $s_1 = s$, $s_n = t$ and for $1 < i \leq n$ one of $s_{i-1} \leftrightarrow_E s_i$, $s_{i-1} \rightarrow_R s_i$, or $s_{i-1} \leftarrow_R s_i$ holds. We say that $s = t$ is *provable* from (E, R) and $>$ if there exists a proof of $s = t$ from (E, R) and $>$.

Note here that in Bachmair [5] it is proved that, given a reduction ordering $>$, if $(E, R) \vdash_{\mathcal{TC}} (E_1, R_1)$, then an equation $s = t$ is provable from (E, R) and $>$ if and only if $s = t$ is provable from (E_1, R_1) and $>$.

Let $s \rightarrow t$ and $u \rightarrow v$ be rewrite rules with no variables in common (rename if necessary), and suppose that some non-variable subterm $s|_p$ of s is unifiable with u , σ being the most general unifier. The *superposition* of $u \rightarrow v$ on $s \rightarrow t$ at position p in s determines the *critical pair* $t\sigma = s\sigma[v\sigma]_p$. $CP(R)$ denotes the set of all critical pairs between rewrite rules in R .

Definition 4.2 By a *completion procedure* we mean a program which accepts as input a pair (E, \emptyset) and a reduction ordering $>$ and generates a derivation in \mathcal{I} from (E, \emptyset) and $>$. Suppose that there is a derivation $(E, \emptyset) \vdash_{\mathcal{I}} (E_0, R_0) \vdash_{\mathcal{I}} \dots$ in \mathcal{I} from (E, \emptyset) and a reduction ordering $>$. If the pair (E_i, R_i) occurs in the sequence, where $E_i = \emptyset$ and each element of $CP(R_i)$ is redundant with respect to R_i , the derivation is said to *succeed*. It is said to fail otherwise.

There are a number of things to note here. Firstly, we can think of a derivation in \mathcal{I} from (E, \emptyset) and $>$ as being an instance of a completion procedure; (E, \emptyset) and $>$ are actual arguments for a completion procedure. Also note that the R_i are indeed rewrite systems; i.e. they consist of rewrite rules. The only way new elements of R_i can be formed is through the Orient and Compose inference rules. The side conditions on these rules mean that the elements of R_i obey the necessary conditions to be a rewrite rule.

Now we go back to Knuth and Bendix [72] and Bachmair [5] (where the proofs can be found) for some lemmas and a theorem which are necessary to prove the convergence of a rewrite system:

Lemma 4.3 Newman's lemma

If a rewrite relation is weakly Church-Rosser and terminating, then it is Church-Rosser.

Theorem 4.4 *A terminating rewrite system R is convergent if and only if each element of $CP(R)$ is redundant with respect to R .*

Thus if there is a derivation $(E, \emptyset) \vdash_{\mathcal{I}} (E_0, R_0) \vdash_{\mathcal{I}} \cdots \vdash_{\mathcal{I}} (\emptyset, R)$ in \mathcal{I} from (E, \emptyset) and a reduction ordering $>$ where each element of $CP(R)$ is redundant with respect to R , then R is a convergent system. It follows from Bachmair [5] that if the above derivation succeeds at (\emptyset, R) , then R is a convergent rewriting system which can decide when an equation is provable from (E, \emptyset) .

In this chapter, all the examples which will be used will come from string completion, which is not immediately covered by the above definitions. It is well known, however, that any problem in string rewriting can be translated into a problem about monadic terms (see Chapter 6). Indeed, this is the basis for implementations of special purpose string completion systems for semigroup and group theory.

4.2.2 Extended Inference Rules for Completion

This section will look at another formulation of completion which we will call c-completion. An extension of the inference rules of Bachmair [5] is presented. Some elementary definitions and results will now be given.

In what follows all equations and rewrite rules will be built from terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$, E will be a set of equations, R will be a set of rewrite rules and C will be a possibly infinite set of equations. Also suppose that $>$ is some reduction ordering on the terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The set of extended inference rules for c-completion (which will be denoted by \mathcal{IC}) is as follows:

$$\text{Orient} \quad \frac{E \dot{\cup} \{l = r\}, R, C}{E, R \cup \{l \rightarrow r\}, C} \quad \text{If } l > r$$

4 COMPLETION AND PROOF DIAGRAMS

Deduce	$\frac{E, R, C}{E \cup \{z\}, R, C \cup \{z\}}$	$z \equiv l[d]_p \sigma = r \sigma \notin C; l \rightarrow r, g \rightarrow d \in R,$ p a non-variable pos. of l, σ m.g.u. of $l _p$ and g
Compose	$\frac{E, R \cup \{l \rightarrow r\}, C}{E, R \cup \{l \rightarrow r'\}, C}$	If $r \rightarrow_R^+ r'$
Simplify	$\frac{E \cup \{s = t\}, R, C}{E \cup \{s = t'\}, R, C}$	If $t \rightarrow_R^+ t'$
Delete	$\frac{E \cup \{s = s\}, R, C}{E, R, C}$	
Collapse	$\frac{E, R \cup \{l \rightarrow r\}, C}{E \cup \{l' = r\}, R, C}$	If $l \rightarrow_{g \rightarrow d}^{p, \sigma} l'$ with $g \rightarrow d \in R$, and either $p \neq \epsilon, g \neq d$ up to renaming, or $r > d$

The rules here are different to Bachmair's rules as they include the sets C . These 'count' the new critical pairs which are calculated, and preclude the generation of critical pairs which already occur in C . We will show in Section 4.2.3 that these sets mimic complex heuristics, called critical pair criteria, in completion.

We have the following definition which is similar to the definition of completion procedure in Section 4.2.1

Definition 4.5 A *c-completion procedure* is a program which accepts as input a triple (E, \emptyset, C) and a reduction ordering $>$ and generates a derivation in \mathcal{IC} from (E, \emptyset, C) and $>$. Suppose there is a derivation

$$(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} (E_1, R_1, C_1) \vdash_{\mathcal{IC}} \dots$$

in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$. If the triple (E_i, R_i, C_i) occurs in the sequence, such that $E_i = \emptyset$ and $CP(R_i) \subseteq C_i$, the derivation is said to *c-succeed* with respect to C_i . It is said to fail otherwise.

The definition of c-success of a derivation in \mathcal{IC} from (E_i, R_i, C_i) and a reduction ordering $>$ is dependent on the sets C_i . Note that $C \subseteq C_i$ for all i , and so the choice of C is very important. This is because the side condition on the Deduce inference rule precludes the generation of a critical pair equation if it already occurs in C . For the purposes of this thesis, C is an arbitrary set of equations unless explicitly stated. A theorem is now presented to show that the set C characterises desirable properties: e.g. when each element of C is an equation which is provably redundant.

Theorem 4.6 *Let there be a derivation $(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} \dots$ in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$. Also let there be a triple (E_i, R_i, C_i) such that the derivation c -succeeds with respect to C_i where each $c \in C$ is redundant with respect to R_i . Then R_i is a convergent rewriting system.*

Proof— R_i is terminating as $>$ is a reduction ordering, which means that the associated rewrite relation \rightarrow_{R_i} is terminating. It is therefore sufficient to show that R_i is Church-Rosser. In fact, it is only necessary to show a local Church-Rosser condition by Theorem 4.4; i.e. it is necessary to show that all critical pairs between elements of R_i have been calculated (or are ‘known about’) and that each of these critical pairs can be rewritten and then deleted. As the derivation c -succeeds with respect to C_i , $E_i = \emptyset$ and $CP(R_i) \subseteq C_i$. If an element of $CP(R_i)$ is in $C_i - C$, then it has at some stage been processed (as $E_i = \emptyset$), and so we only need to check what happens when an element of $CP(R_i)$ is in C . As each equation in this set is redundant with respect to R_i , the condition is fulfilled immediately. \square

The property $CP(R_i) \subseteq C_i$ is similar to Theorem 4.4 which states that a local confluence check is sufficient to check that a rewrite system is Church-Rosser if it is known that the rewrite relation is contained in some reduction ordering. If one has a triple (E_i, R_i, C_i) and a reduction ordering $>$, then certain critical pairs between elements of an R_i may be precluded from being computed: this is exactly what the side condition on the Deduce inference rule in \mathcal{IC} is saying. So if it can be proved that the critical pairs which are being precluded are in fact redundant in some way, then a useful heuristic which may help a c -completion procedure will have been defined. This is exactly the notion of the critical pair criteria which will be studied in more depth in Section 4.2.3.

Now a theorem which relates inference system \mathcal{IC} to inference system \mathcal{I} will be proved.

Theorem 4.7 *Suppose that there is a derivation*

$$(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} (E_1, R_1, C_1) \vdash_{\mathcal{IC}} \dots$$

in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$. Then there exists a derivation

$$(E, \emptyset) \vdash_{\mathcal{I}} (E_0, R_0) \vdash_{\mathcal{I}} (E_1, R_1) \vdash_{\mathcal{I}} \dots$$

in \mathcal{I} from (E, \emptyset) and $>$.

4 COMPLETION AND PROOF DIAGRAMS

Proof – There is an obvious correspondence between the inference rules in the system \mathcal{IC} and the inference system \mathcal{I} . It can easily be seen that the Orient, Delete, Compose, Simplify and Collapse inference rules in \mathcal{IC} manipulate the sets E_i and R_i in exactly the same way as the corresponding sets are manipulated by the inference system \mathcal{I} . It is thus necessary to concentrate on the inference rule Deduce. In \mathcal{IC} Deduce is restricted compared to the corresponding rule in \mathcal{I} , as no equation which occurs in the set C_i can be generated by application of the Deduce inference rule. Given (E_j, R_j, C_j) , then every equation obtained by application of Deduce in \mathcal{IC} can be obtained by application of the inference rule Deduce in \mathcal{I} to (E_j, R_j) . Thus each application of an inference rule from \mathcal{IC} to a triple (E_k, R_k, C_k) can be mimicked by an application of an inference rule from \mathcal{I} to a pair (E_k, R_k) .

□

Of course, the converse of this theorem is not true. Each application of an inference rule from \mathcal{I} does not necessarily correspond to an application of an inference rule from \mathcal{IC} , as the Deduce inference rule is restricted by the condition on the sets C_i .

We now say that an equation $s = t$ is *provable* from (E, R, C) and a reduction ordering $>$ if and only if $s = t$ is provable from (E, R) and $>$. Thus it follows from Bachmair [5] and the theorem above that if $(E, R, C) \vdash_{\mathcal{IC}} (E_1, R_1, C_1)$, then an equation $s = t$ is provable from (E, R, C) and a reduction ordering $>$ if and only if $s = t$ is provable from (E_1, R_1, C_1) and $>$.

Corollary 4.8 *Let there be a derivation $(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} \dots$ in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$. Also let there be a triple (E_i, R_i, C_i) such that the derivation c -succeeds with respect to C_i where each $c \in C$ is redundant with respect to R_i . Then an equation $s = t$ is provable from (\emptyset, R_i, C_i) and $>$ if and only if $s = t$ is provable from (E, \emptyset, C) .*

Proof – Follows from Theorem 4.6 and discussion above.

□

4.2.3 Critical Pair Criteria

The efficiency of a c -completion procedure is dependant on a number of considerations, two of which are the number of equations and rewrite rules generated and how quickly certain

'essential' equations and rules are generated. The first of these questions can be addressed by looking at critical pair criteria. These criteria preclude the generation of certain critical pairs which are provably redundant. If these equations had been generated, then the potentially costly process of normalisation and deletion would have to occur. Of course a critical pair criterion could make a c-completion procedure more inefficient. The costliness of the normalisation and deletion process has to be measured against the costliness of applying the criterion. It can also be seen that, even though the equations which are precluded are redundant, they may be applied in an intermediate stage of deriving one of the useful equations or rewrite rules. This section will investigate how critical pair criteria fit into the theory of the inference system \mathcal{IC} .

Early critical pair criteria were based on ideas from Gröbner basis theory, which has been shown to have many links to completion (see, for example, Bündgen [17]). Criteria which have been developed include the *connectedness criteria* in which a 'smaller' proof of a critical pair exists. These have been developed and refined by Küchlin [76] and Winkler [125].

In Bachmair and Dershowitz [6] a characterisation of critical pair criteria is given in terms of *proof orderings*. These orderings make it possible to show that no information is 'lost' when a critical pair is precluded. In this formulation, critical pair criteria are introduced as 'elimination patterns'; i.e. proofs containing these patterns can be transformed into simpler proofs. We, however, take the approach of Bündgen [17] and Zhang and Kapur [127]. This formulation has the advantage that the intuition for the definitions is much clearer than that of Bachmair and Dershowitz [6]. We use and adapt definitions from Bündgen [17].

Definition 4.9 A *critical pair criterion* is a mapping on sets of equations:

$$CPC : \mathbf{P}(Eq(\mathcal{T}(\mathcal{F}, \mathcal{V}))) \longrightarrow \mathbf{P}(Eq(\mathcal{T}(\mathcal{F}, \mathcal{V})))$$

Intuitively a critical pair criterion is a heuristic which indicates a set of equations which we omit during a c-completion procedure; i.e. the equations precluded by the criterion are meant to be irrelevant in the c-completion procedure. For example, an obvious critical pair criterion could indicate a set of equations which are formed by looking at disjoint critical pairs (this is analogous to Buchberger's first criterion from Gröbner basis theory; see, for instance, Cox *et al* [33]). As an example of this, suppose we had the rules $bb \rightarrow b$ and

4 COMPLETION AND PROOF DIAGRAMS

$aa \rightarrow a$, then a disjoint overlap of these two rules would reduce in the following way: $baa \leftarrow bbaa \rightarrow bba$. But we could rewrite the equation $bba = baa$ to the equation $ba = ba$ in two rewrite steps. It is easy to see that any such overlaps will give rise to redundant equations. The next definition is what characterises the use of a critical pair criterion.

Definition 4.10 Given a finite set of equations $E \subseteq Eq(\mathcal{T}(\mathcal{F}, \mathcal{V}))$, a possibly infinite set of equations $C \subseteq Eq(\mathcal{T}(\mathcal{F}, \mathcal{V}))$ and a reduction ordering $>$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$, a critical pair criterion CPC is said to be *correct* with respect to E , C and $>$ if and only if there is a derivation

$$(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} \dots \vdash_{\mathcal{IC}} (E_j, R_j, C_j)$$

in \mathcal{IC} from (E, \emptyset, C) and $>$ satisfying

1. the derivation in \mathcal{IC} from (E, \emptyset, C) and $>$ succeeds with respect to C_j ; i.e. $E_j = \emptyset$ and $CP(R_j) \subseteq C_j$
2. each element of $CPC(E)$ is redundant with respect to R_j

It is thus the correctness property which characterises the use of critical pair criteria. It will now be shown that this characterisation is exactly mimicked by the set C in a derivation in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$.

Corollary 4.11 *Suppose we have a derivation*

$$(E, \emptyset, C) \vdash_{\mathcal{IC}} (E_0, R_0, C_0) \vdash_{\mathcal{IC}} \dots \vdash_{\mathcal{IC}} (E_j, R_j, C_j)$$

in \mathcal{IC} from (E, \emptyset, C) and a reduction ordering $>$ which c -succeeds with respect to C_j . Also let $C = CPC(E)$, where CPC is a critical pair criterion and each element of $CPC(E)$ is redundant with respect to R_j . Then an equation $s = t$ is provable from (\emptyset, R_i, C_i) and $>$ if and only if $s = t$ is provable from (E, \emptyset, C) .

Proof – Follows directly from Corollary 4.8. □

4.3 Properties of Proof Diagrams in C-Completion

In this section we relate executions of c-completion procedures to proof diagrams. A number of properties of these proof diagrams will be investigated.

4.3.1 A System for C-Completion and Generating Proof Diagrams

We first define a system for c-completion. We also introduce an example which will be used later in this chapter. This will be so that examples of heuristics for c-completion can be discussed in Section 4.4.

Henceforth all equations and rewrite rules will be built from $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Also in what follows, E will be a finite set of equations, R will be a set of rewrite rules, C will be a possibly infinite set of equations and $>$ will be a reduction ordering on $\mathcal{T}(\mathcal{F}, \mathcal{V})$. A system $Syst = (Symb, \Phi, V, \approx)$ (from Definition 3.16) can be defined in the following way:

- $\Phi = \{(E, R, C) \mid E, C \in \mathbf{P}(Eq(\mathcal{T}(\mathcal{F}, \mathcal{V}))), E \text{ is finite, } C \text{ is possibly infinite, } R \in \mathbf{P}(\overline{RR}(\mathcal{T}(\mathcal{F}, \mathcal{V})))\}$
- V , the set of inferences, is the sets of all possible instantiations of applications of the inference rules, and can be described in the following way:

Orient	$\{(E \dot{\cup} \{l = r\}, R, C)\}; (E, R \cup \{l \rightarrow r\}, C), \text{ If } l > r$
Deduce	$\{(E, R, C)\}; (E \cup \{z\}, R, C \cup \{z\}),$ $z \equiv l[d]_p \sigma = r \sigma \notin C; l \rightarrow r, g \rightarrow d \in R,$ $p \text{ a non-variable pos. of } l, \sigma \text{ m.g.u. of } l _p \text{ and } g$
Compose	$\{(E, R \dot{\cup} \{l \rightarrow r\}, C)\}; (E, R \cup \{l \rightarrow r'\}, C), \text{ If } r \rightarrow_R^+ r'$
Simplify	$\{(E \dot{\cup} \{s = t\}, R, C)\}; (E \cup \{s = t'\}, R, C), \text{ If } t \rightarrow_R^+ t'$
Delete	$\{(E \dot{\cup} \{s = s\}, R, C)\}; (E, R, C),$
Collapse	$\{(E, R \dot{\cup} \{l \rightarrow r\}, C)\}; (E \cup \{l' = r\}, R, C).$ $\text{If } l \xrightarrow{p, \sigma}_{g \rightarrow d} l' \text{ with } g \rightarrow d \in R, \text{ and either}$ $p \neq \epsilon, g \neq d \text{ up to renaming, or } r > d$

Each inference in V represents an application of one of the inference rules \mathcal{IC} .

- Now it is necessary to define an equivalence relation \approx on Φ . This can be done by considering the equality of the sets of equations, rewrite rules and critical pairs; i.e. $(E_m, R_m, C_m) \approx (E_n, R_n, C_n)$ if and only if $E_m = E_n, R_m = R_n$ and $C_m = C_n$.

The next definition will show what is meant by 'generating' a proof diagram for the c-completion procedure. This proof diagram represents an instance of the c-completion procedure. This is very similar to the definitions for c-completion procedure and the c-success of a derivation in \mathcal{IC} from (E, \emptyset, C) and reduction ordering $>$ with respect to a C_i .

Definition 4.12 A proof diagram

$$P = \{(\emptyset; (E, \emptyset, C)), (\{(E, \emptyset, C)\}; (E_0, R_0, C_0)), (\{(E_0, R_0, C_0)\}; (E_1, R_1, C_1)) \dots\}$$

4 COMPLETION AND PROOF DIAGRAMS

is said to be *generated* from an axiom $(\emptyset; (E, \emptyset, C))$ and a reduction ordering $>$ if:

- E is a set of equations built from $\mathcal{T}(\mathcal{F}, \mathcal{V})$
- C is a possibly infinite set of equations built from $\mathcal{T}(\mathcal{F}, \mathcal{V})$
- $>$ is a reduction ordering on $\mathcal{T}(\mathcal{F}, \mathcal{V})$

and each $(E_{i+1}, R_{i+1}, C_{i+1})$ ($i \in \mathbb{N}$) can be obtained from (E_i, R_i, C_i) by application of an inference rule from \mathcal{IC} and $>$. P is said to *c-succeed* if there is $(\emptyset, R_i, C_i) \in F(P)$ such that $CP(R_i) \subseteq C_i$

The fact that the disjoint union is being used in the premise of the inference rules in \mathcal{IC} causes an interesting phenomenon. If $(E_i, R_i, C_i) \vdash_{\mathcal{IC}} (E_{i+1}, R_{i+1}, C_{i+1})$ where one of Orient, Compose, Simplify, Delete or Collapse is applied, then $(E_{i+1}, R_{i+1}, C_{i+1}) \not\approx (E_i, R_i, C_i)$ (i.e. $E_{i+1} = E_i$, $R_{i+1} = R_i$ and $C_{i+1} = C_i$ is false). Thus application of these inference rules never leaves the triple (E, R, C) unchanged. It can also be seen that the Deduce inference rule also has this property: this is due to the introduction of the sets C_i . A critical pair is only calculated if it does not already exist in a C_i which has been calculated before this point; i.e. only ‘new’ critical pairs are calculated, and need not be calculated again in the future. This, in essence, is what happens in a critical pair criterion. In Bachmair [5] and Bündgen [17], for example, this is done by restricting to ‘superposition patterns’ which can create certain kinds of critical pairs. The restriction means that certain constraints are put on the Deduce inference rule; to express these would, however, require considerable modification of our notation. Thus each application of an inference rule gains more knowledge. Removal of non-inferences in the realms of c-completion is obviously desirable, as otherwise these non-inferences could be done exclusively, and no progress made in the procedure. The following definition and lemma arise from the above discussion.

Definition 4.13 If the proof diagram P contains no inferences of the form $(A; a)$, such that $a \in A$, then P is called a *faithful* proof diagram.

Lemma 4.14 Let P be generated by axiom $(\emptyset; (E, \emptyset, C))$ and the reduction ordering $>$. Then P is a *faithful proof diagram*.

4 COMPLETION AND PROOF DIAGRAMS

This property means that there can be no cycles of size one in the proof graph $\mathcal{G}(P)$, where P is generated by axiom $(\emptyset; (E, \emptyset, C))$ and the reduction ordering $>$.

Example 4.15 Assume that the axiom $\alpha = (\emptyset; (E, \emptyset, C))$ is given, where $E = \{ba = ab, aab = b, abb = a\}$, and that $C = \emptyset$. The reduction ordering $>_t$ we choose to use is the length then lexicographic ordering with $b > a$. Figure 9 gives a graphical representation of a proof diagram P generated by α and $>_t$.

There are a number of properties of the proof diagram P which are of interest. Firstly, two facts have been marked on the proof diagram by $(*)$ and $(**)$. One can check to see if P c-succeeds with respect to C_i where $(*) = (E_i, R_i, C_i)$. As $E_i = \emptyset$ it must be checked to see if $CP(R_i) \subseteq C_i$ is true. It is easy to check, however, that there is a critical pair between $ba \rightarrow ab$ and $abb \rightarrow a$ which has not been calculated at this point. Thus, up to $(*)$, P does not c-succeed with respect to C_i . The critical pair is calculated after fact $(*)$ and this is then rewritten and deleted. Thus P c-succeeds with respect to C_j where $(**) = (E_j, R_j, C_j)$.

It also appears that there is redundancy in this proof diagram. For instance, the facts which have a dotted box around them are in some sense redundant. The critical pairs which are calculated in this portion of the proof diagram are immediately normalised and deleted. It should be noted, however, that these critical pairs have to be calculated (or at least 'known about') for P to c-succeed. Thus the facts can not be removed arbitrarily. This sort of heuristic was discussed in terms of critical pair criteria in Section 4.2.3.

The final thing to note is that the graph of the associated proof graph is linear: i.e. there is a total order on the nodes. It will be shown that the proof diagrams generated by a fact in general have 'is an ancestor of' as a strict partial order on $F(P)$ in Section 4.3.3. In our example this means that the graph is a line.

4.3.2 Properties of C-Completion

The following proposition makes it possible to refer to proofs of finite, convergent rewriting systems in the context of proof diagrams.

Definition 4.16 Let P be a proof diagram generated from $(\emptyset; (E, \emptyset, C))$ and the reduction ordering $>$ by application of the inference rules \mathcal{IC} . The proposition $KB(P, a)$ is true if:

4 COMPLETION AND PROOF DIAGRAMS

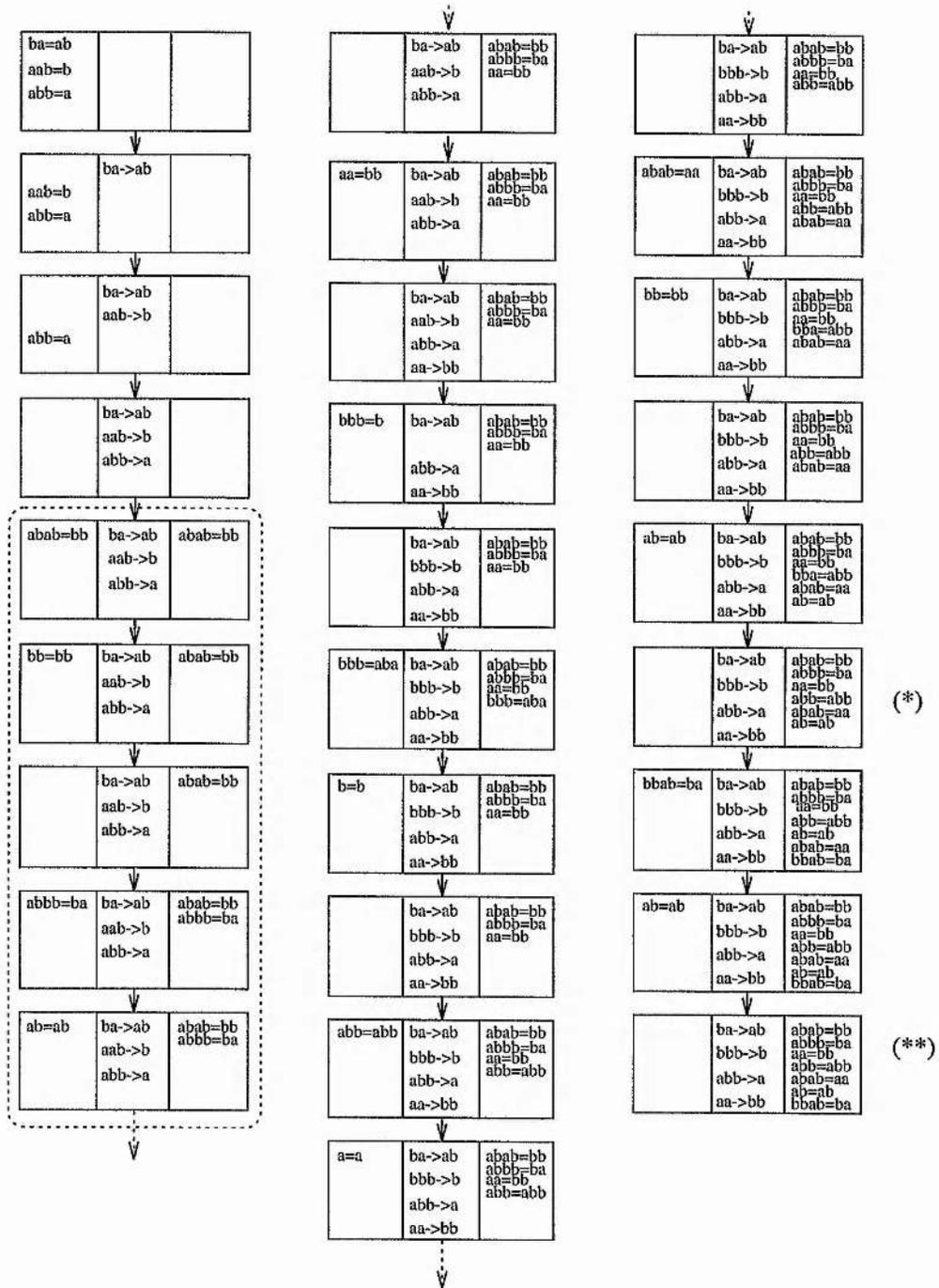


Figure 9: Proof diagram P generated by α and $>$

4 COMPLETION AND PROOF DIAGRAMS

P is a proof of $a = (\emptyset, R_i, C_i)$ where

- P c-succeeds with respect to C_i ; i.e. $CP(R_i) \subseteq C_i$

$KB(P, a)$ is false otherwise.

In Example 4.15, $KB(P, (**))$ is true as all critical pairs have been calculated between the finite number of elements in the final set of rewrite rules. Thus this set is finite and convergent and the set of equations is empty.

In a computer implementation of a c-completion procedure, if the c-completion procedure c-succeeds with respect to a C_i given a finite set of equations and a finite set of rewrite rules, only a finite number of manipulations of these sets can occur on application of an inference rule. The only problem is when unification is used. As we are working in first order theories, if a unifier exists, then there is a most general unifier which is unique up to a renaming of variables. Thus there are only finitely many distinct outcomes of applying the inference rule. We therefore only consider such ‘useful’ applications of the inference rules. The following lemma follows from the discussion above:

Lemma 4.17 *Let P be a proof diagram generated from $(\emptyset; (E, \emptyset, C))$ and the reduction ordering $>$ by application of the inference rules \mathcal{IC} . Then for each $a \in F(P)$ there are a finite number of $b \in F(P)$ such that a is a parent of b in P .*

Corollary 4.18 *Let $KB(P, a)$ be true. Then there are only finitely many $M \in \mathcal{P}_P(a)$, such that M is a proof of a , which are minimal with respect to the proof P of a .*

Proof– There is a single axiom $(\emptyset; (E, \emptyset, C))$. Theorem 3.53 gives the result immediately.

□

4.3.3 Orderings, C-Completion and Proof Diagrams

Now it is necessary to analyse the inference rules \mathcal{IC} . Suppose that

$$(E_i, R_i, C_i) \vdash_{\mathcal{IC}} (E_{i+1}, R_{i+1}, C_{i+1})$$

4 COMPLETION AND PROOF DIAGRAMS

We will study how the cardinality of the sets of equations and rules changes when each inference rule is applied to (E_i, R_i, C_i) . This will be used in this section to prove that the relation 'is an ancestor of' is a strict partial order on the facts of the proof diagram.

Notation 4.19 $|E|$ and $|R|$ denote the cardinality of the sets E, R and in a fact (E, R, C) . $|(E, R)|$ denotes $|E| + |R|$.

It can be seen directly from the definition of the inference system \mathcal{IC} that certain applications of the inference rules correspond to changes in the size of the sets E, R .

Lemma 4.20 Let P be a proof diagram generated by (E, \emptyset, C) and $>$, and suppose that $(\{a = (E_i, R_i, C_i)\}; b = (E_{i+1}, R_{i+1}, C_{i+1})) \in P$. Then:

1. if $|(E_{i+1}, R_{i+1})| > |(E_i, R_i)|$, b is obtained from a by application of Deduce
2. if $|R_{i+1}| > |R_i|$, b is obtained from a by application of Orient

It can also be seen that it is possible to trace some applications of the inference rules \mathcal{IC} by looking at the sets C_i . The following lemma characterises this, and follows immediately from the definition of the inference system \mathcal{IC} .

Lemma 4.21 Let P be a proof diagram generated by (E, \emptyset, C) and $>$, and suppose that $(\{a = (E_i, R_i, C_i)\}; b = (E_{i+1}, R_{i+1}, C_{i+1})) \in P$. Then:

1. if $C_{i+1} \supset C_i$, b is obtained from a by application of Deduce
2. $C_i \not\supset C_{i+1}$,

Theorem 4.22 Let P be a proof diagram generated by $\alpha = (E, \emptyset, C)$ and $>$. Then 'is an ancestor of' is a strict partial order on $F(P)$.

Proof – 'is an ancestor of' is a transitive relation on $F(P)$ so it is sufficient to show that that this relation is irreflexive on $F(P)$; i.e. there are no cycles in $\mathcal{G}(P)$. By Lemma 4.14 all proof diagrams which are being investigated are faithful, so there are no cycles of the form $(\{(E, R, C)\}; (E, R, C))$ in P ; i.e. there are no cycles of length 1 in $\mathcal{G}(P)$.

4 COMPLETION AND PROOF DIAGRAMS

The proof will proceed by cases in the following manner: an arbitrary fact (E_i, R_i, C_i) in $F(P)$ is chosen such that $(\{(E_i, R_i, C_i)\}; (E_{i+1}, R_{i+1}, C_{i+1})) \in P$. If 'is an ancestor of' is not irreflexive on $F(P)$, then there is an inference $(\{(E_{i+k}, R_{i+k}, C_{i+k})\}; (E_i, R_i, C_i)) \in P$ ($k > 1$) and so the proof proceeds by contradiction. The six cases which can result from each of the six inference rules of \mathcal{IC} being applied to (E_i, R_i, C_i) are considered.

Case 1 *The next inference is a Deduce*

$C_{i+1} \supset C_i$ and so at some point there must be an inference in which $C_j \supset C_{j+1}$ ($i + 1 \leq j < i + k$), which is a contradiction by Lemma 4.21.

Case 2 *The next inference is a Delete*

$|(E_{i+1}, R_{i+1})| < |(E_i, R_i)|$, so to get a cycle the number of equations and rules must be increased. The only inference rule which can increase the number of equations and rules is Deduce by Lemma 4.20, which occurs in no cycles by case 1. Thus there is a contradiction and this case can not occur.

Case 3 *The next inference is an Orient*

$|R_{i+1}| > |R_i|$, so a rewrite rule must be removed from an R_{i+p} ($1 < p \leq k$) and at some stage, an equation is either deleted or disappears (i.e. it already exists in a set) from a fact. As Deduce is the only rule which can increase the number of equations and rules, and this is precluded by case 1, there is a contradiction.

Case 4 *The next inference is a Collapse*

The number of elements of R has to be increased, but as the inference rule Orient is precluded, (case 3) this can not happen (as Orient is the only inference rule which can increase $|R|$ by Lemma 4.20) and this gives us a contradiction.

Case 5 *The next inference is a Simplify*

There is no Orient and no Deduce by cases 3 and 1. The equation which is being simplified must be replaced in an E_m ($i + 1 < m \leq i + k$), and the only remaining inference which adds an equation to an E_m is Collapse (Deduce does not occur by case 1). This does not occur by case 4 and there is a contradiction.

Case 6 *The next inference is a Compose*

There is no way to replace the rewrite rule which has been rewritten as the Orient rule is precluded (by case 3), and so there is a contradiction.

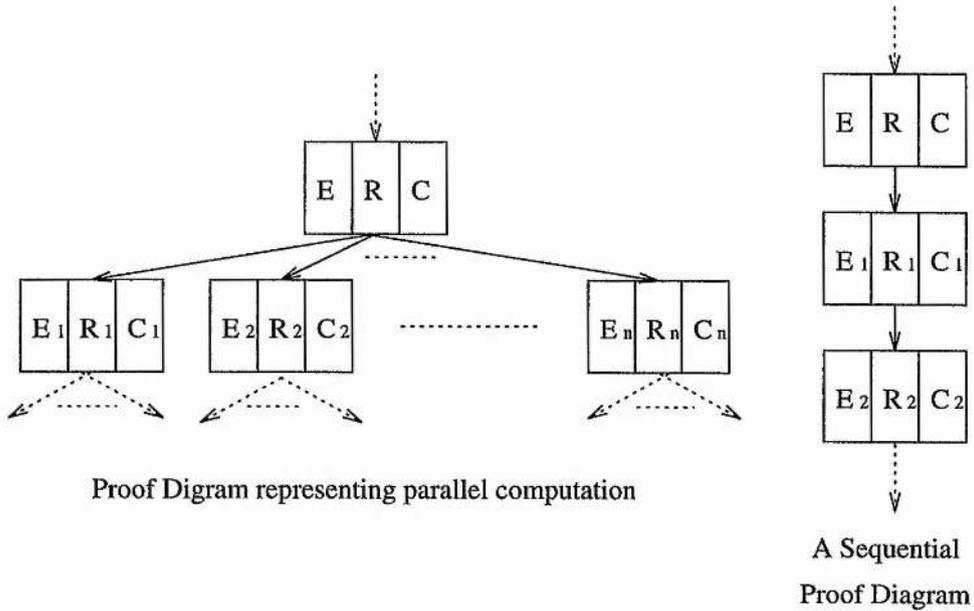


Figure 10: Representation of Sequential and Parallel C-Completion Procedures

Thus 'is an ancestor of' is a transitive, irreflexive relation on $F(P)$, and there are no cycles in P ; thus 'is an ancestor of' is a strict partial order on $F(P)$. □

This result shows that some of the redundancy of the inference rules of Bachmair has been removed; i.e. in Bachmair's formalisation it was possible to get cycles in the associated proof graphs and in the formalisation of c-completion this can not happen. Thus the inference rule formalism here is closely related to how we would like to implement a completion procedure.

4.3.4 Representations of Proof Diagrams

Now a definition will be given which will represent the way that a sequential algorithm is implemented. Figure 10 represents parallel and sequential implementations of a c-completion procedure. There is an implicit local time condition on the relation 'is an ancestor of' on the facts of a proof diagram; this is what gives us the parallel representation. Thus it is possible to define the special case of the sequential proof diagram, which we will prove has the form of a line.

4 COMPLETION AND PROOF DIAGRAMS

Definition 4.23 Let P be a proof diagram generated by $\alpha = (E, \emptyset, C)$ and $>$. P is said to be a *sequential* proof diagram if each fact in $F(P)$ is the parent of at most one other fact in $F(P)$.

The next definition defines a special type of graph which will be shown to correspond to $\mathcal{G}(P)$ where P is sequential.

Definition 4.24 A graph $G = (N, lab, succ, r)$ in which there is:

- a path from the root r to all nodes $n \in N$,
- at most one successor for each node $n \in N$ and
- no cycles

is called a *line*.

Theorem 4.25 Let P be a proof diagram generated by $\alpha = (E, \emptyset, C)$ and $>$ where P is sequential. Then $\mathcal{G}(P)$ is a line.

Proof— From the way $\mathcal{G}(P)$ is defined there is a root r such that there are paths from r to all nodes in $\mathcal{G}(P)$. As the procedure is sequential, each node $p \in F(P)$ is the parent of at most one node. As ‘is an ancestor of’ is a strict partial order on $F(P)$, it is irreflexive on $F(P)$, and hence there are no cycles in $\mathcal{G}(P)$. So $\mathcal{G}(P)$ is a line. \square

Of course, in the general case (i.e. where the procedure is not sequential) the associated proof graph of a proof diagram P will be a DAG.

4.4 Examples and Further Work

Example 4.15 continued. In Figure 9, a portion of the proof diagram has been surrounded by a dotted box, and it will be shown that this is in some sense redundant. The proof diagram represented in Figure 9 is generated from axiom $(\emptyset; (E, \emptyset, \emptyset))$ (where $E = \{ba = ab, aab = b, abb = a\}$) and the length then lexicographic reduction ordering with $b > a$. Using a critical pair criterion CPC , where $CPC(E) = \{abab = bb, abbb = ba\}$, it is

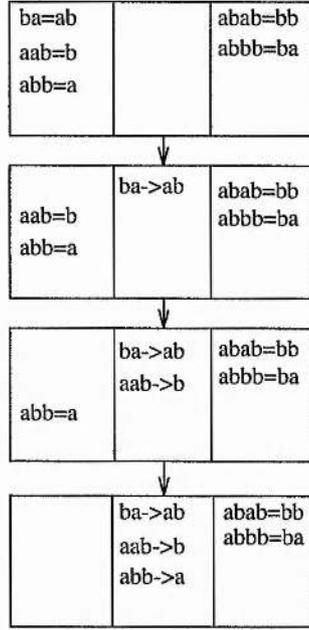


Figure 11: $CPC(E) = \{abab = bb, abbb = ba\}$

possible to construct a proof diagram Q generated by $(\emptyset; (E, \emptyset, CPC(E)))$ and the length then lexicographic reduction ordering. Figure 11 shows a number of inferences which could constitute the ‘start’ of such a proof diagram.

This proof diagram precludes the generation of the critical pairs $abab = bb$ and $abbb = ba$. It can be seen that the proof diagram from Figure 11 can ‘replace’ the first column from Figure 9, and if this is done, a proof Q' of $(**)$ is formed. It is easy to check that the first fact in the second column of Figure 9 (which we will denote by m) really does follow from the last fact from the column in Figure 11 (which we will denote by l).

It is possible to say more than this about the way Q' relates to P in this particular example. Suppose we define $\mathcal{G}(P)$ and $\mathcal{G}(Q')$ in the following way:

- $\mathcal{G}(Q') = (F(Q') \dot{\cup} \{s\}, succ, lab, s)$, where $lab(s) = 0$ and for $f = (E_i, R_i, C_i) \in F(Q'), lab(f) = |(E_i, R_i)|$ and $succ$ is defined as usual
- $\mathcal{G}(P) = (F(P) \dot{\cup} \{r\}, succ_1, lab_1, r)$, where $lab(r) = 0$ and for $g = (E_j, R_j, C_j) \in F(P), lab(g) = |(E_j, R_j)|$ and $succ_1$ is defined as usual

It is possible to define a map $\pi : F(Q') \dot{\cup} \{s\} \rightarrow F(P) \dot{\cup} \{r\}$. We refer by $c_{i,j}$ to the j th

4 COMPLETION AND PROOF DIAGRAMS

fact in the i th column in Figure 9. The map π is defined as follows

- $\pi(s) = r$
- for each b_j in Figure 11 (where b_j is the j th fact occurring in the proof diagram)

$$\pi(b_j) = c_{1,j}$$
- $\pi(c_{k,m}) = c_{k,m}$ for $k > 1$

It is easy to check for this example that the following theorem is true, where P and Q' are as defined in the example above.

Theorem 4.26 *Suppose P and Q' are defined as in Example 4.15. Then $\mathcal{G}(Q') \sqsubseteq_{\mathcal{G}} \mathcal{G}(P)$*

Future work will try to concentrate on how applying an arbitrary critical pair criterion will affect a proof diagram P , and if it is possible to define a way so that the embedding in the example above will always occur.

There is also another way to view the facts in the dotted boxes in Figure 4.15 as redundant. We simply create a derived inference which takes the fact l from Figure 4.15 and returns the fact m in Figure 11; i.e. $\mu = (\{l\}; m)$. Then it would be possible to remove the ‘redundant’ inferences from Figure 4.15 and replace them with μ . This ‘rewriting’ of a proof diagram will be considered further in Chapter 5.

Example 4.27 This example was suggested by U. Martin and shows a number of interesting properties. Again, we will be working with the completion procedure for strings. The given presentation is one for the trivial monoid. The heuristic ‘orient every equation immediately’ will be added to simplify the proof diagram. The proof diagram P is generated from axiom $\beta = (\emptyset; (\{a \rightarrow 1, a^3 \rightarrow 1\}, \emptyset, \emptyset))$ (1 denotes the empty string ϵ) and the length reduction ordering $>$. Figure 12 shows a graphical representation of the proof diagram. We will also see how a rather simple heuristic, which could be implemented in a completion theorem prover such as Otter [93] works on a range of examples.

Figure 12 shows proofs of a number of facts for which $KB(P, a)$ is true, and some different proofs of these facts. It is important, and easy, to see that there is a simple proof of the fact that $\{a \rightarrow 1\}$ is a convergent rewriting system which does not calculate *any* critical pairs.

4 COMPLETION AND PROOF DIAGRAMS

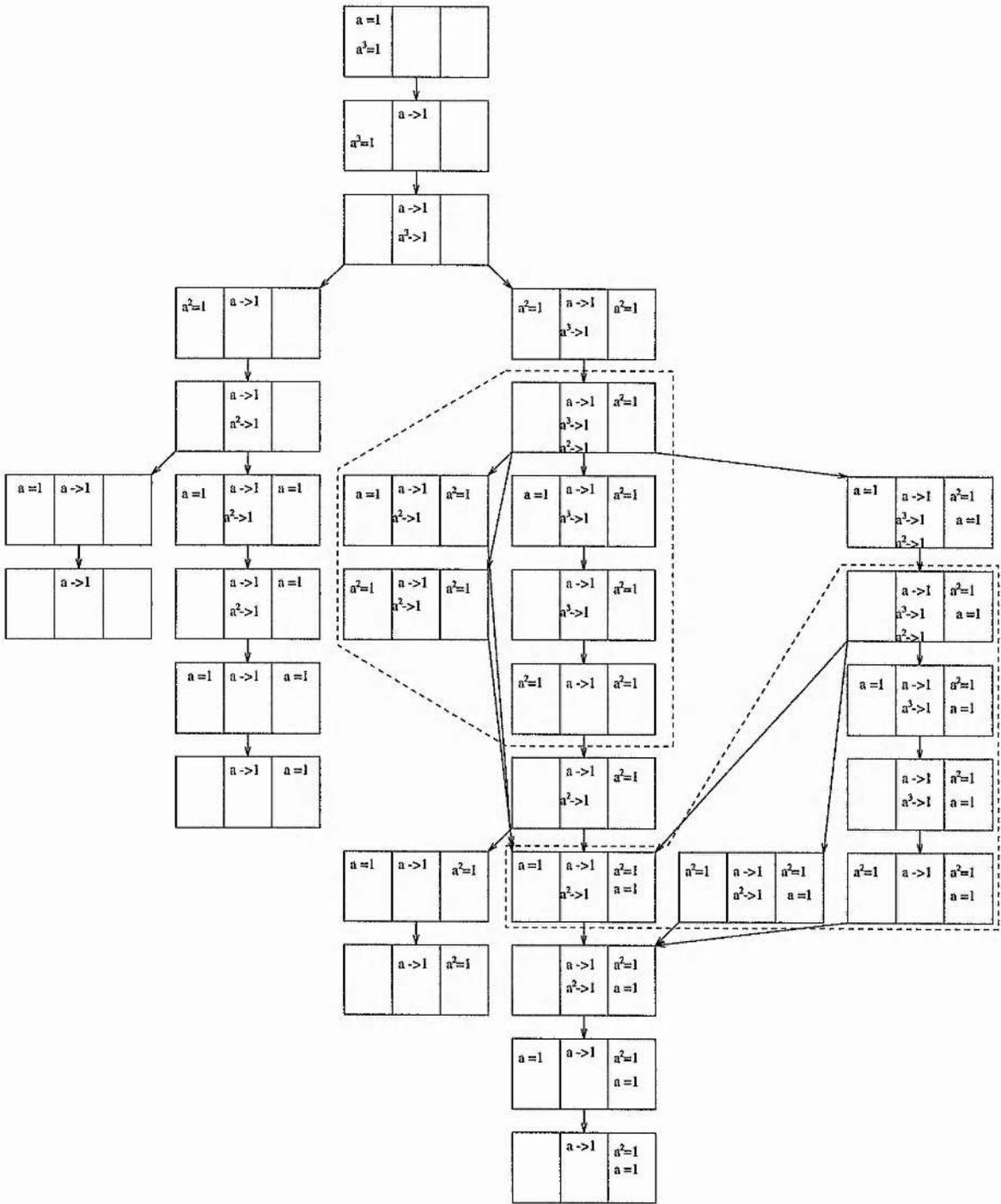


Figure 12: Proof Diagram P generated by β and $>$

4 COMPLETION AND PROOF DIAGRAMS

This is the left-hand-most portion of the proof diagram P . Therefore it can be seen that there is a lot of redundancy in each of the other derivations of the convergent rewriting system $\{a \rightarrow 1\}$ in the proof diagram P .

There is also an amount of repeated, isomorphic substructure to portions of the proof diagram P . Two such substructures have been surrounded by dotted boxes. Each of the facts in the substructures have a different set of redundant critical pairs in them. If it could have been foreseen that these critical pairs were redundant, then these parts of the proof diagram could be 'pruned'. Thus a critical pair criterion could be applied to this problem.

It can also be seen that certain derived rules of inference could have been used to simplify the procedure. It is noticeable that certain equations and rules 'have to occur' in every proof that $\{a \rightarrow 1\}$ is a convergent rewriting system. So, for instance, a useful derived inference may be $(\{a^3 = 1\}, \{a \rightarrow 1\}, C); (\{a^2 = 1\}, \{a \rightarrow 1\}, D)$, as $a^2 = 1$ occurs in every path which includes a terminus in the proof diagram. This 'derived inference' precludes the orientation of $a^3 \rightarrow 1$ (as it can simply be normalised). This notion of the derived inference will be studied in more depth in Chapter 5.

Example 4.28 Now an extension of the previous example will be investigated. Suppose we choose two natural numbers r and s , then given the set of equations $\{a^r = 1, a^s = 1\}$ and the length reduction ordering, then there is a convergent rewriting system $a^{(r,s)} \rightarrow 1$. There are now two cases which may lead to the proof diagram of such a completion procedure growing infinitely.

Firstly, not all equations which are generated have to be oriented. In this case $E_i \neq \emptyset$ for any E_i in a fact (E_i, R_i, C_i) . The heuristic 'orient every orientable equation immediately, if not orientable then delete' can be applied to preclude this eventuality (it should be noted here that the only unorientable equations in this example are the ones of the form $a^j = a^j$, which can be deleted). This heuristic is used in many real implementations of string rewriting systems (see, for example, Holt [60]).

The other problem which may occur is that the rewrite rules may be used to generate more critical pairs, which are then immediately oriented. In theorem provers which have completion capability it is possible to employ a simple heuristic which will stop this problem. There is a system which 'weights' equations so that critical pairs over a certain weight are

not calculated. Obviously, in certain cases this is undesirable, as it may stop a convergent system from being discovered, but for the purposes of this example, it is easy to see that this will be of help.

The weighting system which will be used depends upon the length of the strings: an equation e will be said to have weight $k + l$ if and only if $e \equiv a^k = a^l$. The heuristic could then be stated 'create only critical pairs of weight less than t ' and could be implemented in a system such as Otter. We choose $r + s = t$. If $t = 0$ was chosen, then the critical pairs which would need to be calculated as a Church-Rosser test would not be used, although of course this would still lead to a convergent system.

This approach has the advantage that the DAG created by generating a proof diagram P from the axiom $(\emptyset; (\{a^r = 1, a^s = 1\}, \emptyset, \emptyset))$ and the length reduction ordering is finite. Suppose that there were infinitely many facts b_i which were derived by an inference $(\{b\}; b_i)$ in P . This is not true, as with finite sets E_i and R_i in a fact $(E_i, R_i, C_i) \in F(P)$, there are only finitely many operations possible. In particular, given $\rho_1, \rho_2 \in R_i$, there are only finitely many critical pairs which can be calculated between ρ_1 and ρ_2 . So the path should be infinitely long. If this were true then there would be no fact $(E_i, R_i, C_i) \in P$ such that $E_i = \emptyset$ and $CP(R_i) \subseteq C_i$. Every equation is immediately oriented or deleted, and so there are certainly facts in the path for which $E_i = \emptyset$. So the only possible way for such an infinite path to exist is for there to be a way to create infinitely many new equations which could then be oriented. This can only occur if there are infinitely many critical pairs calculated. But as the heuristic 'generate critical pairs of weight less than $r + s$ ' is used, it is possible to see that there are only going to be finitely many of these critical pairs. This gives the following theorem.

Theorem 4.29 *Let P be a proof diagram generated by $(\emptyset; (\{a^r = 1, a^s = 1\}, \emptyset, \emptyset))$ and the length reduction ordering. Suppose that the heuristics*

- '*orient immediately, if not orientable then delete*' and
- '*calculate only critical pairs of weight less than $r + s$* '

are used, then $\mathcal{G}(P)$ is a finite DAG.

4.5 Conclusions

Thus it can be seen that proof diagrams can be used to observe the behaviour of heuristics in as complex a procedure as Knuth-Bendix completion. In general, a small number of completion programs (for instance ReDuX [18]) implement critical pair criteria. Although a given equation may indeed be 'redundant' in a completion procedure, this equation, or some direct consequence of it, may well help at some intermediate stage, and make the procedure more efficient by doing a necessary simplification at an early stage. Still, work is going on into investigating such heuristics and looking at sets of examples where these criteria do work.

In general more specialised heuristics are needed to complete even moderately sized interesting examples. It seems as if using 'clever' data structures (see, for example, Sims [112]) gives much the same sort of efficiency as using these general heuristics, and it seems intuitive that an understanding of the particular problem and solution space will give rise to more efficient instances of completion procedures. This will be looked at in more depth in Chapter 6.

5 Derived Inferences and Proof Diagrams

5.1 Introduction

In this chapter a model of a derived inference in the setting of proof diagrams will be introduced. Derived inferences in some sense ‘abbreviate’ parts of a proof of a fact. If one wants to prove lengthy theorems in a logical system, then derived inferences speed up the process of proving facts. Using just primitive inferences, relatively simple theorems can take many inference steps to prove: derived inferences mean that one sound step can be taken which represents many steps in the underlying logic.

Derived inferences are also seen in proof development systems in a different guise. Proof Development Systems (PDSs) in the LCF tradition (Gordon *et al* [47]) have been built on the idea of developing proofs in logical systems in a practical manner. The fundamental ideas of these provers were the following:

- user interaction is through a programmable meta-language, ML
- formulae and ways to manipulate the formulae are ML data
- the prover guarantees soundness

In order to make the proof methods effective, it was obvious that complex ‘heuristics’ were needed in order to pare the search space and cut down the workload of the theorem prover. There were three main methods for making theorem proving more practical in these PDSs:

- use of already proved lemmas
- derived rules of inference
- *tactics* and *tacticals*

Proof development systems are used in a variety of fields; for instance constructive mathematics, hardware verification and software verification. These uses demand that the theorem provers are robust and have an automatic inference system which, if guided correctly, will make complex proofs ‘easy’ to construct, and perhaps as importantly, easy to understand.

Gordon and Melham [46] p. 364 informally define tactics in HOL as follows:

5 DERIVED INFERENCES AND PROOF DIAGRAMS

A tactic is an ML function that when applied to a goal G reduces it to (1) a list of subgoals G_1, \dots, G_n along with (ii) a justification function mapping a list of theorems to a theorem.

Refinement tactics in Nuprl are defined similarly. The justification function does exactly what its name implies, it constructs a justification of the inference in which G is inferred from G_1, \dots, G_n through application of the primitive inference rules. Tactics can take a long time to execute, as each step needs to be justified. Tacticals are the 'glue' which makes it possible to build compound tactics. For instance if one wanted to sequence two tactics T_1 and T_2 for further use, one might build a tactic $T_3 = T_1 \text{ THEN } T_2$. Paulson [100] p. 81 informally defines tactics for Isabelle as follows:

[tactics] are essentially functions from theorems to theorem sequences, where the theorems represent states of backward proof.

Nuprl also has *transformation tactics*. Constable *et al* [29] say that such a 'tactic serves as an operation on proofs, *transforming* one proof to another.' It is possible for the use of a transformation tactic to generate a new proof which is justified by a large number of applications of primitive inference rules.

In Paulson [100] p. 14 derived rules are informally defined as 'conservative extensions of the object logic, [which] may permit simpler proofs'. This means that derived inference rules do not strengthen the logic. It is possible to build derived inferences in Isabelle, but Nuprl currently includes no derived inference rules, and the concept of derived rule in HOL includes a 'justification in the primitive inferences'.

Given a formula G , there may be a derived inference in which G can be derived from G_1, \dots, G_n . It may also be possible to apply a tactic to G which returns G_1, \dots, G_n and a justification function. Looking at a static representation of a proof of G , the result of applying a derived rule and a tactic may appear similar. However, the effect of the application of a derived inference or tactic is different: applying a derived inference takes just one step, whereas applying a tactic may mean repeated applications of many primitive inference rules.

The idea of 'proof abbreviations' is that they will allow the theorem prover to do some of the

more ‘tedious’ parts of a proof of a fact, while retaining soundness. This means taking short cuts through a proof of a fact by throwing away some of the applications of the primitive inferences and deriving theorems which are still true in the logic of the theorem prover.

The emphasis of this chapter is to show that derived inferences can easily be modelled with the aid of proof diagrams. Derived inferences are important heuristics for many types of machine assisted theorem provers. We also introduce a notion of rewriting proof diagrams which can clean up complex pieces of proof of a fact. Nuprl’s transformation tactics are the motivation for this. It should, however, be noted that the effect of applying a transformation tactic to a proof is different from the notion of rewriting proof diagrams in this chapter: the application of a transformation tactic to a proof may be an extremely lengthy process whereas we rewrite proof diagrams in one step.

Section 5.2 defines derived inferences in the context of proof diagrams and shows that these inferences really are sound extensions to the system. The relationship between proof diagrams defined in Section 5.2 is briefly investigated in Section 5.3. Section 5.4 investigates how derived inferences can be used in defining rewrite rules for proof diagrams, and a method of rewriting proof diagrams is introduced. This is somewhat like the notion of a transformation tactic from Constable *et al* [29]. Section 5.4.3 looks at how it could be possible to generate useful derived inferences automatically, and gives some promising research directions for the future.

5.2 Derived Inferences

This section will investigate the role of derived inferences in proof diagrams. In particular, it will be shown that derived inferences correspond to a condition on the proof graph associated with a proof diagram. It will also be shown that these inferences correspond to a sound extension of the system.

5.2.1 Definitions for Derived Inferences

The proof diagram is investigated in order to see which nodes are passed through to get from the axioms of to the ‘goal’ node (i.e. the node in the graph which corresponds to the fact which we wish to prove). It is a set of nodes which ‘cover’ all paths from the root of

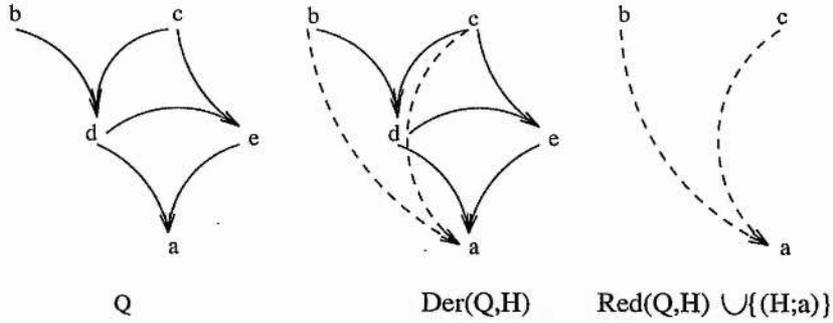


Figure 13: An example of a proof diagram with a derived inference

the proof graph to the goal node which are to be considered in a derived inference.

Definition 5.1 Let Q be a proof of a fact a ($a \notin Ax(Q)$) which is minimal with respect to a proof P of a . The set of facts $H = \{h_1, h_2, \dots, h_n\}$ ($a \notin H$) are said to form a *covering set* for a in Q if every rooted path in $\mathcal{G}(Q)$ to the fact a has at least one of the h_i ($i = 1, \dots, n$) as an element. If no proper subset of H forms a covering set for a in Q , then H is said to form a *minimal covering set* for a in Q .

In Figure 13, one can see covering sets for a in Q . An obvious choice is the set of axioms themselves, $\{b, c\}$. $\{d, e\}$ and $\{c, d\}$ could also be chosen as minimal covering sets for a in Q . Any other choice of a set of nodes is not a minimal covering set for a in Q . Now notation is introduced so that it will be possible to refer to minimal covering sets in certain situations.

Definition 5.2 The proposition $MinCS(Q, P, H, a)$ is true if and only if Q is a proof of a fact a which is minimal with respect to a valid proof P of a and $H = \{h_1, h_2, \dots, h_n\}$ is a minimal covering set for a in Q . It is false otherwise.

The following definitions allow us to characterise various properties and constituent parts of a proof diagram Q when $MinCS(Q, P, H, a)$ is true. This analysis will let us replace parts of the proof diagram in a sound way. The new inference which is used is supposed to represent a derived inference in a logical system. Firstly we define the derived inference itself.

Definition 5.3 Let $MinCS(Q, P, H, a)$ be true. The proof of a with derived inference is defined to be the proof diagram:

$$Der(Q, H) = Q \cup \{(H; a)\}$$

$(H; a)$ is called the *derived inference*.

So in Figure 13, the derived inference $(\{b, c\}; a)$ is used. This is represented by the inference shown with a dashed line, and $Der(Q, H)$ is the resulting proof diagram, where $H = \{b, c\}$.

When $MinCS(Q, P, H, a)$ is true we construct a proof subdiagram of Q . This proof subdiagram will be shown to contain all necessary information so that there will be proofs of each of the $h_i \in H$. We characterise it first by looking at the relation 'is an ancestor of' on the facts of a proof diagram.

Definition 5.4 Let $MinCS(Q, P, H, a)$ be true. The proof diagram

$$Red(Q, H) = \{(A; b) \in Q \mid \exists i, 1 \leq i \leq n, b = h_i \text{ or } b \text{ is an ancestor of } h_i\}$$

is called the *reduced proof diagram of Q* .

The proof diagram $Red(Q, H) \cup \{(\{b, c\}; a)\}$ is shown in Figure 13. This represents the proof diagram

$$Red(Q, H) \cup \{(\{b, c\}; a)\} = \{(\emptyset; b), (\emptyset; c), (\{b, c\}; a)\}$$

Lemma 5.5 Let $MinCS(Q, P, H, a)$ be true. Then $Red(Q, H)$ is a proof of all $h_i \in H$.

Proof— As the proof Q of a is minimal with respect to the proof P of a , Q is a valid proof and hence is a proof of all $h_i \in H$. $Red(Q, H)$ retains all inferences from Q which are used in any derivation of an h_i . Thus if $Red(Q, H)$ were not a proof of an h_i , neither would Q be a proof of this h_i , which is a contradiction. \square

The next definition characterises a set of inferences in Q which are not necessary in a proof diagram when an inference $(H; a)$ is known and $MinCS(Q, P, H, a)$ is true.

Definition 5.6 Let $MinCS(Q, P, H, a)$ be true. The set of *indirect elements of Q with respect to H* is defined to be:

$$Ext(Q, H) = Q - Red(Q, H)$$

Thus $Ext(Q, H)$ contains all those elements of Q which are, in some sense, deduced from the covering set; i.e. the hypotheses in H form a sufficient set to deduce a . Suppose we choose to look at the covering set $H = \{b, c\}$ in the proof diagram Q in Figure 13.

$$Ext(Q, H) = \{(\{b, c\}; d), (\{c, d\}; e), (\{d, e\}; a)\}$$

The following lemma shows that our characterisation of a derived inference is in some sense correct: i.e. by looking at a reduced proof diagram and a derived inference, we still have a proof (which has nice properties) of a fact.

Theorem 5.7 *Let $MinCS(Q, P, H, a)$ be true. Then $R = Red(Q, H) \cup \{(H; a)\}$ is a proof of a which is minimal with respect to the proof $Der(Q, H)$ of a .*

Proof— Firstly it will be shown that $Red(Q, H) \cup \{(H; a)\}$ is a proof of a . This follows immediately from the fact that $Red(Q, H)$ is a proof of all h_i by Lemma 5.5, and a does not occur in $Red(Q, H)$ ($a \neq h_i$ and a is not an ancestor of h). So it is necessary to show that the proof R of a is minimal with respect to the proof $Der(Q, H)$ of a . There are no ‘smaller’ proof diagrams than R in $Der(Q, H)$; if there were, then the proof Q of a would not be minimal with respect to the proof P of a , as an inference could be removed from Q and a proof of a retained. Thus $Small(R, Der(Q, H), a)$ is true and the proof R of a is minimal with respect to the proof $Der(Q, H)$ of a by Lemma 3.38. \square

5.3 Relationships Between Proof Diagrams

This section will discuss the relationship between $Red(Q, H) \cup \{(H; a)\}$ and Q when $MinCS(Q, P, H, a)$ is true. It is possible to see that these two proof diagrams are closely related as there is an amount of structural information which is preserved when constructing $Red(Q, H) \cup \{(H; a)\}$ from Q .

A derived inference replaces a ‘cluttered’ piece of a proof of a fact by something which is easier to read. Cluttered is a word used by the Nuprl community to describe a piece of a proof which is complex. We wish to retain local information; i.e. the relation ‘is an ancestor of’ on the facts of $Red(Q, H) \cup \{(H; a)\}$ should be preserved from the same relation on the facts of Q .

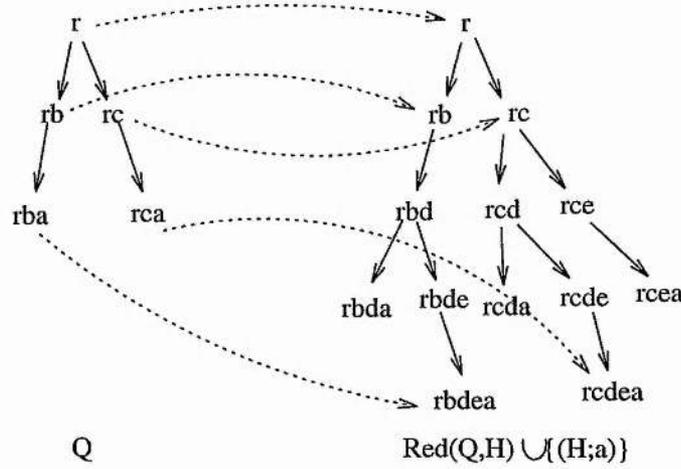


Figure 14: relationship between unravelled proof graphs

Definition 5.8 Let P and Q be proofs of a fact a . P is said to be *strongly related* to Q if:

- $F(P) \subseteq F(Q)$ and
- for each pair of elements $r, s \in F(P)$, if r is an ancestor of s in Q , then r is an ancestor of s in P .

Lemma 5.9 Let $MinCS(Q, P, H, a)$ be true. Then $Red(Q, H) \cup \{(H; a)\}$ is strongly related to Q .

Proof— h_1, \dots, h_n, a are in $F(Q)$ (as H forms a minimal covering set for a in Q) and so $F(Red(Q, H)) \subseteq F(Q)$. By the definition of covering set, h_1, \dots, h_n are ancestors of a in Q , and in $Red(Q, H)$ they are parents of a . Thus the desired ancestry property exists, as all other inferences which occur in $Red(Q, H)$ also occur in Q . □

This is very much like the notion of ‘throwing away’ structure which was mentioned in Chapter 2. So we would like to say something more than this about the relationship between the facts of the two proof diagrams. We shall investigate the proof graphs associated with the proof diagrams $Red(Q, H) \cup \{(H; a)\}$ and Q from Figure 13. For clarity we omit the labels of the nodes. In Figure 14 we see that it is possible to define an injective mapping from the nodes of $\mathcal{G}(Red(Q, H) \cup \{(H; a)\})$ to the nodes of $\mathcal{G}(Q)$ (the dotted arrows show the mapping). From the way that the labelling is defined for an unravelled graph, there is the following obvious theorem.

Theorem 5.10 *Let Q and $\text{Red}(Q, H) \cup \{(H; a)\}$ be as in Figure 13. Then $\mathcal{G}(\text{Red}(Q, H) \cup \{(H; a)\}) \sqsubseteq_{\mathcal{G}} \mathcal{G}(Q)$.*

It seems very likely that this will generalise, but formalising the definition of the mapping turns out to be difficult. We have the following conjecture.

Conjecture 5.11 *Let $\text{MinCS}(Q, P, H, a)$ be true. Then $\mathcal{G}(\text{Red}(Q, H) \cup \{(H; a)\}) \sqsubseteq_{\mathcal{G}} \mathcal{G}(Q)$.*

This conjecture is particularly interesting as it shows that the notion of homeomorphic embedding characterises heuristics in an intuitive way.

5.4 Rewriting Proof Diagrams

In this section a way to rewrite proof diagram is introduced. This is motivated by the idea of a *transformation tactic* in Constable *et al* [29]:

‘transformation tactics take proofs as arguments and return proofs. These tactics . . . [can] produce a proof which is analogous to the given proof and perform various optimisations to the proof such as replacing subproofs by more elegant or concise ones’

The derived inferences introduced in the last section will be used as a sort of ‘rewrite rule’. This rewriting of proofs of facts makes them easier to read (i.e. it removes ‘clutter’ from proofs of facts). It should be noted here that the notion of rewriting a proof of a fact for proof diagrams does not provide the justification function of a tactic.

5.4.1 Rewrite Rules and Proof Diagrams

In this section a rewrite rule for proof diagrams is defined. This will allow the application of derived rules of inference to transform proofs of facts which have been constructed in a given system. This will build on the ideas of Section 5.2, and a theorem will be proved that a ‘rewritten’ proof diagram is still a proof of a given fact; i.e. the notion of rewriting is sound.

There has been a lot of work on proof transformation; see for example Andrews [1], Pfening [103], Lingenfelder [82] and Pierce [104]. This has mainly consisted of transforming proofs from one logical system into another. The main motivation for this work has been to construct 'human readable proofs'. This section takes a more abstract view of transforming proofs, as we deal with no specific logical system.

Definition 5.12 Let $MinCS(Q, P, H, a)$ be true. The pair $\rho = (Ext(Q, H), (H; a))$ is called a *pd-rewrite rule*. If $Ext(Q, H) \neq \{(H; a)\}$ then it is called a *proper pd-rewrite rule*.

Thus the inferences in $Ext(Q, H)$ will be considered to be redundant and will be replaced in a proof diagram by the single inference $(H; a)$: this is what will constitute the 'rewriting'. As H forms a minimal covering set for a in Q , it is possible to derive a from the hypotheses H , and so such a replacement is sound in the underlying logical system. From now on we will consider all pd-rewrite rules to be proper, as these are essentially the rules which will be 'of use' as a heuristic aid for 'simplifying proofs of facts'.

The next definition gives a way to rewrite proofs of facts by simply rewriting sets of inferences. This means that the operations are carried out on proof diagrams. Later in this section we will give an example of how information can be 'lost' in this operation and give a side condition which will make it possible to rewrite a valid proof into a valid proof.

Definition 5.13 Let $MinCS(Q, P, H, a)$ be true. Suppose that $\rho = (Ext(Q, H), (H; a))$ is a pd-rewrite rule and let R be a proof of a fact b with $Ext(Q, H) \subseteq R$. In the proof diagram

$$Rewrite(R, \rho) = (R - Ext(Q, H)) \cup \{(H; a)\}$$

R is said to be *rewritten* by ρ .

Thus if $MinCS(Q, P, H, a)$ is true and $\rho = (Ext(Q, H), (H; a))$, then $Rewrite(Q, \rho) = Red(Q, H) \cup \{(H; a)\}$, and thus $Rewrite(Q, \rho)$ is a valid proof. Of course, one can not rewrite a proof diagram arbitrarily, as information which may be of use in the rest of the proof diagram could be thrown away in $Ext(Q, H)$. The following example shows how this can occur.

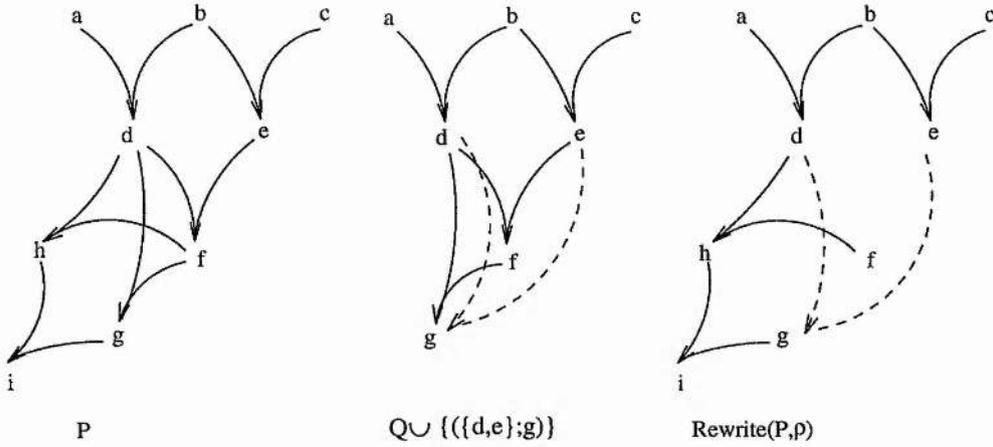


Figure 15: Failed Rewriting

Example 5.14 Figure 15 gives a graphical representation of the proof diagram P . Figure 15 also shows a proof Q of a fact g , which is minimal with respect to the proof P of g . It can be seen that $\{d, e\}$ forms a minimal covering set for g in Q , so it is possible to refer to the derived inference $(\{d, e\}; g)$, which has been added to Q as an inference with dotted lines. Thus $Ext(Q, H) = \{(\{d, e\}; f), (\{d, f\}; g)\}$ and the pd-rewrite rule

$$\rho = (Ext(Q, H), (\{d, e\}; g))$$

can be formed.

If the proof P of i is rewritten with the pd-rewrite rule ρ , then it can be seen in Figure 15 that the resulting proof diagram is not a valid proof; in particular $Rewrite(P, \rho)$ is not a proof of f, h or i . The reason for this is that, although for the purposes of the derived inference the fact f is 'irrelevant' in the proof Q of g , it is not 'irrelevant' in the proof P of i . Thus there is no 'justified' path from f back to the axioms of the proof diagram.

Example 5.14 shows that a side condition must be introduced so that it will be possible to rewrite valid proofs and obtain a valid proof. This side condition is now defined and says that in a proof P of a each of the facts which are being 'thrown away' in the rewriting process can not be the parent of a fact which is used in a different part of the proof of a .

Definition 5.15 Let $MinCS(Q, P, H, a)$ be true. Suppose that $\rho = (Ext(Q, H), (H; a))$ is a pd-rewrite rule and let R be a proof of a fact b such that $Ext(Q, H) \subseteq R$. ρ is

said to be *applicable* to R if no $m \in F(Ext(Q, H)) - F(\{(H; a)\})$ is a parent of any $l \in F(R) - F(Ext(Q, H))$ in R .

Theorem 5.16 *Let $MinCS(Q, P, H, a)$ be true, $\rho = (Ext(Q, H), (H; a))$ be a pd-rewrite rule which is applicable to a valid proof R of a fact b . Then $Rewrite(R, \rho)$ is a valid proof.*

Proof – We need to show that $Rewrite(R, \rho)$ is a proof of all $f \in F(Rewrite(R, \rho))$. As R is a valid proof, we only need to check that there are proofs of all elements $g \in F(R) - F(Ext(Q, H))$ which have an $e \in F(Ext(Q, H)) - F(\{(H; a)\})$ as an ancestor of g in R . There are no such elements, however, as ρ is applicable to R . \square

This theorem gives us a sound way to rewrite proofs of facts: i.e. given a valid proof R and an applicable proper pd-rewrite rule ρ , if R is rewritten by ρ then $Rewrite(R, \rho)$ is a proof of all $f \in F(Rewrite(R, \rho))$.

5.4.2 Properties of Rewriting Proof Diagrams

This section gives a more global perspective on proof diagrams and rewriting. At this point, only one particular pd-rewrite rule is being applied to a proof of a fact. In this section, a method to rewrite a proof of a fact by a set of pd-rewrite rules is given. Thus it is also necessary to define what constitutes a useful set of pd-rewrite rules. This will refer back to the idea of a system from Definition 3.16, as the derived inferences which are being considered must be related to the ‘underlying logic’. In the setting of proof diagrams, the system is the underlying logic. The first two definitions of this section define such a set of pd-rewrite rules.

Definition 5.17 Let $Syst = (\Sigma, \Phi, V, \approx)$ be a system. The *pd-rewrite system* is defined to be the set:

$$\rho(Syst) = \{(A, B) \mid \exists P, Q \subseteq V, H \subseteq \Phi, a \in \Phi \text{ such that } MinCS(Q, P, H, a) \text{ is true, } A = Ext(Q, H) \text{ and } B = (H; a)\}.$$

Definition 5.18 Let $Syst = (\Sigma, \Phi, V, \approx)$ be a system. $Rewr \subseteq \rho(Syst)$ is said to be a *proper pd-rewrite system* if all $r \in Rewr$ are proper pd-rewrite rules.

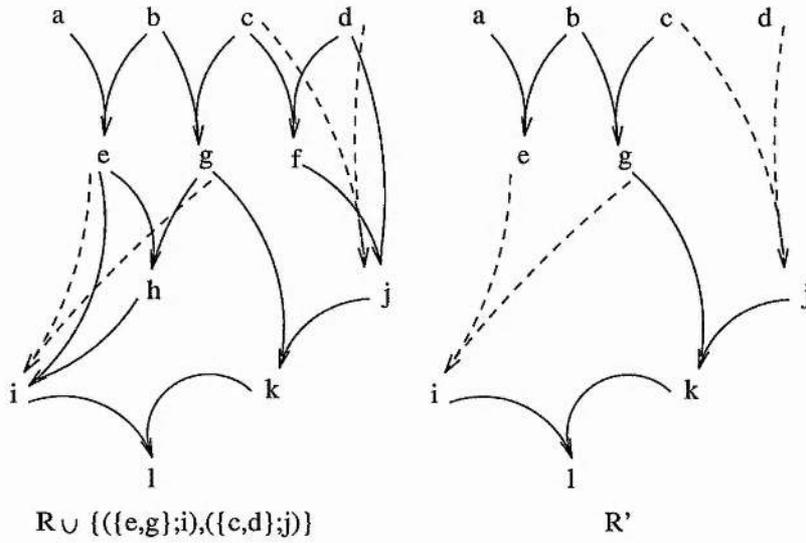


Figure 16: Rewriting with a system of pd-rewrite rules $Rewr$

Now it is possible to define how a proof of a given fact is rewritten. This is a simple extension of the idea used in Theorem 5.16. The proof diagram is rewritten if and only if the applicability conditions of Definition 5.15 are satisfied.

Definition 5.19 Let $Syst = (\Sigma, \Phi, V, \approx)$ be a system, $Rewr \subseteq \rho(Syst)$ be a proper pd-rewrite system with $\rho = (Ext(Q, H), (H; a)) \in Rewr$ and $R \subseteq V$ be a valid proof of a fact $b \in F(R)$. The following inference rule is called the *proof rewriting rule*:

$$\frac{R}{Rewrite(R, \rho)} \quad Ext(Q, H) \subseteq R, \rho \text{ is applicable to } R$$

The following example shows how the process of rewriting a proof diagram proceeds.

Example 5.20 Figure 16 is a representation of a proof diagram R . We have a pd-rewrite system

$$\begin{aligned}
 Rewr = \{ & \{(\{e, g\}; h), (\{e, h\}; i)\}, (\{e, g\}; i), (\{(\{c, d\}; f), (\{d, f\}; j)\}, (\{c, d\}; j)), \\
 & \{(\{d, f\}; j), (\{g, j\}; k)\}, (\{f, d, g\}; j)\}
 \end{aligned}$$

So there are three pd-rewrite rules. We mark with dotted arrows two of the derived inferences. We are able to rewrite R with the two pd-rewrite rules

$$\{(\{e, g\}; h), (\{e, h\}; i)\}, (\{e, g\}; i) \text{ and } (\{(\{c, d\}; f), (\{d, f\}; j)\}, (\{c, d\}; j))$$

shown in Figure 16. The resulting proof diagram, R' , is shown on the right. It is then easy to see that the final pd-rewrite rule

$$(\{\{d, f\}; j\}, \{\{g, j\}; k\}, \{\{f, d, g\}; j\})$$

cannot be applied to R' .

The following theorem shows a useful property for rewriting proof diagrams: the process is terminating when the proof diagram is finite and the proper pd-rewrite system is finite.

Theorem 5.21 *Let $Syst = (\Sigma, \Phi, V, \approx)$ be a system, $R \subseteq V$ be a proof of a fact b , such that $|R|$ is finite, and $Rewr \subseteq \rho(Syst)$ be a proper pd-rewrite system, where $Rewr$ is a finite set. Then there are only finitely many possible applications of the proof rewriting rule to the proof R of b .*

Proof— As each $\rho \in Rewr$ is a proper pd-rewrite rule, $|R| > |Rewrite(R, \rho)|$ and as $|R|$ is finite, only a finite number of applications of such pd-rewrite rules is possible. Thus there are only finitely many possible applications of the proof rewriting rule to the proof R of b .
□

It can also be seen that such a process will not necessarily end in a unique ‘normal form’; i.e. in general the set of pd-rewrite rules are not confluent. This can be seen in Example 5.20. If the pd-rewrite rule $(\{\{d, f\}; j\}, \{\{g, j\}; k\}, \{\{f, d, g\}; j\})$ is used first, then the pd-rewrite rule $(\{\{e, g\}; h\}, \{\{e, h\}; i\}, \{\{e, g\}; i\})$ is applied to R , the remaining pd-rewrite rule will have no effect on the resulting proof diagram and the proof diagram of Figure 17 will occur.

5.4.3 Finding Useful Derived Rules of Inference

This section will give a short exposition on how it is possible to automatically find derived rules of inference with the aid of proof diagrams. The method presented here is obviously very complex, and hence would be too inefficient to implement in a real machine assisted theorem prover. The aim, though, is to show that it is possible to generate automatically such useful heuristics for further use.

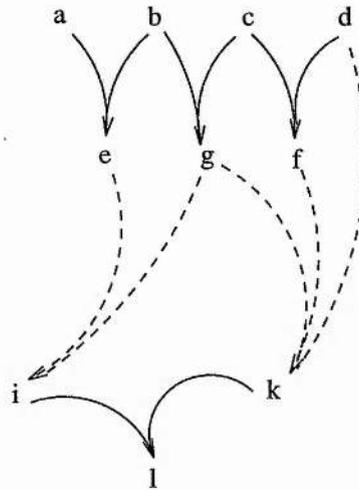


Figure 17: Diagram to show non-confluence of rewriting

The approach has an added attraction: it mirrors the way that humans would derive a tactic or heuristic for a theorem prover. This is generally done by the user seeing patterns that recur in a set of proofs of facts which are being developed. It is by recognising these patterns and recognising that there is a way to automate this part of the proof of a fact that such heuristics are usually generated.

Firstly, it will be necessary to define a set of proofs of different facts in a particular system:

Definition 5.22 Let $Syst = (\Sigma, \Phi, V, \approx)$. Then a *library of proofs in Syst* is defined to be:

$$Lib(Syst) = \{P \subseteq V \mid P \text{ is a valid proof}\}$$

The set $Lib(Syst)$, or any subset of it, represents a set of proofs of facts in the underlying logic which the system is modelling. Most theorem provers have a library of lemmas which are already proved and which may be of use in proving theorems at a later stage.

We need to find a proof Q of a which is minimal with respect to a proof P' of a . The following procedure $prune(P, a)$ finds a proof Q of a which is minimal with respect to a proof P' of a , where the proof P' of a is a subproof of the proof P of a .

Input: finite proof diagram P , fact a , such that $Proof(P, a)$ is true

Output: proof Q of a which is minimal with respect to some proof P' of a , $Proof(Q, a)$ is true

5 DERIVED INFERENCES AND PROOF DIAGRAMS

1. while $Proof(P, a)$ is true do
 - $Q := P$
 - $P := \text{remove}(C; b)$ from P
- end
2. return Q

This procedure returns a proof Q of a which is minimal with respect to itself. We have the following theorem.

Theorem 5.23 *Given a proof diagram P such that $|P|$ is finite and $Proof(P, a)$ is true, the result of the procedure $prune(P, a)$ is a proof of a which is minimal with respect to some proof P' of a .*

Proof— Suppose that the procedure $prune(P, a)$ results in a proof Q of a . Then the result follows from Lemma 3.38 as $Small(Q, Q, a)$ is true. \square

The pruning process thus does not necessarily generate a proof Q of a which is minimal with respect to the original proof P of a , but does at least find a proof of a which is minimal with respect to some proof P' of a . The procedure which will be described next will use a subset of the set of proofs $Lib(Syst)$ to generate derived inferences which may be of use in future attempts to prove facts.

Input: A subset \bar{L} of $Lib(Syst)$

Output: pd-rewrite rule $(Ext(Q, H), (H; a))$

1. Look at P in \bar{L} . Choose a fact a which occurs 'often' in the P s,
2. repeat
 - choose a P such that $a \in F(P)$
 - $prune(P, a)$, call the resulting proof diagram Q
 - find minimal covering sets H for a in Q

until $h_j \in H$ occur 'often' in \bar{L}

3. return ($Ext(Q, H), (H; a)$)

This procedure describes a method for generating new candidate derived inference rules. 1 looks at proofs of facts which occur in \bar{L} and sees which facts occur often in the proofs of facts. If a fact a does occur often, then it may be useful to have heuristics to generate a in the future. In 2 an arbitrary choice of a proof of a is chosen. If the covering set and a occur often in the proofs of facts in \bar{L} , this matches the methods of proving facts which the user is applying. This means that this recurring pattern may be of use in further proofs of facts.

Investigating repeated occurrences of facts in a proof diagram is a rather simplistic way of trying to develop new heuristics. A more complex method might investigate *repeated patterns* in proof diagrams. Boyer and Moore [11] use common patterns of reasoning as a heuristic to guide inductive proofs; these 'patterns' are spotted by hand and coded by hand. Common patterns of reasoning are also explored in the work of Bundy [19] on Proof Plans. These are used to guide the search for proofs in a family of similar proofs, and have successfully been developed in the area of inductive reasoning. 'Common patterns of reasoning [are] identified and represented computationally as proof plans'. 'Capturing common patterns of reasoning' is generally done by hand, however there is work on automating aspects of this; see, for instance, Desimone [39]. An approach whereby one looks for repeated 'patterns' of inferences in proof diagrams would seem to be an interesting idea to explore further. Describing what would constitute a useful 'pattern' is an obvious first step. The 'common patterns' which occur in families of similar proofs of facts are complex (as can be seen in Bundy [19]) and obviously domain specific. The success of the approaches of Boyer and Moore [11] and Bundy [19] make this an interesting area to investigate in future work.

5.5 Conclusions

This section has investigated how proof diagrams and the homeomorphic embedding relation can be used to reason about derived rules of inference in logical systems. In particular, a methodology for rewriting proof diagrams has been introduced. This was motivated the idea of a transformation tactic in a proof development system such as Nuprl.

Section 5.4.3 raises many questions which are of further interest. It would be useful to

5 DERIVED INFERENCES AND PROOF DIAGRAMS

automatically derive new heuristics by investigating commonly occurring substructures in proofs of facts which are already in a library of theorems. A procedure to look for such candidate heuristics is then proposed.

6 Group Theory and Automated Deduction

6.1 Introduction

This chapter is similar to a paper Linton and Shand [84]. The work presented here was developed with guidance from Dr. Steve Linton, particularly in techniques from computational group theory

In this chapter we will look at a number of results in abstract algebra, specifically in the area of finitely-presented groups. The use of completion in abstract algebra goes back to Knuth and Bendix [72], but has focused mainly on special-purpose programs (see Sims [114]) or general systems to complete axiom systems (see, for example, Martin and Lai [89]). In contrast, the objective of this chapter is to look at some results which have been obtained by special purpose programs, but which can be represented as equational reasoning and completion problems, and to investigate how these calculations can be done using various general purpose completion and theorem-proving systems. In general, the existence of finite convergent rewriting systems for finitely presented groups is undecidable. We will concentrate on finite and nilpotent groups, so that the existence of such rewriting systems will not be in question.

We take the view that using general-purpose systems is desirable for at least two reasons. Firstly, fewer programs need to be written, maintained and debugged and secondly, there are a range of examples which are outside the scope of the special-purpose programs, such as groups satisfying additional laws (see Sections 6.7 and 6.8.2).

We compare the performance of various programs on a range of group-theoretical problems. We also investigate different ways of formulating the problems, the effectiveness of various special features of the programs and ways to use additional mathematical information to improve performance. The aim is not so much to set out 'challenge problems' (though some of the problems are quite challenging) as to look at the common features of a range of problems.

Section 6.2 sets out the mathematical background common to all of our problems, and describes the two ways in which the abstract mathematical problems can be converted into concrete problems in first-order equational reasoning. We illustrate this with explicit input

for the LP and Otter provers which can be seen in Appendix A.

Section 6.4 introduces the four systems which we will be using; Otter, a fast resolution theorem prover with added Knuth-Bendix facilities, LP, a flexible and interactive completion theorem prover, KBMAG, a specialised Knuth-Bendix completion program for strings and TC, a very fast coset enumeration program.

In Section 6.5 we look at a number of presentations of finite groups which have been used as test examples in the computational group theory community for many years.

In Section 6.6 we consider presentations of the Fibonacci groups $F(2, n)$ and look in particular at $F(2, 7)$ which has a number of features of interest.

In Section 6.7 we look at the Burnside groups: the free groups in the varieties of groups of fixed exponent. There are natural questions about these groups which cannot be answered directly by the tools of computational group theory, but which lie squarely within the province of equational reasoning.

In Section 6.8.1 we look at some results about infinite nilpotent groups, based on examples from Sims [113], and some open questions about Engel groups.

6.2 Mathematical Background

The material in this section is well-known. We include it here for ease of reference and to establish our notation. References for the group-theoretic material are given. For the universal algebra see Meinke and Tucker [96].

6.2.1 Groups and Presentations

A group is defined as a set equipped with an associative binary operation, a (two-sided) identity and a (two-sided) inverse for every element. In the language of universal algebra we have a signature \mathcal{G} with one sort, one binary operation (written as infix $*$ or implied by juxtaposition), one unary operation (written as postfix $^{-1}$) and one constant e . The class of groups is then the equational class $\text{Alg}(\mathcal{G}, E)$ where E (which is called the theory of the free group) contains the following equations:

$$x * (y * z) = (x * y) * z$$

$$x * e = x$$

$$e * x = x$$

$$x * x^{-1} = e$$

$$x^{-1} * x = e$$

A presentation $\langle C \mid R \rangle$ of a group consists of a set C of constants and a set R of ground elements in the term algebra of $\mathcal{G} \cup C$. An element r of R , called a relator, can be viewed as denoting the equation (or relation) $r = e$. Any equation $t = t'$ can be represented (modulo E) by the relator tt'^{-1} . The group presented by $\langle C \mid R \rangle$ is then the initial algebra with the signature $\mathcal{G} \cup C$ subject to the equations $E \cup R$. For an introduction to group theory, see Lederman [78], for more details see Aschbacher [4], for more information specifically about presentations see Johnson [67].

Determining the structure of groups given by finite presentations has been a major area of computational endeavour for group theorists since the nineteenth century. A 'mechanical procedure' was formalised by Todd and Coxeter [123] even before the advent of electronic computers. Computer implementations of the procedure, called 'coset enumeration', followed as soon as the hardware was available and the algorithm has been progressively refined (see Cannon *et al* [24] and Havas [53]). The algorithm is related to model-building methods used in theorem proving, and constructs a concrete representation for the group from which much information about its structure can be efficiently determined.

The same information can also be determined from a complete rewriting system for the equational theory given by $E \cup R$. If the group is finite then it is easy to see that such a system must exist, and we can construct it by Knuth-Bendix completion (see Sims [114]). This method has the advantage that a complete system may exist even when the group is not finite, whereas the Todd-Coxeter algorithm can never terminate in this case (leaving aside the question of enumerating over a subgroup). Sims [113, 112] has also shown that Knuth-Bendix completion can sometimes be used for other purposes, such as verifying isomorphisms between finitely-presented groups.

6.2.2 Notation

For brevity we introduce some additional operators, defined in terms of $*$ and $^{-1}$. We define:

$$\begin{aligned}x^0 &= e \\x^{n+1} &= x^n * x \text{ for } n > 0 \\x^{-n} &= (x^n)^{-1} \text{ for } n > 1 \\[x, y] &= x^{-1} * y^{-1} * x * y \\[x_1, \dots, x_k] &= [[x_1, \dots, x_{k-1}], x_k] \text{ for } k > 2.\end{aligned}$$

This enables us to define one especially important structural property of groups. A group G is called *nilpotent of class c* if there exists a natural number c such that:

$$\forall x_0, \dots, x_c \in G \quad [x_0, \dots, x_c] = e$$

Any finitely-generated nilpotent group has a finite complete rewriting system with respect to a suitable generating set (see Sims [112]).

6.2.3 The Alternative Formulation of the Problems

Although the description above does formulate a group presentation as a first-order equational theory, it turns out to be rather inefficient in practice. The reason for this is easy to see. With an ordering such that

$$((x * y) * z) > (x * (y * z))$$

the two words

$$\begin{aligned}w_1 &= a_1 * (a_2 * (a_3 * \dots * (a_{n-1} * a_n) \dots)) \\ \text{and } w_2 &= b_1 * (b_2 * (b_3 * \dots * (b_{m-1} * b_m) \dots))\end{aligned}$$

are in normal form, but putting $w_1 * w_2$ into normal form will involve approximately n rewriting steps.

It is possible that this could be avoided by implementing special matching, unification and completion procedures for associative operators along the lines of Peterson and Stickel [102]. The main problem is that there is no unique most general unifier for such theories. We can also avoid this with an alternative formulation of the problems. This formulation has been used *inter alia* in Martin and Lai [89], as well as being the basis of the specialised programs.

First note that under the theory E of the free group, any ground term in $\mathcal{G} \cup C$ (except e) is equivalent to a term of the form

$$t = f_1 * (f_2 * (f_3 * \dots * (f_{k-1} * f_k) \dots))$$

where each f_i is either c or c^{-1} for some $c \in C$. We can form a new signature $\Sigma(C)$ with one sort and $2|C|$ unary operations: c and \bar{c} for each $c \in C$. We can transform any term t in $\mathcal{G} \cup C$ into a term t' in $\Sigma(C)$, by putting it into the above form and then writing

$$t' = g_1(g_2(g_3 \dots g_{k-1}(g_k(x) \dots)))$$

where

$$g_i = \begin{cases} c & \text{if } f_i = c \\ \bar{c} & \text{if } f_i = c^{-1} \end{cases}$$

The term e is replaced by the variable x . We then consider the equational theory $E'(C, R)$ with the equations

$$\begin{aligned} c(\bar{c}(x)) &= x \\ \bar{c}(c(x)) &= x \\ r' &= x \text{ for each } r \in R \end{aligned}$$

It is not hard to see that this formulation is equivalent to the original one, in the sense that there is a bijection between the initial algebras, and the operations of each can be modelled in the other. In the new formulation associativity is built into the term structure, dramatically reducing the amount of rewriting needed. On the other hand, the fact that group elements are no longer one of the sorts of the signature prevents quantification over the group, so that additional laws (for example) cannot be stated.

We will use this alternative formulation for most of our examples, except when we wish to use special features of the programs for which it is not suitable. Appendix A shows input files for the Larch Prover and Otter on a small example.

6.3 Theoretical Results

There is a large body of theory concerning which groups have finite complete rewriting systems, of which we indicate a few high-points here. The approach is typically topological, based on extending the directed graph of words, connected by elementary rewriting steps, to a higher-dimensional cell complex.

Perhaps the most important result (see Anick [3], Groves [49] and Squier [115]) is that all such groups satisfy a cohomological finiteness condition called FP_∞ . Since non- FP_∞ groups with solvable word problem are known, this shows that not all groups with solvable word problem have finite complete rewriting systems. All soluble FP_∞ groups have finite complete rewriting systems, (see Groves and Smith [50] and Kropholler [73]).

In Squier [116] it is shown that when a group has a finite complete rewriting system then any presentation of that group has a property called 'finite derivation type'. Note that it is possible to have a finite complete rewriting system with respect to one set of generators but not with respect to another, (see Jantzen [66] and Kapur and Narendran [69]). This will be discussed in greater depth in Chapter 7.

Most of these results hold for monoids as well as for groups.

6.4 The Programs

We will use four programs in our experiments. Two general-purpose theorem-provers capable of Knuth-Bendix completion, a specialised completion program and a coset enumeration program.

6.4.1 Otter

Otter [93] is a resolution-style theorem-proving program, which also has a built-in version of the Knuth-Bendix completion algorithm. Otter is mainly designed for non-interactive use, but has a primitive interactive mode. Otter allows us to use the lexicographic recursive path ordering and the length then lexicographic ordering in the completion examples. We use Otter version 3.0.

6.4.2 Larch

The Larch Prover (LP) [45] is an interactive theorem prover for multi-sorted first order logic. It also includes an implementation of the Knuth-Bendix completion algorithm. LP has a number of orderings available and can also implement polynomial orderings controlled by user input. The default ordering is called `noeq-dsmpos`, which is the same as the lexicographic recursive path ordering from the left. Although it is not as efficient as other programs, we use LP for these tests because of its greater degree of interaction and extra features which are useful on some of the examples. We use LP version 3.1 beta.2.

6.4.3 String Knuth-Bendix

In the alternative formulation described in Section 6.2.3 the terms which arise are simply strings of unary operators and the Knuth-Bendix completion procedure can be optimised in a number of ways. Simpler data structures can be used, and more efficient indexing and rewriting procedures employed. These are described in Sims [112]. We use a fast implementation (called KBMAG) of such an algorithm due to Holt [40]. No version number is given, but we used the version current in late 1994.

6.4.4 TC

The Todd-Coxeter coset enumeration algorithm (see Todd and Coxeter [123] and Havas [53]) has already been mentioned. We use George Havas' fast and flexible implementation of it, at version 4.06.

The CPU times given were obtained on a SparcStation 10-41 with 64 megabytes of main memory, running SunOS 4.1.3. Times are given in seconds unless indicated otherwise.

6.5 Finite Groups

In this section we examine a number of presentations that have been used as test examples for the Todd-Coxeter algorithm, and report the performance of various Knuth-Bendix completion programs on them.

6.5.1 The Examples

We take our examples from Cannon *et al* [24], which is established among computational group theorists as a landmark study. Tables 3 and 4 of that paper report the performance of various versions of the Todd-Coxeter algorithm on a number of non-pathological and pathological examples. The pathological examples show a large expansion of the intermediate data structures before collapse to a much smaller final result. In Havas [53] the pathological examples are repeated using more recent versions of the algorithm.

Some of the examples involve enumeration of the cosets of a non-trivial subgroup, which corresponds to a special form of ground completion not supported by any of the programs we are using, so we omit those examples. We are left with nine non-pathological examples and nine pathological ones.

The non-pathological cases are:

$$\begin{aligned}
G_1 &= \langle a, b, c \mid a^3 = b^7 = c^9 = (ab)^2 = (bc)^2 = (ca)^2 = (abc)^2 = 1 \rangle \\
G_2 &= \langle a, b \mid a^7 = b^7 = (ab)^2 = (a^{-1}b)^3 = 1 \rangle \\
G_3 &= \langle a, b \mid a^6 = b^6 = (ab)^2 = (a^2b^2)^2 = (a^3b^3)^5 = 1 \rangle \\
G_4 &= \langle a, b \mid a^{30} = b^{30} = (ab)^3 = (a^{-1}b)^{10} = a^3b^3 = 1 \rangle \\
G_5 &= \langle a, b \mid a^4 = b^4 = (ab)^4 = (a^{-1}b)^4 = (a^2b)^4 = \\
&\quad (ab^2)^4 = (a^2b^2)^4 = (a^{-1}bab)^4 = (ab^{-1}ab)^4 = 1 \rangle \\
G_6 &= \langle a, b \mid a^7 = b^2 = (ab)^6 = [a, b]^3 = [a^2, b]^2 = [a^3, b]^2 = 1 \rangle \\
G_7 &= \langle a, b \mid a^4 = b^6 = (ab)^2 = (a^{-1}b)^{12} = [a^{-1}, b]^3 = 1 \rangle \\
G_8 &= \langle a, b \mid a^2 = b^3 = (ab)^{11} = [a, b]^4 = 1 \rangle \\
G_9 &= \langle a_1, \dots, a_6 \mid a_i^2 = 1 \text{ for all } i, (a_i a_j)^2 = 1 \text{ when } |i - j| > 1, \\
&\quad (a_i a_{i+1})^3 = 1 \text{ for } 1 \leq i \leq 4, (a_5 a_6)^4 = 1 \rangle
\end{aligned}$$

G_5 is the Burnside group $B(2, 4)$ (see Section 6.7). G_9 is the Weyl group of type B_6 (see Humphreys [61]).

The pathological cases are:

$$\begin{aligned}
P_1 &= \langle r, s, t \mid t^{-1}rtr^{-2} = r^{-1}sr s^{-2} = s^{-1}tst^{-2} = 1 \rangle \\
P_2 &= \langle a, b \mid a^2 = b^5 = (ab)^7 = [a, b]^2 = 1 \rangle \\
P_3 &= \langle a, b \mid a^{11} = b^2 = (ab)^3 = (a^4ba^{-5}b)^2 = 1 \rangle \\
P_4 &= \langle a, b \mid a^2 = b^3 = (ab)^7 = [a, b]^7 = 1 \rangle \\
P_5 &= \langle r, s \mid r^2sr sr^{-3}s^{-1} = s^2r sr s^{-3}r^{-1} = 1 \rangle \\
P_6 &= \langle a, b, c \mid a^3 = b^7 = c^{16} = (ab)^2 = (bc)^2 = (ca)^2 = (abc)^2 = 1 \rangle \\
P_7 &= \langle a, b \mid b^{-1}a^{-1}bab^{-1}aba^{-2} = a^{-1}b^{-1}aba^{-1}bab^{-4} = 1 \rangle \\
P_8 &= \langle a, b \mid b^{-1}a^{-1}bab^{-1}aba^{-2} = a^{-1}b^{-1}aba^{-1}bab^{-6} = 1 \rangle \\
P_9 &= \langle a, b \mid b^{-1}a^{-1}bab^{-1}aba^{-3} = a^{-1}b^{-1}aba^{-1}bab^{-3} = 1 \rangle
\end{aligned}$$

The presentation P_1 (also called E_1) is due to Higman [59]. It is the first in an infinite series of presentations due to Neumann, which all present the trivial group. The next member of the series is defined by letting $R = s^{-1}tst^{-2}$, $S = t^{-1}rtr^{-2}$ and $T = r^{-1}sr s^{-2}$ and then

$$E_2 = \langle r, s, t \mid T^{-1}RTR^{-2} = R^{-1}SR S^{-2} = S^{-1}TST^{-2} = 1 \rangle$$

where the relators are expanded to become words of length 25 in r , s and t . It is easy to show theoretically or by Knuth-Bendix completion that this presentation presents the trivial group, but this has never been demonstrated by coset enumeration.

For each of these groups, Table 1 below gives the results of using the Havas Todd-Coxeter program. Table 2 gives the results of the specialised Knuth-Bendix program, and Table 3 gives the results obtained with Otter, using the alternative formulation. In each case the length then lexicographic and lexicographic recursive path orderings were used.

A number of observations can be drawn from these figures.

Firstly, the low-level simplicity and high level of development of coset enumeration makes it faster in almost all cases, even when, as in example P_6 , it defines numerous redundant cosets while the Knuth-Bendix techniques go directly to the complete system. Nevertheless, such examples (and the case of E_2) suggest that the Knuth-Bendix procedure may sometimes show more 'insight' into the structure of the group than coset enumeration does.

Secondly, the recursive path ordering nearly always uses fewer rules in both the maximal

Table 1: Results with Coset Enumeration

Group	Order	Max. Cosets	Total Cosets	Time
G_1	504	504	506	0.02
G_2	1092	1092	1092	0.02
G_3	3000	3000	3003	0.18
G_4	3000	3000	3016	0.15
G_5	4096	4096	4341	0.30
G_6	5040	5040	5040	0.36
G_7	5184	5184	5184	0.20
G_8	6072	6072	6072	0.23
G_9	46080	46080	46080	3.0
P_1	1	190	201	0.02
P_2	1	230	230	< 0.01
P_3	660	660	726	0.03
P_4	1092	1248	1332	0.07
P_5	120	652	660	0.02
P_6	21504	52004	52179	2.5
P_7	3	3269	3280	0.12
P_8	5	6474	6478	0.27
P_9	16	26136	26218	0.97
E_2	1	—	—	—

Table 2: Results with a Specialised Knuth-Bendix Completion Program

Group		Length-Lex			RPO		
Name	Order	Max. Rules	Final Rules	Time	Max. Rules	Final Rules	Time
G_1	504	325	325	1.3	57	57	2.1
G_2	1092	467	467	3.0	76	76	1.5
G_3	3000	697	697	17.6	158	158	9.5
G_4	3000	520	520	16.7	16	16	2.3
G_5	4096	822	822	9.7	159	159	3.7
G_6	5040	995	995	16.4	338	338	21.0
G_7	5184	437	437	23.1	160	160	25.4
G_8	6072	688	688	86.1	255	255	22
G_9	46080	39	37	0.1	47	47	0.1
P_1	1	129	6	0.1	57	6	0.06
P_2	1	58	4	0.2	53	4	0.05
P_3	660	236	236	1.6	87	82	1.2
P_4	1092	148	148	6.5	54	54	5.4
P_5	120	284	141	0.3	154	11	1.3
P_6	21504	4666	4666	481	781	690	159
P_7	3	863	6	1.4	390	4	10
P_8	5	1381	6	3.1	2128	4	267
P_9	16	1248	17	3.1	—	—	—
E_2	1	1700	6	12.3	331	6	3

Table 3: Results with the Otter Theorem-proving Program

Group		Length-Lex				RPO			
Name	Order	Rules kept	Given	Final	Time	Rules kept	Given	Final	Time
G_1	504	571	368	347	60.7	578	117	56	19.6
G_2	1092	881	484	467	328	307	112	77	39.0
G_3	3000	2224	744	706	2212	732	204	159	483
G_4	3000	1642	574	520	1045	110	56	17	16.5
G_5	4096	5508	954	822	1462	2214	277	160	322
G_6	5040	2060	1025	985	5696	1163	373	339	2564
G_7	5184	1661	571	540	1843	967	223	161	662
G_8	6072	1633	606	571	6328	980	303	256	2411
G_9	46080	134	90	47	2.0	134	90	48	1.24
P_1	1	681	92	6	4.4	399	62	7	7.1
P_2	1	22	13	4	0.1	22	13	5	0.12
P_3	660	771	236	194	122	407	114	83	75.8
P_4	1092	475	133	100	106	380	93	55	62.2
P_5	120	1538	198	141	24	2327	149	12	86.0
P_6	21504	—	—	—	—	10489	847	691	14920
P_7	3	2358	120	6	37.7	—	—	—	—
P_8	5	7567	224	6	211	—	—	—	—
P_9	16	—	—	—	—	—	—	—	—
E_2	1	7864	341	6	539	9236	295	6	2539

and the final system than the length then lexicographic ordering, and is usually faster, but not as much faster as the smaller systems would suggest. This is because the irreducible strings are usually longer. It does especially well in cases where one or more generators (or their inverses) at the top of the precedence can be quickly eliminated. This will be discussed in more detail in Section 6.6. The notable exceptions are the related presentations P_7 , P_8 and P_9 . In these cases the recursive path ordering gives rise to increasingly long rules.

More detailed examination of the output of Otter suggests that most of its time is spent in forwards and (to a much lesser extent) back demodulation, that is, in rewriting. The comparison with the specialised program suggests that there is a difference in the asymptotic complexity of the rewriting (as the size of the rule system increases), as well as the constant difference that would be expected from Otter's greater generality.

The case of G_9 is remarkable in that a rather large group is represented by a very small complete rewriting system. This is explained by the fact that G_9 is a Coxeter group presented with respect to its canonical generating system of simple roots. There is a large theory of such groups, too complex to describe here (see Humphreys [61] for more details), but important results describe the conditions when a word w , multiplied by a generator s is equal to word w' of length less than or equal to that of w . These results come close to specifying a (small) complete rewriting system directly, which is close to the systems obtained by our programs.

6.6 Fibonacci groups

These give an interesting test case for completion theorem provers. We define the Fibonacci group

$$F(2, n) = \langle a_1, a_2, \dots, a_n \mid a_1 a_2 = a_3, a_2 a_3 = a_4, \dots, a_{n-1} a_n = a_1, a_n a_1 = a_2 \rangle.$$

For some small values of n (3, 4, 5 and 7) a complete system exists from which it can be deduced that the group is finite, and we can obtain this with completion and an appropriate ordering.

These groups were introduced by Conway [30] who posed the following question: can it be proved that $F(2, 5)$ is isomorphic to the cyclic group of order 11? A number of suggestions were made as to how the problem could be solved (see Conway [31]), and many

of the answers involved hand calculations essentially equivalent to rewriting and critical pair formation.

The given presentation of the Fibonacci group $F(2, n)$ is quite circular, so it is not immediately obvious that we will be able to cut down the search space and make large examples viable. If we want to prove that a group is cyclic (as is the case in $F(2, 5)$), then we must eliminate $n - 1$ of the generators. We can therefore get the ordering to do some of the work for us. An ordering such as the lexicographic recursive path ordering which totally orders the generators and then orders terms initially by the largest generator which they contain will encourage elimination of generators; we just use a precedence such as $a_1 > a_2 > \dots > a_n$. The Knuth Bendix ordering (see Knuth and Bendix [72]) can be weighted in such a way as to do the same job of generator elimination (i.e. we weight it so that we get a similar precedence to the one above).

We can use a group presentation simplification program (which will be discussed in detail in Chapter 7) to transform the given presentation into one with two generators and two relations. In the general purpose theorem provers which we used, the completion procedure generated many more than the two relations which we know to be sufficient for a presentation on two generators: therefore, more matching and rewriting was done in the completion example with the given presentation than with the transformed presentation.

As we will see, $F(2, 7)$ is really too large to be done by hand. $F(2, 7)$ is isomorphic to the cyclic group of order 29; but proving that $F(2, 7)$ is cyclic is considered relatively difficult by group theorists using their regular 'tools' of the nilpotent quotient algorithm and the Todd-Coxeter procedure.

In a communication from Bündgen [16], it was stated that the above example took under three minutes on the ReDuX system (see Bündgen [18]). $F(2, 7)$ was completed with the Knuth Bendix ordering with the weighting total in the desired way on the operators. Also, a critical pair criterion (called the transformation criterion in Bündgen [17]) was used. As was stated in Chapter 5, little is understood why these criteria work on some problems better than others. Progress in this area would be especially valuable if it allowed one to exploit information about the structure of the system.

LP could not complete the presentation $F(2, 7)$ even in the transformed two generator, two

relation form. Large rewrite rules and large numbers of rewrite rules made this example particularly difficult. Otter only completed the transformed presentation. The specialised programs were extremely quick. In this example we see why the special purpose programs are used by group theorists; they are far quicker than the general purpose provers which are doing a lot of extraneous work. It should be noted, however, that the Havas Todd-Coxeter program is also doing a lot of extra work on this example: it calculates 72,395 cosets when the minimum number it could use is 29. The short run time shows how efficient the coding of the program really is, and does not reflect badly on the theorem provers against which we are testing it.

There is a great deal of literature on the Fibonacci groups and generalisations. Some further examples of this type can be found in Campbell *et al* [21, 22], Johnson *et al* [68] and Thomas [119]. They look, for instance, at longer strings on the left hand sides, i.e. $F(m, n)$ where the relations are of the form $a_i a_{i+1} \cdots a_{i+m} = a_{i+m+1}$ (where the indices are reduced mod(n)). Further generalisations allow relations with a parameter giving a length to the right hand sides of the equations as well.

$F(2, 3)$, $F(2, 5)$ and $F(2, 7)$ are all non-trivial finite groups, but $F(2, 6)$ and $F(2, n)$, $n > 7$ are infinite. Of course, Knuth-Bendix completion can complete certain presentations of infinite groups. However, as yet we have been unable to complete the given presentation of $F(2, 6)$, if indeed there is a finite, complete system.

6.7 Burnside Groups

One of the areas where we feel machine assisted theorem-proving programs should be applicable to group theory is the study of the varieties of groups which satisfy additional laws. Examples include the variety of abelian groups, satisfying $x * y = y * x$ and the variety of nilpotent groups of given class c , satisfying $[x_0, \dots, x_c] = e$. Attempting to study these varieties by simply completing the axioms usually seems to fail, giving rise to unorderable equations. Unfailing completion (see Bachmair *et al* [7]) does not fail, but does not terminate either.

Completeness is actually a stronger condition than is really necessary. Ground completeness would suffice, but is undecidable in general. Future work could investigate the ideas of

Martin and Nipkow [90] which can sometimes demonstrate ground completeness. In this section we consider some simple results which can be proved in the case of the variety of groups of given exponent.

The Burnside group $B(m, n)$ is the free m -generator group in the variety of groups satisfying the additional law $x^n = e$. These groups have been extensively studied both computationally and theoretically, but a number of important questions remain open (see Vaughan-Lee [124]).

Attempting to complete $E \cup \{x^n = e\}$, we quickly encounter unorderable laws such as $x * y = y * x$ (when $n = 2$) or $x * y * x * y = y * y * x * x$ (when $n = 3$). Unfailing Knuth-Bendix does not appear to terminate. However in the special case of $n = 2$, unfailing Knuth-Bendix can be used (in Otter for example) to prove easily that

$$E \cup \{x * x = e\} \Rightarrow x * y = y * x.$$

Given that fact, which makes $*$ an AC-operator, AC-completion of $E \cup \{x * x = e\}$ is easy.

In the case when n is a prime power (the most theoretically interesting case), the finite quotients of $B(m, n)$ are all nilpotent. These quotients can be explored using a modified version of the nilpotent quotient algorithm (see Havas and Newman [55]), which can itself be viewed as a specialised Knuth-Bendix completion procedure (as was noted in Sims [112]). Investigation of finite quotients (and some theory) reveals that the groups $B(m, 3)$ are all finite and nilpotent of class no greater than 3. Proving this amounts to showing that

$$E \cup \{x * x * x = e\} \Rightarrow [x, y, z, w] = e.$$

As $[x, y, z, w]$ expands to a word of 22 letters, this problem is very hard for most theorem provers. However, in the McCune [2] shows that this problem can be solved by the Otter theorem prover. The input for McCune's solution appears in Appendix A. No explanation is given to explain as to how it was decided to set the flags in the particular way McCune has set them. The two generator group $B(2, 3)$ is actually nilpotent of class 2, as may be proved by showing that, with constants a and b ,

$$E \cup \{x * x * x = e\} \Rightarrow [a, b, b] = e \wedge [a, b, a] = e.$$

These results appear to represent the limit of what can be achieved automatically using the exponent law as a law, however, it is easy to prove that if a Burnside group $B(m, n)$ is finite

then there exist words w_1, \dots, w_k such that

$$B(m, n) = \langle a_1, \dots, a_m \mid w_1^n, \dots, w_k^n \rangle$$

so that some finite collection of n th powers is sufficient. Experimentation with (say) a Todd-Coxeter program (and a little theory) quickly shows that

$$\begin{aligned} B(m, 2) &= \langle a_1, \dots, a_m \mid a_i^2 = (a_i a_j)^2 = 1 \rangle \\ B(2, 3) &= \langle a, b \mid a^3 = b^3 = (ab)^3 = (a^2 b)^3 = 1 \rangle \\ B(3, 3) &= \langle a, b, c \mid a^3 = b^3 = c^3 = (ab)^3 = (bc)^3 = (ca)^3 = (a^2 b)^3 = \\ &\quad (b^2 c)^3 = (c^2 a)^3 = (abc)^3 = (a^2 bc)^3 = (ab^2 c)^3 = (abc^2)^3 = 1 \rangle \\ B(2, 4) &= \langle a, b \mid a^4 = b^4 = (ab)^4 = (a^{-1} b)^4 = (a^2 b)^4 = (ab^2)^4 = (a^2 b^2)^4 = \\ &\quad (a^{-1} bab)^4 = (ab^{-1} ab)^4 = 1 \rangle \end{aligned}$$

The last case, $B(2, 4)$, is example G_5 in Section 6.5.

Gathering these problems together, we have three theorem proving problems which cannot be handled by specialist group theory programs:

$$\text{T1: } E \cup \{x * x = e\} \Rightarrow x * y = y * x$$

$$\text{T2: } E \cup \{x * x * x = e\} \Rightarrow [a, b, b] = e \wedge [a, b, a] = e$$

$$\text{T3: } E \cup \{x * x * x = e\} \Rightarrow [x, y, z, w] = e$$

We also have the presentations above for $B(2, 3)$ (order 27) and $B(3, 3)$ (order 2187). We attempted these five problems with whichever of LP, Otter, the specialised Knuth-Bendix program KBMAG (with its recursive path ordering) and Todd-Coxeter were appropriate. Otter was used in autonomous mode for problems T1, T2, and T3; LP was set to prove the results by completion (problem T1 had to be slightly rephrased (to prove $[x, y] = e$ instead of $xy = yx$)).

The very long time taken by LP on the last problem is partly due to its lack of a fast many-one matcher, and partly due to the high cost of ordering the very deeply nested terms which arise.

Table 4: Run-times for the Burnside Group Problems

Problem	LP	Otter	KBMAG	T-C
T1	4.24	0.06	—	—
T2	73	3.56	—	—
T3	—	—	17742	—
B(2,3)	4.7	0.28	0.02	0.01
B(3,3)	38397	88	1.37	0.15

6.8 Nilpotent Groups

6.8.1 Sims' Example

In Sims [113], an algorithm is introduced which attempts to prove the nilpotency of groups. This is an enhancement of the nilpotent-quotient algorithm (see Sims [112]) which gives the largest nilpotent quotient of a group G , and does not say whether this equals the group G itself.

The nilpotent quotient algorithm gives a generating set for the nilpotent quotient, and a finite complete rewriting system with respect to that generating set. Sims' approach is to use completion, with a carefully chosen ordering (of which more below), to try and derive the relations (rewrite rules) of the quotient from the original presentation of the group, thereby showing the two to be equal.

Sims uses a special-purpose Knuth-Bendix algorithm for groups to try certain examples in Sims [113], and claims to have some fast results for the groups which he has tried. These groups provide a good test for completion theorem provers, and bring up some interesting points about what we can do with such provers.

The main examples used pose the following problem: is the quotient of a free group on n generators by the normal subgroup generated by a certain set of basic commutators the appropriate free nilpotent group? In Sims [113] he considers two examples, where $n = 2$ or $n = 3$ for commutators up to weight 4. The (relatively easy) completion problem arising in the $n = 2$ case is given in Otter format as:

```

set(knuth_bendix).
set(print_lists_at_end).

lex([e(x), e1(x), c(x), c1(x), b(x), b1(x), a(x), a1(x))]).

set(free_all_mem).

list(sos).
(x = x).                (a1(a(x)) = x).
(b1(b(x)) = x).        (c1(c(x)) = x).
(e1(e(x)) = x).        (b(a(x)) = a(b(c(x))))).
(c(a(x)) = a(c(x))).  (e(b(x)) = b(e(x))).
(c(b(x)) = b(c(e(x)))) (a(a1(x)) = x).
(b(b1(x)) = x).        (c(c1(x)) = x).
(e(e1(x)) = x).
end_of_list.

```

The larger example is given in Sims [113].

This family of examples is interesting as it is dealing with infinite groups. We are given a finite presentation of a group and the algorithm looks at the structure of a complete system for this presentation under a given ordering. Not only can we prove that the group is nilpotent from the output, but we can also prove that the group is indeed infinite.

An interesting aspect of this example is Sims' 'less collected than' ordering. This uses a total ordering on the precedence of the generators and thus forces the generators down the lower central series of the group. This may be intuitive for group theorists, but it is also more usually known in theoretical computer science circles as the recursive path ordering. Here is a direct mathematical application for an ordering which is widely seen as one of the most important in theoretical computer science.

This example also reflects a need for more flexible heuristics in completion theorem provers. On orienting the equations we find that the size of the left hand sides of the equations are all the same (for the $n = 3$ example, this means that there are 49 equations all with the same length left hand sides). In Otter, we can weight the equations differently, and the formation of critical pairs will be done with respect to this weighting strategy; so we have total control

over the procedure. LP, however, has a fixed heuristic that looks at the size of the left hand sides. In the statistics below, we have three cases. The first is where we have tried to complete the given presentation (in the table below, this is L1 (and 2)). The second is where we have tried to 'fool' the heuristic by 'stacking' the generators on one side (corresponding to heuristic 6 in the table), i.e. if $g = g'$ we use the equation $g(g')^{-1} = e$. The third case is where the heuristic has been reimplemented by the program's author (corresponding to heuristic 5 in the table below) to look at the size of left and right hand sides.

These examples can generate a large number of equations and involve a very large search space; it is often a long time before any useful information emerges to cut down the size of the problem. There are also no obvious subsets of the equations which we can complete, so the search space may be too large for the provers to handle. There are various ways to cut down the work of the prover. For example, if it could be automated, one could use the following group theoretic theorem: a group G with centre Z is nilpotent if and only if G/Z is nilpotent. Automated proofs, such as Sims' example, will become much more useful to group theorists if we can actually automate standard results of group theory such as the one above; more powerful methods for proof and more intuitive proofs may result.

We were able to invoke the above theorem in LP using deduction rules. These rules allow one to deduce new equations from equations and rewrite rules that already exist. They can be used to cut down the work that the theorem prover needs to do. For $n = 3$ this was necessary to actually prove that the group was indeed nilpotent, as the indexing strategy of LP was not able to deal effectively with the vast numbers of rules generated. The deduction rule which was used was:

when $a*x=x*a, b*x=x*b, c*x=x*c$ yield $x=id$

We can obviously only invoke this theorem when using the formulation of group theory in which the generators of the group are constants, as we cannot assert such theorems in a higher order logic in our first order theorem prover. In the statistics below we look at the case when we use the above theorem and at the case when we use unary operators.

This sort of automation of a theorem is obviously a good property to have in a general purpose theorem prover. There are other such theorems that can be implemented in this way.

Table 5: Statistics for $n = 3$ Sims example

Method Used	Ordering	Rewrites	cpu Time
L,1 (and 2)	-	-	-
L,2,3	-	-	-
L,1,4	-	-	-
L,1,3,4	1848	6080095	11:14:56
L,1,4,6	1665	3932093	4:23:16
L,2,6	-	-	-
L,1,3,4,5	1611	2698396	4:08:37
O,2	1187	120597	2:10
KBMAG	NA	NA	0.9

Code	Description
L	LP
O	Otter
1	Completion with group axioms
2	Completion of unary operators,
3	extra information added
4	'Factoring' theorem implemented
5	Change cp-ing strategy for LP
6	'stacking' operators on one side

We see from the table the huge number of rewrites that LP is doing in the case of $n = 3$. This is mainly because of the indexing strategy. LP uses a list to index its equations and as a vast number of equations are formed, this strategy does not help. We see that Otter is doing less than 5% of the rewriting of LP. The specialised Knuth-Bendix of KBMAG is even quicker: again, this shows the unsurprising advantage of the indexing strategy which KBMAG uses, and we can complete the presentation as given.

6.8.2 Engel Groups

These groups are another example of groups with laws and there are still some outstanding questions which may be able to be resolved with completion. The Engel group $E(2, 3)$ is the free 2-generator group in the variety of groups satisfying the additional law $[x, y, y, y] = e$.

This is not a finite presentation; however, it can be shown that only finitely many instances of the Engel law ($[x, y, y, y] = e$) are needed to present the group, and experiments (Heineken [56]) reveal a set of five instances (below) which suffice. We use the methods of Sims [113] to prove both that these instances suffice and that $E(2, 3)$ is actually nilpotent.

Starting with this set of instances, the nilpotent quotient algorithm gives a presentation for the largest nilpotent quotient in terms of the original generators a and b and four more: $c = [b, a]$; $d = [c, a]$; $e = [c, b]$ and $f = [d, b]$. It is easy to check that this nilpotent group satisfies the Engel law. It remains to check whether the relations of the nilpotent group are implied by the five instances of the law. Since the relations of the nilpotent group are a finite complete rewriting system, and obey a certain ordering, this is simply a completion problem. The input, in KBMAG format, is:

```
gens {a,b,c,d,e,f}
inverses {case_change}
rels {
  [b,a]=c,      [B,a,a,a],
  [c,a]=d,      [a,b,b,b],
  [c,b]=e,      [A,b,b,b],
  [d,b]=f,      [a,a*b,a*b,a*b],
  [b,a,a,a],   [A,a*b,a*b,a*b]
}
```

This is a fairly easy completion and returns exactly the set of rules obtained by the nilpotent quotient algorithm, showing that $E(2, 3)$ is nilpotent and that these five instances of the law are enough to define $E(2, 3)$.

There are obvious related problems for the groups $E(2, n)$ which are two generator groups under the law

$$[x, y, \dots, y] = e$$

where y is repeated n times. It is conjectured that $E(2, n)$, for some higher values of n are also nilpotent but this question remains open, in particular for the case $n = 4$. If we could treat the laws directly, instead of via instances, as discussed in Section 6.7, then we should be able to solve problems such as these.

6.9 Conclusions

This section has shown that many classes of problems which can be attacked with specialised programs in group theory can also be solved by general purpose theorem provers. There is, not surprisingly, a heavy performance price to pay by using general purpose systems. A number of problem classes which can not be attacked by conventional computational means have also been highlighted. These offer more hope to the automated deduction community: the generality of the techniques make it possible to guide the search in an intelligent way.

7 Tietze Transformations

7.1 Introduction

In this chapter, Tietze transformations (see Tietze [122] and Johnson [67]) will be defined. We can then discuss the relationship between Knuth-Bendix completion programs and group presentation simplification programs: this link will then be used to provide possible heuristics for more efficient and effective implementations of these programs.

7.2 Definitions for Tietze Transformations

The definition and following theorem show that a presentation of a group can be manipulated by a sequence of operations and the result of these manipulations is the presentation of an isomorphic group and every such presentation arises in this way.

Definition 7.1 Suppose we have a presentation $G = \langle X \mid R \rangle$. Each Tietze transformation T_i ($i = 1, \dots, 4$) transforms it into a presentation $G = \langle X' \mid R' \rangle$ in accordance with the following definitions:

T1 If $r \in X^*$ and $r = 1$ is a relation which holds in G , then let $X' = X$ and $R' = R \cup \{r\}$

T2 If $r \in R$ is such that the relation $r = 1$ holds in the group $G = \langle X \mid R \setminus \{r\} \rangle$ then let $X' = X$ and $R' = R \setminus \{r\}$

T3 If $w \in X^*$ and $z \notin X$ put $X' = X \cup \{z\}$ and $R' = R \cup \{wz^{-1}\}$

T4 If $z \in X$ and $w \in (X \setminus \{z\})^*$ such that $wz^{-1} \in R$ then substitute w for z in every other element of R to get R_1 , so we get $X' = X \setminus \{z\}$ and $R' = R_1$

So the Tietze transformations correspond to adding a relation, removing a relation, adding a generator or removing a generator. The following theorem is standard in group theory, for a proof see, for example, Johnson [67] or Tietze [122]:

Theorem 7.2 *Given any two finite presentations*

$$\langle X \mid R \rangle \quad \text{and} \quad \langle Y \mid S \rangle$$

for a group G , one can be transformed into the other by means of a finite sequence of Tietze Transformations

Despite this, the isomorphism problem for finitely presented groups is undecidable (see e.g. Book and Otto [9]).

7.3 Tietze Transformations and Completion

There has been work on showing that certain groups have either finite or infinite convergent rewriting systems depending on which generators are chosen for the group (Jantzen [66]) and depending on which orderings are chosen (see, e.g., Martin [91]). A procedure which automatically introduces 'new generators' will be stated, and it will be shown that this gives a heuristic to transform an infinite convergent rewriting system into a finite one. We do this by modifying the ordering: a new generator is introduced by putting it at the bottom of the precedence of the generators.

7.3.1 Related Work

There is a large body of work on divergence of completion procedures and infinite rewriting systems. This section will highlight some of the most notable work on this subject from both the rewriting and computational group theory communities.

Hermann [57] investigates the structure of rewrite systems and the way that critical pairs are formed in order to characterise two forms of divergence: forward and backward crossed rewrite systems. A number of ways to avoid divergence are also discussed. One of these suggests adding in inductive theorems to the system, but no way to do this explicitly is stated. Another is to change the chosen ordering; Hermann calls this 'backtracking'. Martin [91] gives a comprehensive description of orderings for string rewriting systems and the finite and infinite rewrite systems which can result from a choice of ordering. Many interesting questions about finding 'good' rewriting systems are asked.

Sattler-Klein [108] also investigates the role of critical pairs in divergent rewriting systems and characterises a number of conditions for a completion procedure to diverge. Interestingly, Sattler-Klein gives an example which shows that given a diverging completion pro-

cedure, there need not exist any rule which generates infinitely many new rules by critical pair creation.

The term rewriting community has worked to solve the divergence problem by using a number of methods from artificial intelligence (AI) and rewriting itself. Hermann and Kirchner [58] introduce a way to automatically detect divergence and automatically generate meta-rules from syntactic conditions of divergence, building on the work of Hermann [57] and Kirchner [70]. Thomas and Jantke [120], Lange [80] and Jantke and Lange [65] use a method called inductive inference from AI to generalise infinite families of rewrite rules from a finite number of instances of the generalised rule.

Chen *et al* [27] develop recurrence rewriting to solve the divergence problem. Recurrence terms are introduced in order to identify similarities in the structure of terms. Recurrence relations are created to encode this structural similarity.

Thomas and Watson [121] introduce a method which enriches the signature of the original rewriting system by creating new sorts and new operator arities.

As was mentioned in Section 6.3, there is a whole area in group theory looking at properties which characterise whether a group has a finite complete rewriting system. Computational group theory also has some interesting results and examples to do with the divergence problem. Jantzen [66] shows that different choices of generating sets for a group can result in an infinite set of rewrite rules rather than a finite one, but gives no automatic method to achieve this.

Needham [98] describes a method to deal with ‘infinite families’ of rewrite rules. The procedure identifies these families and each of them is parameterised by a single rewrite rule. He also shows that his procedure can create a ‘canonical’ system for non- FP_∞ groups.

7.3.2 Generator Introduction

Intuitively, the Knuth-Bendix completion procedure for strings can be seen as $T1$, $T2$ and (to a lesser extent) $T4$. $T1$, the adding in of a new relator to the group, is similar to deducing a critical pair (and then ordering it) in completion. $T2$, the deleting of a relator which can be proved true by the other relators in the presentation, is equivalent to an equation or rewrite

rule being rewritten and then deleted. If x is a generator, w is a word in which x does not occur and a completion procedure generates a rewrite rule $x \rightarrow w$, then we can think of x as being 'rewritten out'. When the rewrite system and set of equations are normalised with respect to $x \rightarrow w$, then the only occurrence of x will be in $x \rightarrow w$. The lexicographic recursive path ordering (see Sims [113] and Dershowitz [38]), orders the rule this way when there is a precedence on the generators such that

$$x > \{ \text{generators occurring in } w \}$$

This is equivalent to Tietze transformation $T4$.

We will first give an example to justify our approach. The example comes from Jantzen [66], who showed that the group in question had an infinite convergent rewriting system given one set of generators, and a finite convergent rewriting system given another.

Example 7.3 Let $G = \langle a, b \mid a * b * b * a \rangle$. If one attempts to complete $\{a * b * b * a = 1\}$ with the lexicographic recursive path ordering with precedence $b > a$ one finds that there are infinitely many rules of the form $b * b * (a * b)^i * a * a \rightarrow (a * b)^i$. It is important to note here that this is a monoid presentation for the group; i.e. we are not using any of the natural inverse relations. An obvious choice of substring to 'rewrite out' would seem to be $a * b$, which occurs infinitely many times. Suppose we added $a * b \rightarrow c$ into the rewriting system. Running the completion procedure one gets the following 'new' rewriting system:

$$\begin{aligned} R' = \{ & c * b \rightarrow b * b * a, \\ & b * b * c^i * a * a \rightarrow c^i, \\ & a * b \rightarrow c \} \end{aligned}$$

i.e. this rewriting system is infinite. It is also possible to see that the 'same' critical pairs are being calculated. So to choose a 'useful' substring to 'rewrite out' of the presentation, it would seem wise to check the overlaps which are creating the critical pairs. In this case the overlap of interest is

$$b * b * (a * b)^{i+1} * a * a \leftarrow a * b * b * b * (a * b)^i * a * a \rightarrow (a * b)^{i+1}$$

So we need a way to 'destroy' these critical pairs; i.e. choose a string which will stop $a * b * b$ from overlapping with $b * b * (a * b)^{i+1} * a * a$ in the way mentioned above. One possible

choice is $b * b = c$. Completing $\{a * b * b * a = 1, b * b = c\}$ with the lexicographic recursive path ordering with precedence $b > a > c$, one obtains the finite rewriting system S :

$$S = \{b * b \rightarrow c, b * c \rightarrow c * b, a * c \rightarrow c * a, c * a * a \rightarrow e, b * a * a \rightarrow a * a * b\}$$

Of course, this heuristic could change a finite system into an infinite one. A (ground) complete rewrite system is so useful, however, that it is often worth exploring this possibility.

It should be noted at this point that adding in the generator is the Tietze transformation $T3$. Thus it follows straight from Theorem 7.2 that the presentation with the newly defined generator is isomorphic to the original group.

Needham [98] needs ways to determine when an infinite family of rewrite rules are being created by overlaps. Possible rules for deciding when an infinite family of rewrite rules is being generated are stated. We wish, though, to define a heuristic, and so the rules we can use to introduce new generators are much weaker than the rules needed in the work of Needham [98].

We now give an informal description of the procedure which we are using. This depends upon being able to analyse how the infinite families of critical pairs are being generated, and being able to 'extend' a reduction ordering $>$ in which there is a precedence on the generators. We will analyse families of overlaps in order to determine where infinite families of critical pairs originate. We simply choose a substring of an overlap, which, if it is renamed, will ensure that this overlap will not occur again. So in Example 7.3 there is an infinite family of critical pairs $b * b * (a * b)^i * a * a = (a * b)^i$ and an infinite family of overlaps $a * b * b * b * (a * b)^i * a * a$. Thus the overlap will not occur again if we choose to 'rewrite out' the substring $b * b$. In order to introduce a generator, a new constant c is defined and c is put at the bottom of the precedence in the reduction ordering. Thus the rule is oriented $b * b \rightarrow c$, and the substring $b * b$ is rewritten out of the presentation. This is called *breaking the overlap*.

The procedure which is used will now be described.

1. given a set of equations and a reduction ordering which includes a precedence on the operators, start the completion procedure.
2. if the procedure seems to diverge, break an overlap which causes this divergence

3. continue with completion procedure

We can think of all of the equations which define potential new generators as always having existed, but there are times when modifying the ordering so that they are then 'of use' makes sense. If the rewrite rules are convergent on a set of generators, the rest of the potential new generators can be added at the top of the precedence in the reduction ordering. This will mean that they will be oriented

$$\text{new generator} > \text{word in 'already defined' generators}$$

and automatically this rewriting system will be convergent, as the left hand sides of these rewrite rules do not overlap with the left hand side of any other rewrite rules. Hence, in a sense, the finiteness of a rewriting system depends entirely upon the 'choice of ordering'.

Example 7.4 We will now look at a particularly interesting family of examples from Martin [91]. The groups in question have the following presentation for $k \geq 1$:

$$H_k = \langle a, b \mid b^{2k} = 1, aba = b^{2k-1}, abb = bba \rangle$$

H_k has 2 finite rewriting systems, and the following $k - 1$ infinite rewriting systems. The infinite rewriting systems are defined as follows for $1 \leq p \leq k - 1$ and $1 \leq t \leq \infty$:

$$\begin{aligned} R_p = \{ & b^{2(k-p)+1} \rightarrow a^p b a^p, \\ & a^{p+1} b a^{p+1} \rightarrow b^{2(k-p-1)+1}, \\ & abab \rightarrow 1, \\ & baba \rightarrow 1, \\ & abb \rightarrow bba, \\ & aba^t b \rightarrow ba^t ba, \\ & b^2 a^t b a \rightarrow a^{t-1} b, \\ & b^{2(k-p)} a^t b \rightarrow a^{t+p} b a^p \} \end{aligned}$$

R_1 arises when using the recursive path ordering with $b > a$. There are also $k - 1$ infinite convergent rewrite systems R'_p which have the words in the above presentation reversed. A preliminary analysis of the rules of R_p shows that there are the following critical pairs (where $cp(l_1, l_2)$ denotes that there is a critical pair between rule $l_1 \rightarrow r_1$ and rule $l_2 \rightarrow r_2$):

7 TIETZE TRANSFORMATIONS

1. $cp(abb, bba^{(t-1)}b) = bba^tba \rightarrow a^t b$
2. $cp(baba, aba^t b) = bba^tba \rightarrow a^t b$
3. $cp(abab, aba^t b) = bba^tba \rightarrow a^t b$
4. $cp(abb, b^{2(k-p)}a^t b) = b^{2(k-p)}a^{t+1}b \rightarrow a^{t+p}ba^p$

So a first new generator to introduce may be $baa = d$. Making the operator precedence $b > a > d$ with the lexicographic recursive path ordering, one can complete H_k to a finite, convergent system:

$$\begin{aligned} C = \{ & d^{2k} \rightarrow 1, \\ & add \rightarrow dda, \\ & ada \rightarrow ddd, \\ & b \rightarrow d^{2(k-2)}a^2d \} \end{aligned}$$

Future work will concentrate on investigating the idea of adding in generators in a more theoretical framework. In particular the work of Bachmair [5] in proof orderings will be used to prove theoretical results about such a system.

7.4 Tietze Transformations and Group Presentation Simplification

Automated group presentation simplification programs are a tool much used in computational group theory to simplify the very large presentations produced by some algorithms, permitting further human or machine-assisted investigation. Typically, presentations which require simplification are the result of finding a presentation of a subgroup H of finite index in a finitely presented group G via the Reidemeister-Schreier procedure. This procedure tends to produce presentations which have many redundant generators and relations. A simplification of such a presentation is essential if one wants to apply other computational techniques to it. No precise definition of 'simplification' is given, but that usually accepted in practice is the following:

1. a reduction in the number of generators,
2. a reduction in the total length of the relators and

7 TIETZE TRANSFORMATIONS

3. a reduction in the number of relators.

This corresponds roughly to an increase in the practicability of applying well-known tools (such as coset enumeration or the nilpotent quotient algorithm) to the presentation. It should be noted that there is really *no* satisfactory definition of simplification for the presentation of a finitely presented group. A presentation may have many generators, many relations and a great overall length, but it may also have the property that it is a finite convergent rewriting system. This means that it is possible to ‘read off’ important structural information about the group. The standard tools from group theory, e.g. the coset enumeration, nilpotent quotient algorithms, may not be useful in studying such a presentation.

Applying a simplification procedure to a presentation is generally much less time-consuming than subsequent analysis of the presentation, so that even small additional simplifications at significant costs in simplification time are worthwhile. Sims [112] proposes a presentation simplification program using Knuth-Bendix completion. In general, however, this approach is not widely used. This is because the presentations are simplified with respect to a given term ordering, and not the three criteria mentioned above.

Group presentation simplification programs (see e.g. Havas *et al* [54]) tend to be based on the Tietze Transformations T_2 , T_3 and T_4 . This can be seen in large group theory packages such as GAP [109]. T_2 and T_4 are done automatically by these programs, but T_3 is used interactively.

We propose an additional heuristic for a Tietze transformation program which uses the idea of the ‘critical pair’ from Knuth-Bendix completion. When the presentation simplification program can do no more, we convert the relators into a set of rewrite rules and calculate a set of critical pairs associated with those rewrite rules. This is then converted back to the language of relators and the resulting presentation is again simplified. This will help in the automatic simplification of a given presentation. As a justification for this approach, we have the following example.

Example 7.5 We investigate the example $F(2, 5)$ which was referred to in Chapter 6. We state the presentation again for ease of reference:

$$F(2, 5) = \langle a, b, c, d, e \mid a * b = c, b * c = d, c * d = e, d * e = a, e * a = b \rangle.$$

7 TIETZE TRANSFORMATIONS

The GAP group presentation simplification program eliminates b , d and e and returns the presentation:

$$G = \langle a, c \mid a^{-1} * c^2 * a^{-1} * c * a^{-2}, c * a^{-1} * c^2 * a * c^{-1} * a \rangle$$

Feeding this into a Knuth-Bendix program for strings (see Holt [60]) with the recursive path ordering with the precedence $c > a$, and completing the rewriting system we obtain the following set of rewrite rules:

$$\begin{aligned} \text{Rewrite Rules} = \{ & c^{-1} \rightarrow a * a * a * a * a * a, \\ & a^{-1} \rightarrow a * a * a * a * a * a * a * a * a * a * a, \\ & c \rightarrow a * a * a * a * a, \\ & a * a * a * a * a * a * a * a * a * a * a * a \rightarrow 1 \} \end{aligned}$$

The resulting simplified presentation is

$$G' = \langle a \mid a^{11} \rangle$$

Of course, the presentation could have been fed directly to a Knuth-Bendix program; experimental data (see Chapter 6), however, shows that the approach of simplifying the presentation first is more efficient. There is also a question of which critical pairs to calculate. The critical pairs which are added into the presentation are defined below.

Definition 7.6 Given a presentation $G = \langle X \mid R \rangle$, with $BA, AC^{-1} \in R$, the relator BC is called a *relator critical pair* in G .

Lemma 7.7 Given a presentation $G = \langle X \mid R \rangle$, any relator critical pair of relators $r_1 = BA, r_2 = AC^{-1} \in R$ is a relator which holds in G

Proof— It is possible to form the overlap BAC^{-1} , which can be rewritten in two ways to obtain $B = C^{-1}$. Therefore BC is a relator. \square

This is equivalent to a critical pair in Knuth-Bendix completion, and adding the critical pair to R is an instance of the Tietze transformation $T1$. For experimental purposes, we chose to convert each of the relators $r \in R$ into a rewrite rule and take critical pairs in the usual way:

a number of orderings can be chosen and different orderings will correspond to different sets of relator critical pairs being computed. There is also a question of symmetrising the rules by multiplying through with the inverses of the generators which occur. Experiments, however, suggest that this can make the rewrite system unduly large. We choose to look at the rewrite rules $r \rightarrow e$ for each $r \in R$ and calculate the critical pairs of these rules.

We have the following six step algorithm; given a presentation $G = \langle X \mid R \rangle$, we simplify the presentation as follows:

1. Simplify the presentation with a standard group presentation simplification program to obtain $G_1 = \langle X_1 \mid R_1 \rangle$
2. Convert each $r \in R_1$ into a rewrite rule $r \rightarrow e$
3. Calculate all critical pairs of the rewrite rules from 2
4. Take the union of the rules and equations from 2 and 3
5. For each equation and rewrite rule in 4, convert to relators: e.g. $r_1 \rightarrow r_2$ becomes $r_1(r_2)^{-1}$ and $r_3 = r_4$ becomes $r_3(r_4)^{-1}$; thus obtain a set of relators R_2
6. Simplify presentation $G_2 = \langle X_1 \mid R_2 \rangle$

In practice it is useful to iterate (2) – (6) a number of times in order to obtain a reasonably simplified presentation. We now give some results as to how the new heuristic works when used in conjunction with the GAP presentation simplification program. We call our presentation simplification heuristic *TT-KB*: this was implemented in GAP in order to be able to use the efficient presentation simplification program which is implemented in it. The above algorithm is used and the best presentation after 3 iterations is chosen. This is compared to the automatically derived GAP simplified presentation; these results appear in Table 6. The experiments were executed on a SPARC IPX with 64Mb of memory, and the cpu times are in seconds.

The presentations considered occur in Havas *et al* [54]. The Reidemeister-Schreier algorithm is used to generate a presentation for the subgroup $H(n) = \langle a, b^2 \rangle$ of index $|2n+3|$ (for $n = 5, \pm 7, \pm 8, \pm 10$) in the group $G(n) =$

$$\langle a, b \mid (a * b * a * b^{-1})^n * (b * a^{-1} * b * a * b^{-1} * a)^{-1}, a * b^2 * a^{-1} * b * a^2 * b^{-1} \rangle$$

Table 6: Comparing Automatic Group Presentation Simplification Programs

gen	number of generators
rel	number of relators
len	sum of lengths of relators
cpu	cpu time in seconds

Group	Original			GAP				TT-KB			
	gen	rel	len	gen	rel	len	cpu	gen	rel	len	cpu
$H(5)$	14	26	238	2	3	37	0.9	2	3	37	1.5
$H(7)$	18	34	378	4	20	1,189	2	2	5	112	30
$H(-7)$	12	22	228	2	12	613	1.3	2	12	598	11
$H(8)$	20	38	466	4	22	1,095	3	3	22	3,258	68
$H(-8)$	14	26	294	4	16	474	1.3	4	16	474	9
$H(10)$	24	46	648	2	24	8,391	16	2	24	8,391	575
$H(-10)$	18	34	450	4	20	818	2.3	4	21	725	21.3

The results in Table 6 show that the TT-KB approach does work in the area in which presentation simplification programs are used. The gains are modest, and the CPU times longer, but, as remarked above, this may well be worthwhile in practice. This method has the advantage that it can be varied to use the expertise built up both in group theory, where presentation simplification algorithms have been designed, and also the theoretical computer science community which has expertise in Knuth-Bendix completion.

8 Conclusions and Future Work

This thesis has introduced the notion of a proof diagram. A proof diagram represents the execution trace of proof procedures such as Knuth-Bendix completion and resolution from a given set of clauses. Proof diagrams are simply sets of inferences, so conditions necessary to give these structures meaning must be stated; for instance, the definition of a *proof* of a fact was given. This corresponds in an obvious way to the notion of proof of a fact which is implicit in a proof procedure; i.e. we have a proof of a fact ϕ if there is a justified path from some subset of the axioms to ϕ .

We have also defined a proof graph. This is obtained by mapping a proof diagram onto a directed graph. The proof graph makes it possible to reason about local properties of proofs of facts; for instance, if a 'is an ancestor of' b in the proof diagram then there is a path from a to b in the proof graph. We are then able to reason about the dependencies of the facts which occur in a proof diagram.

The structures of the proof diagram and the proof graph make it possible to reason about proofs of a fact ϕ which are minimal with respect to a proof Q of ϕ . The intuition for this definition comes from two sources:

- we do not want any strongly irrelevant information to be used in a proof of a fact
- we wish to miss out complex intermediate substructure

The first criterion above is formalised by the definitions of direct proof of a fact and valid proof. The notion of homeomorphic embedding characterises the 'throwing away' of complex pieces of proof of a fact. Intuitively, a proof P of ϕ is minimal with respect to a proof B of ϕ if there is no 'short cut' to ϕ . A homeomorphic embedding relation for rooted DAGs is introduced so that this can be formalised for proof graphs.

Chapter 3 introduced the definitions for proof diagrams and proved some simple consequences of these definitions. Most importantly, if $Min(P, B, a)$ is true, then $\mathcal{G}(P)$ is a DAG.

This thesis has also investigated the Knuth-Bendix completion procedure. It has been shown that this is an interesting algorithm to investigate as there are a number of choices which

have to be made in any particular implementation of a completion procedure. For instance, as can be seen in Chapter 6, which critical pairs are calculated and which ordering is chosen can have a great effect on the efficiency of the completion program. A new inference system for completion has been introduced in Chapter 4. This is similar to the inference system of Bachmair, but has some interesting additional properties. In particular, it has been shown that critical pair criteria are characterised by this inference system. These are criteria for detecting redundant critical pairs, and so are possibly a powerful heuristic for a completion procedure. It has also been shown that the relation ‘is an ancestor of’ is a strict partial order on the facts of a proof diagram associated with a completion procedure. We have also investigated possible heuristics used in completion by looking at proof diagrams. Through analysis of these proof diagrams we have shown that it is possible to invent heuristics which can be implemented in a machine assisted theorem prover such as Otter.

Derived rules of inference are the building blocks of complex heuristics used in LCF-style theorem provers such as Nuprl, Hol and Isabelle. These heuristics are called *tactics*. In Chapter 5, derived inferences have been investigated in the context of proof graphs. They correspond, in a natural way, to a property of the paths in $\mathcal{G}(P)$ when $Min(P, B, a)$ is true. All paths from the root of $\mathcal{G}(P)$ to the goal node must be ‘covered’.

We have used these derived inferences to formalise a rewrite rule for proof diagrams; we call this a *pd-rewrite rule*. This means we can replace a complex part of a proof of a fact by a derived rule. A side condition is defined so that we are able to rewrite valid proofs into valid proofs. This is motivated by the notion of a transformation tactic in the Nuprl proof development system. We have also briefly discussed the possibility of automating the search for useful derived inference rules.

Chapter 6 has shown that many of the calculations normally done with special-purpose group theory programs can be completed with today’s general-purpose completion theorem provers, but at such a large performance penalty that the specialised programs are likely to remain with us for some time. On the other hand, the general-purpose systems allow us to address some problems that are beyond the scope of more specialised tools (as in Section 6.7) and to apply additional mathematical insight (as in Section 6.8.1).

The work presented in Chapter 7 has shown that group presentation simplification programs can be enhanced by adding in a simple idea from Knuth-Bendix completion: the critical pair.

Also, the idea of generator introduction for Knuth-Bendix completion has been discussed.

8.1 Future Work

The original motivation for this work was to understand how heuristics can help guide search in machine assisted theorem provers. This is a rich and complex area: there are many algorithms and decision procedures to which the theory of proof diagrams could be applied.

Of particular interest is the possibility of automatically deriving useful heuristics. This seems a likely area in which the theory of proof diagrams could be extended. An automatic way of generating new heuristics for theorem provers would certainly be a powerful tool.

A number of the problems mentioned in the abstract setting we have presented for proof diagrams we can also conclude from Chapter 6. Therefore we identify a number of features which would make general-purpose completion systems more suitable for the kinds of problems from group theory which we have examined:

- Better user control of heuristics and orderings. Mathematical information that cannot easily be stated in the first order language of the problem can often be exploited in this way.
- The facility to distinguish certain rules as special. Many of the optimisations of the specialised Knuth-Bendix program amount to forming critical pairs with certain rules at particular times.
- Efficient handling of very large rewriting systems and very deeply nested terms. Many interesting problems seem not to have short or direct solutions.
- Implementation of special matching and unification procedures for associative operators as mentioned in Section 6.2.3.

One approach to the sort of computation investigated in Chapter 6 is to combine the flexibility of general-purpose systems with the speed of specialised programs by allowing the general-purpose systems to delegate suitable sub-problems to a special-purpose system. One approach to this is the 'cooperating decision procedures' being added to LP at present,

8 CONCLUSIONS AND FUTURE WORK

another is the use of tactic languages which have the ability to invoke external programs and use their results as either an oracle or a source of hypotheses. It would be very interesting to try and use these methods to combine a general-purpose theorem prover with existing group-theoretic tools.

It would also be interesting to formalise the idea of Tietze transformations in an inference rule setting. This will not only show how completion relates to the notion of the Tietze transformations, but it also seems likely that the divergence problem for string completion will be better understood.

References

- [1] Andrews, P.B., Transforming Matings into Natural Deduction Proofs, in Ed. W. Bibel and R. Kowalski, Proc. CADE-5, LNCS 87, Springer-Verlag, 1980, pp. 281-292.
- [2] Newsletter 28 of the Association for Automated Reasoning, January 1995.
- [3] Anick, D.J., On the Homology of Associative Algebra, Transactions of the AMS 296, 1986, pp. 641-659.
- [4] Aschbacher, M., Finite Group Theory, Cambridge Studies in Advanced Mathematics 10, Cambridge, 1986.
- [5] Bachmair, L., Canonical Equational Proofs, in the series Progress in Theoretical Computer Science, Birkhäuser, 1991.
- [6] Bachmair, L. and Dershowitz, N., Critical Pair Criteria for Completion, Journal of Symbolic Computation 6, 1988, pp.1-18.
- [7] Bachmair, L., Dershowitz, N. and Plaisted, D., Completion without Failure, in Eds. H. Ait-Kaci and M. Nivat, Proc. Coll. on Resolution of Equations in Algebraic Structures 2: Rewriting Techniques, Academic Press, New York, 1989, pp. 1-30.
- [8] Barendregt, H.P., van Eekelen, M.C.J.D, Glauert, J.R.W., Kennaway, J.R., Plasmeijer, M.J. and Sleep, M.R., Term Graph Rewriting, in Eds. J.W. de Bakker, A.J. Nijman and P.C. Treleaven, Proc. PARLE '87, LNCS 259, Springer-Verlag, 1987, pp. 141-158.
- [9] Book, R.V., and Otto, F., String-rewriting systems, Springer-Verlag Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993.
- [10] Bosma, W. and Cannon, J., Handboook of Magma Functions, Dept. of Pure Mathematics, University of Sydney, 1993.
- [11] Boyer, R.S., and Moore, J.S., A Computational Logic, Academic Press, New York, 1979.
- [12] Boyer, R.S., and Moore, J.S., Proof Checking the RSA public key Encryption Algorithm, American Mathematical Monthly 91(3), 1984, pp. 181-189.

REFERENCES

- [13] Brown, K.S., The Geometry of Rewriting Systems: A Proof of the Anick-Groves-Squier Theorem, in Eds. G. Baumslag and C.F. Miller, *Algorithms and Classification in Combinatorial Group Theory*, Springer-Verlag, 1991, pp. 137–163.
- [14] Buchberger, B., Gröbner bases: an algorithmic method in Polynomial Ideal Theory, in Ed. N.K. Bose *Recent Trends in Multidimensional Systems Theory*, Reidel, 1985, Chapter 6.
- [15] Buchberger, B., A Criterion for detecting unnecessary reductions un the construction of Gröbner bases, in Ed. W. Ng, *Proc. EUROSAM '79, LNCS 72*, Springer-Verlag, 1979, pp.3–21.
- [16] Bündgen, R., e-mail communication, 1994.
- [17] Bündgen, R., *Term Completion Versus Algebraic Completion*, Technical report WSI 91-3, Universität Tübingen, Tübingen, 1991.
- [18] Bündgen, R. and Walter, J., *The ReDuX User Guide (Version 1.3)*, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany, 1993.
- [19] Bundy, A., The Use of Explicit Plans to Guide Inductive Proofs, in Ed. E. Lusk and R. Overbeek, *Proc. CADE-9, LNCS 310*, Springer-Verlag, 1988, pp. 111–120.
- [20] Cameron, P.J., *Combinatorics: Topics, Techniques, Algorithms*, CUP, 1994.
- [21] Campbell, C.M., Coxeter, H.S.M, and Robertson, E.F., Some Families of Finite Groups having two Generators and two Relations, *Proceedings of the Royal Society London A357*, 1977, pp. 423–438.
- [22] Campbell, C.M., and Robertson, E.F., Finitely Presented Groups of Fibonacci type II, *Journal of the Australian Mathematical Society* 28, 1979, pp. 250–256.
- [23] Campbell, C., lecture notes from Computational Group Theory course, St Andrews, 1994.
- [24] Cannon, J.J., Dimino, L.A., Havas, G., and Watson, J.M., Implementation and Analysis of the Todd-Coxeter Algorithm, *Math. Comp.*, 27, 1973, pp. 463–490.
- [25] Chang, C.-L. and Lee, R. C.-T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, London, 1973.

REFERENCES

- [26] Char, B.W., Maple V language reference manual. Springer-Verlag, 1991.
- [27] Chen, H., Hsiang, J. and Kong, H-C., On Finite Representations of Infinite Sequences of Terms, in Ed. M. Okada, Proc. 2nd CTRS, LNCS 516, Springer-Verlag, 1990, pp.100–114.
- [28] Comon, H., Inductionless Induction, lecture notes for the 2nd International Summer School in Logic for Computer Science, Chambery, 1994.
- [29] Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T. and Smith, S.F., Implementing Mathematics with the Nuprl Proof Development System, Prentice Hall, New Jersey, 1986.
- [30] Conway, J.H., Advanced Problem 5327, American Mathematical Monthly 72, 1965, p. 91.
- [31] Conway, J.H., Solution to Advanced Problem 5327, American Mathematical Monthly 74, 1967, pp. 91–93.
- [32] Conway, J., *et al*, The Atlas of Finite Simple Groups, Oxford University Press, 1987.
- [33] Cox, D., Little, J., and O'Shea, D., Ideals Varieties and Algorithms, Undergraduate Texts in Mathematics, Springer-Verlag, 1993.
- [34] Davenport, J., Systems of Non-Linear Equations, talk given at Workshop on Symbolic Rewriting Techniques, Monte Verita, 1995.
- [35] Denzinger J., and Schulz, S., Learning Domain Knowledge to Improve Theorem Proving, in Ed. M. McRobbie and J. Slaney, Proc. of CADE-13, LNAI 1104, Springer-Verlag, 1996, pp.62–77.
- [36] Dershowitz, N., Orderings for Term Rewriting Systems, Theoretical Computer Science 17, 1982, pp. 279 – 301.
- [37] Dershowitz, N., and Jouannaud, J-P., Rewrite Systems, Handbook of Theoretical Computer Science vol. B, North-Holland, 1989, Chapter 6.
- [38] Dershowitz, N., Kaplan, S., Plaisted, D.: Rewrite, Rewrite, Rewrite, Rewrite, Rewrite ... , Theoretical Computer Science 83, 1991, pp. 71–96.

REFERENCES

- [39] Desimone, R.V., Learning Control Knowledge within an Explanation-Based Learning Framework, Ph.D. thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1989.
- [40] Epstein, D.B.A., Holt, D.F. and Rees, S.E., The Use of Knuth-Bendix Methods to Solve the Word Problem in Automatic Groups, *Journal of Symbolic Computation* 12, 1991, pp. 397–414.
- [41] Fitting, M., First Order Logic and Automated Theorem Proving, Texts and Monographs in Theoretical Computer Science, Springer-Verlag, 1990.
- [42] Fujita, M., Slaney, J., and Bennett, F., Automatic Generation of Some Results in Finite Algebra, Proc. of IJCAI '93, Chambéry, 1993, pp.52–57.
- [43] Gallier, J., What's so Special about Kruskal's tree theorem and the ordinal Γ_0 ?, A Survey of Some Results in Proof Theory, *Annals of Pure and Applied Logic* 53, 1991, pp. 199–260.
- [44] Gallier, J., Logic for Computer Science: Foundations of Automated Theorem Proving, Wiley, New York, 1987.
- [45] Garland, S., and Guttag, J.V., A Guide to LP, The Larch Prover, MIT., 1993.
- [46] Gordon, M.J.C. and Melham T.F., Introduction to HOL, CUP, 1993.
- [47] Gordon, M.J.C., Milner, R., and Wadsworth, C.P., Edinburgh LCF: A Mechanised Logic of Computation, LNCS 78, Springer-Verlag, 1979.
- [48] Green, J., Sets and Groups: A First Course in Algebra, RKP, 1988.
- [49] Groves, J.R.J., Rewriting Systems and Homology of Groups, in Ed. L.G. Kovács, Groups—Canberra 1989, Lecture Notes in Mathematics 1456, Springer-Verlag, Heidelberg, 1990, pp. 114–141.
- [50] Groves, J.R.J. and Smith, G.C., Soluble Groups with a Finite Rewriting System, Proc. Edinburgh Math. Soc 36, 1993, pp. 283–288.
- [51] Haken, A., The Intractibility of Resolution, *Theoret. Comput. Sci.* 39, 1985, pp. 297–308.

REFERENCES

- [52] Hamilton, A.G, *Logic for Mathematicians*, CUP, 1989.
- [53] Havas, G., *Coset Enumeration Strategies*, Technical report 200, Key Centre for Software Technology, University of Queensland, 1991.
- [54] Havas, G., Kenne, P.E., Richardson, J.S. and Robertson, E.F: A Tietze Transformation Program, in Ed. M.D. Atkinson, *Computational Group Theory*, Academic Press, London, 1984, pp. 69–73.
- [55] Havas, G. and Newman, M.F., *Applications of Computers to Questions like those of Burnside*, In Ed. J.L. Mennicke, *Burnside Groups*, Lecture Notes in Mathematics vol. 806, Springer-Verlag, Berlin, 1980, pp. 211–230.
- [56] Heineken, H., *Engelsche Elemente der Länge drei*, *Illinois J. Math.* 5, 1961, pp. 681–707.
- [57] Hermann, M., *Vademecum of Divergent Term Rewriting Systems*, CRIN research report 88-R-082, Nancy, 1988.
- [58] Hermann, M., and Kirchner, H., *Meta-rule synthesis from crossed rewrite systems*, in Ed. M. Okada, *Proc. 2nd CTRS, LNCS 516*, Springer-Verlag, 1990, pp.143–154.
- [59] Higman, G., *A Finitely Generated Infinite Simple Group*, *J. London Math. Soc.* 26, 1951, pp. 61–64.
- [60] Holt, D.F., *Knuth–Bendix in Monoids, and Automatic Groups (a package of programs)*, Mathematics Institute, University of Warwick, England, 1995.
- The file `kbmag.tar.Z` can be obtained by anonymous ftp in `/people/dfh/` at `lomond.warwick.ac.uk`. This system is also available as a GAP share package in `/pub/incoming/` at `ftp.math.rwth-aachen.de`.
- [61] Humphreys, J.E., *Reflection Groups and Coxeter Groups*, *Cambridge Studies in Advanced Mathematics 29*, Cambridge, 1990.
- [62] Jackson, P., *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*, Ph.D. Thesis, Cornell University, 1995.
- [63] Jäger, G, *First Order Logic Programming*, working material for course on First Order Logic Programming, Marktoberdorf Summer School, Marktoberdorf, 1995.

REFERENCES

- [64] Janet, M., Sur les Systèmes d'équations aux dérivées partielles, *Journal de Mathématiques* III(8), 1920.
- [65] Jantke, K.P., and Lange, S., inductive completion for transformation of equational specifications, in Ed. H. Ehrig, K.P. Jantke, F. Orejas and H. Reichel 7th Workshop on Specification of Abstract Data Types, LNCS 534, Springer-Verlag, 1991, pp. 117–140.
- [66] Jantzen, M., A note on a Special One-rule semi-Thue system, *Inf. Proc. Letters* 21, 1985, pp. 135–140.
- [67] Johnson, D.L., *Presentations of Groups*, Cambridge University Press, Cambridge, 1990.
- [68] Johnson, D.L., Walmsey, J.W., and Wright, D., The Fibonacci Groups, *Proceedings of the London Mathematical Society* 29, 1974, pp. 577–592.
- [69] Kapur, D. and Narendran, P., A Finite Thue System with Decidable Word Problem and Without Equivalent finite Canonical System, *Theoret. Comput. Science* 35, 1985, pp. 337–344.
- [70] Kirchner, H., Schematization of Infinite sets of rewrite rules generated by divergent completion processes, *Theoret. Comput. Sci.* 62, 1989, pp.303–332.
- [71] Klop, J.W., Term Rewriting Systems, *Handbook of Logic in Computer Science* vol. 2, Oxford Science Publications, 1992 pp. 1–116.
- [72] Knuth, D.E., and Bendix, P.B., Simple Word Problems in Universal Algebra, in Ed. J.Leech *Computational Problems in Abstract Algebra*, Pergamon Press, 1970.
- [73] Kropholler, P.H., On groups of type $(FP)_{\infty}$, *J. Pure Appl. Algebra*, 90, 1993, pp. 55–67.
- [74] Kruskal, J.B., Well-Quasi-Orderings, the Tree Theorem, and Vásonyi's Conjecture, *Trans. American Math. Soc.* 95, 1960 pp. 210–225.
- [75] Kruskal, J.B., The Theory of Well Quasi-Orderings: a Frequently Rediscovered Concept, *Journal of Combinatorial Theory (A)* 13, 1972, pp. 297–305.
- [76] Küchlin, W., A Confluence Criterion based on the generalised Newman lemma, in Ed. B. Caviness, *Proc. Eurocal '85*, LNCS 204, Springer-Verlag, pp. 390–399.

REFERENCES

- [77] Le Chenadec, P., Canonical Forms in Finitely Presented Algebras, in Proc. CADE-7, LNCS 170, Springer-Verlag, 1984, pp. 142–165.
- [78] Ledermann, W., Introduction to Group Theory, Longman 1973.
- [79] Lescanne, P., Well Quasi Orderings in a paper by M. Janet, Bulletin of EATCS, No. 36, 1989, pp. 185–188
- [80] Lange, S., Towards a Set of Inference Rules for Solving Divergence in Knuth-Bendix Completion, in Ed. K.P. Jantke, Proc AII '89, LNAI 397, Springer-Verlag, 1989, pp. 304–316.
- [81] Levy, A., Irrelevance Reasoning in Knowledge Based Systems, Ph.D. thesis, Stanford University, 1993.
- [82] Lingensfelder, C., Structuring Computer Generated Proofs, Proc. IJCAI '89, Detroit, 1989, pp. 378–383.
- [83] Linton, S., and Shand, D., Some Group Theoretic Examples with Completion Theorem Provers, in Ed. J. Slaney, ARIA workshop, CADE-12, Nancy, 1994.
- [84] Linton, S., and Shand, D., Some Group Theoretic Examples with Completion Theorem Provers, to appear in Journal of Automated Reasoning.
- [85] Linton, S., Martin, U., Prohle, P., and Shand, D., Algebra and Automated Deduction, in Ed. M. McRobbie and J. Slaney, Proc. of CADE-13, LNAI 1104, Springer-Verlag, 1996, pp.448–462.
- [86] Madlener, K. and Otto, F., Pseudo-natural algorithms for the word problem for finitely presented monoids and groups, Journal of Symbolic Computation 1, 1985, pp. 383–418.
- [87] Madlener, K. and Otto, F., Pseudo-natural Algorithms for Finitely Generated Presentations of Monoids and Groups, Journal of Symbolic Computation 5, 1988, pp. 339–358.
- [88] Madlener, K. *et al*, On the Problem of Generating Small Convergent Systems, Journal of Symbolic Computation 16, 1993, pp. 167–188.
- [89] Martin, U. and Lai, M., Some Experiments with a Completion Theorem Prover, Journal of Symbolic Computation, 13, 1992, pp. 81–100.

REFERENCES

- [90] Martin, U. and Nipkow, T., Ordered Rewriting and Confluence, in *Automated Deduction – CADE-10*, LNAI 449, Springer-Verlag 1990, pp. 366–380.
- [91] Martin, U., Theorem Proving with Group Presentations: Examples and Questions, in Ed. M. McRobbie and J. Slaney, *Proc. of CADE-13*, LNAI 1104, Springer-Verlag, 1996 pp. 358–372.
- [92] Mayr, E., and Meyer, A., The Complexity of the Word Problem for Commutative Semigroups and Polynomial Ideals, *Adv. Math.* 46, 1982, pp. 305–329
- [93] McCune, W.M., *Otter 3.0 Reference Manual and Guide*, Argonne National Laboratory, report ANL-94/6, 1994.
- [94] McCune, W.M., Single Axioms for Groups and Abelian Groups with Various Operations, *J. Automated Reasoning* 10(1), 1993, pp. 1–13
- [95] McCune, W.M., and Padmanabhan, R., *Automated Deduction in Equational Logic and Cubic Curves*, LNAI 1095 Springer-Verlag, 1996.
- [96] Meinke, K. and Tucker, J.V., Universal Algebra, in Eds. S. Abramsky, D.V. Gabbay, and T.S.E. Maibaum, *Handbook of Logic in Computer Science*, vol. 1, Clarendon, Oxford, 1992, pp. 189–398.
- [97] Murthy, C., *Extracting Classical Content from Classical Proofs*, Ph.D. dissertation, Cornell University, 1990.
- [98] Needham, R.E., *Infinite complete group presentations*, Department of Mathematics of The City College of The City University of New York, preprint.
- [99] Nipkow, T., More Church-Rosser Proofs (in Isabelle / HOL), in Ed. M. McRobbie and J. Slaney, *Proc. of CADE-13*, LNAI 1104, Springer-Verlag, 1996 pp. 358–372.
- [100] Paulson, L., *Isabelle – a Generic Theorem Prover*, LNCS 828, Springer-Verlag, 1994.
- [101] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, 1984.
- [102] Peterson, G. E. and Stickel, M. E., Complete Sets of Reductions for Some Equational Theories, *Journal of the ACM* 28, 1981, 233–264.

REFERENCES

- [103] Pfenning, F., Analytic and Non-Analytic Proofs, in Ed. R.E. Shostak, CADE-7, LNCS 170, Springer-Verlag, 1984, pp.394–413.
- [104] Pierce, W., Toward Mechanical Methods for Streamlining Proofs, in Ed. M. Stickel, Proc. CADE-10, LNCS 449, Springer-Verlag, 1990, pp.351–365.
- [105] Robertson, N., and Seymour, P., Graph Minors – a Survey, in Ed. I. Anderson, Surveys in Combinatorics, CUP, 1985, pp. 153–171.
- [106] Robinson, D.J.S., A course in the Theory of Groups, GTM 80, 1982, pp. 119–120.
- [107] Robinson, J.A, A Machine Oriented Logic Based on the Resolution Principle, JACM 12(1), 1965, pp. 23–41.
- [108] Sattler-Klein, A., Divergence Phenomena during Completion, in Ed. R. Book Proc. RTA, LNCS 488, Springer-Verlag, 1988, pp. 374–385.
- [109] Schönert, M., et. al.: GAP – Groups, Algorithms, and Programming, Lehrstuhl D für Mathematik, RWTH Aachen, Germany, 1992. GAP can be obtained by anonymous ftp in /pub/gap/ at ftp.math.rwth-aachen.de.
- [110] Shand, D. and Brock, S.H., Proofs as Graphs, Proc. SEGRAGRA '95, Electronic Notes in Theoretical Computer Science, Elsevier, 1995.
- [111] Shankar, N., Proof Checking Metamathematics, Ph.D. Dissertation, University of Texas at Austin, 1986.
- [112] Sims, C.C., Computation with Finitely Presented Groups, CUP, 1994.
- [113] Sims, C.C., Verifying Nilpotence, Journal of Symbolic Computation 3, 1987, pp. 231–247.
- [114] Sims, C.C., The Knuth-Bendix Procedure for Strings as a Substitute for Coset Enumeration, Journal of Symbolic Computation 12, 1991, pp. 439–442.
- [115] Squier, C.C., Word Problems and a Homological Finiteness Condition for Monoids, J. Pure Appl. Algebra, 49, 1987, pp. 201–217.
- [116] Squier, C.C., A Finiteness Condition for Rewriting Systems, preprint.

REFERENCES

- [117] Sutcliffe, G., Suttner, C. and Yememis T., The TPTP Problem Library, in Ed. A. Bundy Automated Deduction — CADE-12, LNAI 814, Springer-Verlag, 1994, pp. 252–266.
- [118] Sutherland, W.A., Introduction to Metric and Topological Spaces, Clarendon Press, Oxford, 1989.
- [119] Thomas, R.M., The Fibonacci Groups Revisited, in Eds. C.M. Campbell and E.F. Robertson, Groups St. Andrews 1989, Vol. 2, London Mathematical Society Lecture Note Series, Vol. 160 CUP, pp. 445–454.
- [120] Thomas, M., and Jantke, K.P., Inductive Inference for Solving Divergence in Knuth-Bendix Completion, in Ed. K.P. Jantke, Proc. AII '89, LNAI 397, Springer-Verlag, 1989, pp.288–303.
- [121] Thomas, M. and Watson, P., Solving Divergence in Knuth-Bendix Completion by Enriching Signatures, Theoret. Comput. Sci 112, 1993, pp. 145–185.
- [122] Tietze, H., Über die topologischen Invarianten mehrdimensionaler Mannigfaltigkeiten, Monatsh. für Math. und Phys., 19, 1908, pp. 1–118.
- [123] Todd, J.A. and Coxeter, H.S.M., A Practical Method for Enumerating the Cosets of a Finite Abstract Group, Proc. Edinburgh Math. Soc. 5, 1936, pp. 26–34.
- [124] Vaughan-Lee, M., The Restricted Burnside Problem, Oxford, 1990.
- [125] Winkler, F., Reducing the Complexity of the Knuth-Bendix Completion Algorithm: a 'unification' of different approaches, in Ed. B. Caviness, Proc. Eurocal '85, LNCS 204, Springer-Verlag, pp. 378–389.
- [126] Wolfram, S., Mathematica: a System for doing Mathematics by Computer, Addison-Wesley, Reading, 1991.
- [127] Zhang, H. and Kapur, D., Unnecessary Inferences in Associative-Commutative Completion Procedures, Math. Systems Theory, 23, 1990, pp. 175-206.

A Example Inputs for LP and Otter

Consider the presentation

$$\langle a, b \mid a^2 = b^2 = (ab)^3 = 1 \rangle.$$

We will see how it would appear in the two different formulations (one with the generators as constant symbols, the other with generators as unary function symbols) as an Otter or LP input file.

```
% Otter input in first formulation
set(knuth_bendix).                set(print_lists_at_end).
lex([e,x*y,i(x),a,b]).

list(sos).
(x = x).                          ((x* (y*z)) = ((x*y)*z)).
(x*e = x).                        (x*i(x) = e).
(a*a = e).                        (b*b = e).
(a*b*a*b*a*b = e).
end_of_list.

%Otter input in second formulation
set(knuth_bendix).                set(print_lists_at_end).

lex([a(x),a1(x),b(x),b1(x)]).

list(sos).
(x = x).
(a(a1(x)) = x).                  (a1(a(x)) = x).
(b(b1(x)) = x).                  (b1(b(x)) = x).
(a(a(x)) = x).                  (b(b(x)) = x).
(a(b(a(b(a(b(x)))))) = x).
end_of_list.
```

A EXAMPLE INPUTS FOR LP AND OTTER

%LP input in first formulation

```
declare sort e1
declare operators
__ * __ : e1,e1 -> e1
i : e1->e1,
e,a,b : -> e1
..
declare variables x,y,z :e1
register height a > b > i > * > e

assert
x*(y*z) = (x*y)*z;      x*e = x;
x*i(x) = e;             a*a = e;
b*b = e;                a*b*a*b*a*b = e;
..
complete
```

%LP input in second formulation

```
declare sort e1
declare operators
a, b, a1, b1 : e1 -> e1
..
declare variables x:e1

assert
a(a1(x)) = x;           a1(a(x)) = x;
b(b1(x)) = x;           b1(b(x)) = x;
a(a(x)) = x;           b(b(x)) = x;
a(b(a(b(a(b(x)))))) = x;
..
complete
```

A EXAMPLE INPUTS FOR LP AND OTTER

The following is an Otter script prepared by McCune (see McCune [2]) to solve the $B(m, 3)$ problem.

```
% A difficult group theory commutator problem.
% If  $xxx = e$ , then  $[[[x,y],z],u] = e$ , where  $[uv] = u'v'uv$ .
%
% Specialized formulation and strategy.

set(knuth_bendix).

lex([e,A,B,C,D,E,*(_,_),g(_),h(_,_)]).

clear(lrpo).

clear(print_kept).
clear(print_new_demod).
clear(print_back_demod).
clear(detailed_history).
assign(report, 33600).

assign(pick_given_ratio, 2).
assign(max_weight, 105).
assign(max_mem, 14000).

assign(fpa_literals, 0). % to save memory (and a little time)
assign(fpa_terms, 0). % to save memory (and a little time)

list(usable).
x = x.
end_of_list.

list(sos).
x*x*x*y = y.
(x*y)*z = x*y*z.
h(h(h(A,B),C),D)*E != E.
end_of_list.
```

A EXAMPLE INPUTS FOR LP AND OTTER

```
list(demodulators).  
h(x,y) = g(x)*g(y)*x*y.  
g(x) = *(x,x).  
end_of_list.
```

```
weight_list(pick_and_purge).  
weight(x, 4). weight(A, 0). weight(B, 0).  
weight(C, 0). weight(D, 0). weight(E, 0).  
end_of_list.
```