USE OF THESESES

This copy is supplied for purposes of private study and research only. Passages from the thesis may not be copied or closely paraphrased without the written consent of the author.
Addenda

The following sections correct, clarify and add some further details to issues covered in the thesis. These comments have been added in response to queries and suggestions offered by anonymous referees.

1 Errata

p 40: The heading “elimination elimination” should read “elimination rules”. Also, the symbols $\Pi_I$ and $\Pi_E$ that occur in this figure stand for introduction proof components and elimination proof components respectively.

p 109: In the second paragraph the word “architecture” should be “architecture”.

2 Notes

Whenever the term *axiom* is used, it should be understood as shorthand for the term *non-logical axiom*. Since natural deduction systems do not employ any *logical axioms*, this shorthand is not ambiguous.

It has been pointed out that all the examples in the thesis are in the datalog (function symbol free) form. This is to make the examples simple to understand. The natural deduction framework has nothing to say about the structure of terms. The reader should assume that we are dealing with first order terms, as defined in the beginning of chapter 2, throughout.

The proof of lemma 1 in chapter 2 supplies just the reduction steps needed for normalization. A proof of termination of the normalization procedure can be found in [Prawitz 65].

With reference to the positive definite language, described in chapter 4: The *dependency constraint* associated with the $\forall I$ rule can be more efficiently implemented if we employ Skolem functions $f^S(X_1,\ldots,X_n)$ instead of just Skolem constants $X^S$. The parameters $X_1,\ldots,X_n$ being the set of free variables that occur in assumptions on
which the skolemized formula depends. N-Prolog [Gabbay & Reyle 84] and fragments of λ-Prolog [Miller et al. 89] offer well advanced implementations of this language.

The notion of atomic normal form proof is a rediscovery of expanded normal form proof published in [Prawitz 71]. That paper contains also an interesting discussion of the strengths of natural deduction proof theories compared to other approaches.

3 Sequent Calculus

An outline of the relationship between sequent calculus [Gentzen 35] and the natural deduction systems introduced in the thesis is interesting. Note that sequent calculus was historically derived from natural deduction.

In sequent calculus inferences no longer carry us from a collection of premiss formulae to a conclusion formula, but from a collection of premiss sequents to a conclusion sequent. A sequent being an object having the structure

\[ F_1, F_2, \ldots, F_m \Rightarrow G_1, G_2, \ldots, G_n \]

where the Fs andGs are formulae. The Fs here correspond to the assertion formulae and the Gs to the goal formulae of our natural deduction framework.

For classical predicate calculus the above sequent is informally equivalent to the formula

\[ (F_1 \land F_2 \land \ldots \land F_m) \supset (G_1 \lor G_2 \lor \ldots \lor G_n) \]

Formally, the structure of the sequent is given meaning by the so called structural rules of inference. The \texttt{CUT} rule of inference introduced in the thesis is a natural deduction analog of the structural rule known as \textit{cut} (schnitt). In sequent calculus we could write the \texttt{CUT} rule of the system \texttt{C~} of figure 3.13 in this way

\[
\frac{\Delta \Rightarrow B \quad A, \Delta \Rightarrow C}{(\Delta \Rightarrow C)\Theta} \quad \text{where: } A\Theta = B\Theta
\]

This analogy cannot however be taken too far. The thesis puts forward the idea that one should think of an asserted formula as a license to perform certain inferences. For example, a formula \(A \land B \supset C\) has the force of the inference rule

\[
\frac{\Delta \Rightarrow A \quad \Delta \Rightarrow B}{\Delta \Rightarrow C} \quad \text{where: } (A \land B \supset C) \in \Delta
\]

The function of the \texttt{CUT} rule is to build proofs using these derived rules or proof components. Still, both the sequent calculus cut and \texttt{CUT} mean transitivity of derivability, and hence the name.
ADDENDA

4 Implementation

Comparisons were made between a Prawitz normal form and an atomic normal form minimal logic implementation for propositional examples (kindly supplied by Neil Tennant). These experiments were inconclusive about the comparative efficiency of atomic and non-atomic systems. For the example set given, differences in performance were small and favoured one system or the other depending on the problem.

Implementations, in the form of Prolog meta-interpreters, were developed for first order systems up to minimal logic (according to the classification of chapter 4). Work is continuing on more serious implementations, refer [Keranen 93].

References

[Gabbay & Reyle 84]

[Gentzen 35]

[Keranen 93]

[Miller et al. 89]

[Prawitz 65]

[Prawitz 71]
Computational Natural Deduction

Seppo R. Keronen

A thesis submitted for
the degree of Doctor of Philosophy
of the Australian National University

© Seppo R. Keronen
November 1991
I hereby state that this thesis contains only my own original work, except where explicit reference is made to the work of others.

Seppo Keronen
Dedicated to
my mother
Leila Keronen
and my father
Reino Keronen
Abstract

The formalization of the notion of a logically sound argument as a natural deduction proof offers the prospect of a computer program capable of constructing such arguments for conclusions of interest. We present a constructive definition for a new subclass of natural deduction proofs, called atomic normal form (ANF) proofs. A natural deduction proof is readily understood as an argument leading from a set of premisses, by way of simple principles of reasoning, to the conclusion of interest. ANF extends this explanatory power of natural deduction. The very detailed steps of the argument are replaced by derived rules of inference, each of which is justified by a particular input formula. ANF constitutes a proof theoretically well motivated normal form for natural deduction. Computational techniques developed for resolution refutation based systems are directly applicable to the task of constructing ANF proofs.

We analyse a range of languages in this framework, extending from the simple Horn language to the full classical calculus. This analysis is applied to provide a natural deduction based account for existing logic programming languages, and to extend current logic programming implementation techniques towards more expressive languages. We consider the visualization of proofs, failure demonstrations, search spaces and the proof search process. Such visualization can be used for the purposes of explanation and to gain an understanding of the proof search process. We propose an introspection based architecture for problem solvers based on natural deduction. The architecture offers a logic based meta language to overcome the combinatorial and other practical problems faced by the problem solver.

Keywords: computational logic, natural deduction, logic programming.
# Contents

**Preface** 8

1 **Introduction** 10
   1.1 Natural Deduction 11
   1.2 A Meta Deduction Problem 14
   1.3 Atomic Normal Form Deduction 15
   1.4 Computation 18
   1.5 Logic Programming 22
   1.6 Extended Languages 23
   1.7 Control by Introspection 24
   1.8 Multiple Context Evaluation 27

2 **Natural Deduction** 29
   2.1 Logical Preliminaries 29
   2.2 Proof 31
   2.3 Rules of Inference 33
   2.4 Normal Form 37
   2.5 Cut Normal Form 39
   2.6 Atomic Normal Form 42

3 **Deduction Problems** 45
   3.1 Computational Preliminaries 45
   3.2 Search Spaces and Strategies 47
      3.2.1 Forward Chaining Search 47
      3.2.2 Backward Chaining Search 49
      3.2.3 ANF Search 50
   3.3 Atomic Normal Form Solutions 51
      3.3.1 The Deduction System $C_\Sigma(\Phi)$ 52
      3.3.2 Goals and Assertions 54
      3.3.3 Search Graphs and Solution Graphs 56
CONTENTS

3.4 Extend and Compose ........................................ 57
  3.4.1 Extend ................................................. 57
  3.4.2 Compose ............................................... 60

4 Expressive Power and Inference .......................... 64
  4.1 Containment and Conservative Extension .............. 64
  4.2 The Horn Language ................................... 65
    4.2.1 Horn Formulae and Their Extensions ............. 66
    4.2.2 CUT and The Quantifier Rules $\forall$ and $\exists$ .... 67
  4.3 The Positive Edinburgh Prolog Language ............ 68
    4.3.1 The Or Introduction Rules $\lor$ ............. 69
  4.4 The Positive Definite Language ...................... 70
    4.4.1 The Quantifier Introduction Rule $\forall$ .... 70
    4.4.2 The Implication Introduction Rule $\supset$ .... 71
  4.5 The Positive Language ................................ 73
    4.5.1 The And Elimination Rule $\land$ ............. 73
    4.5.2 The Existential Elimination Rule $\exists$ .... 74
    4.5.3 The Or Elimination Rule $\lor$ ............... 75
  4.6 Minimal Logic ........................................ 78
    4.6.1 The Negation Rules $\lnot$ and $\lnot$ .......... 78
  4.7 Intuitionistic Logic .................................. 79
  4.8 Classical Logic ...................................... 80
  4.9 Alternative Languages ................................. 82

5 Inference Engines ............................................ 84
  5.1 AND/OR Graphs and Derived Rules of Inference ...... 84
  5.2 Search as a Constraint Satisfaction Problem .......... 86
    5.2.1 Axiom and Query Relevance .................... 86
    5.2.2 Resolved Choice ................................ 87
    5.2.3 Substitution Consistency ...................... 88
    5.2.4 Loop Freeness ................................ 89
  5.3 Implementation Technology ............................. 90
    5.3.1 The Prolog Inference Engine ................. 90
  5.4 Extended Logic Programming ........................... 93
  5.5 Implementation Techniques for Extended Languages ... 94
    5.5.1 More Expressive Sequential Languages .......... 94
    5.5.2 Single Solution AND Parallelism ............. 96
    5.5.3 Multiple Solutions AND/OR Parallelism ........ 97
CONTENTS

6 Exploiting the Representation 100
   6.1 Visualization ........................................... 100
      6.1.1 Flat Explanation .................................. 101
      6.1.2 Structured Explanation ............................ 102
      6.1.3 Testing and Debugging ............................. 105
   6.2 Introspection ........................................... 108
      6.2.1 An Extended Introspective Architecture .......... 109
      6.2.2 Ordering the Search ............................... 111
      6.2.3 Detecting Loops .................................... 113
      6.2.4 Pruning the Search Space .......................... 113
      6.2.5 Negation As Failure ................................ 114
      6.2.6 Allocating Computational Resources ............. 115
      6.2.7 Exploiting Models .................................. 116
      6.2.8 Specifying Communication .......................... 117

7 Conclusion 118
   7.1 Proof Theory ........................................... 118
   7.2 Languages ............................................... 119
   7.3 Computation ............................................. 120
   7.4 Visualization .......................................... 121
   7.5 Introspection .......................................... 121
Preface

A central challenge for the discipline of deductive logic is to precisely characterize what is to count as an acceptable or sound argument in any domain of discourse whatsoever. In summary, an argument consists of a collection of statements (premisses) from which another statement (conclusion) is claimed to follow. Originally conceived to formalize reasoning in the domain of mathematics [Whitehead & Russell 1910-1913], classical first order predicate calculus provides, to date, the most widely accepted approach to meet this challenge.

A deduction system provides a formal, inductive definition of an argument. We will refer to this formal counterpart of an argument as a proof. Such a system offers the exciting prospect of a computer program which, given the description of a particular problem domain as premisses, is capable of constructing sound arguments for conclusions of interest. After early implementations revealed the computational intractability of this task, two important advances were made:

- The work of Alan Robinson [Robinson 65] on resolution refutation demonstrated a dramatic improvement in efficiency. The search space generated by the resolution rule of inference is smaller than for many other deduction systems. The fundamental unit of computation for the inference engine is the unification of two atomic formulae. The resolution refutation regime is readily applied in conjunction with goal directed and heuristic search strategies.

- Patrick Hayes [Hayes 73] and Robert Kowalski [Kowalski 79a] pioneered the idea that the search for solutions to deduction problems constitutes a worthy problem domain in itself. That is, a deduction problem consists of two distinct components, one contains knowledge describing the problem domain, the other contains knowledge to direct the search process.

The power of the above two ideas is captured beautifully in the logic programming language Prolog [Clocksin & Mellish 81]. We adopt the Prolog language, and its impressive implementation technology, as a point of reference for our investigation. We
propose a new approach to controlled deduction, still relying on essentially these same foundations, but incorporating the following two ideas:

- The natural deduction systems of Gerhard Gentzen [Gentzen 35] and Dag Prawitz [Prawitz 65] aim to reflect in a formal system rigorous arguments constructed by humans. Due to this similarity between natural deduction proofs and informal arguments, natural deduction is the standard proof theory for interactive proof editors, such as LCF [Paulson 87]. We argue that on these same grounds natural deduction is the right proof theoretic framework when explanation, debugging and control of deduction is required.

- Pick up any textbook on proof theory and you will notice that most of the discussion is at the meta level. The vocabulary is about formulae and proofs. Statements are schematic, meta-variables of various sorts standing for classes of these objects. We wish to offer this expressive power to the user of computer based problem solving systems. In particular, the search knowledge component of a deduction problem is best expressed as assertions in such a meta language. This idea has its roots in the artificial intelligence community [Hayes 73], [Davis 76] and [Weybrauch 80].

While much of our investigation focuses on the application of these ideas to logic programming, it is also relevant for other applications. Computational logic is an important, unifying theme for research in artificial intelligence [Genesereth & Nilsson 87] and deductive databases [Minker 88]. The need for more expressive, yet computationally tractable languages is acutely felt in these areas [McCarthy & Hayes 69].

In addition to the citations already made, the following have had a major influence on the current work: Nils Nilsson [Nilsson 80] on rule-based deduction systems, Jon Doyle [Doyle 79] and Johan de Kleer [de Kleer 86] on truth maintenance, Seif Haridi [Haridi 81] on natural deduction based logic programming, Peter Schroeder-Heister [Schroeder-Heister 84] on extensions of natural deduction and Neil Tennant [Tennant 87] on relevant deduction.

I wish to thank Professor Robin Stanton for the opportunity to carry out the research reported here. His encouragement and sharp insights have been invaluable. Andy Bollen, Rob Edmondson, Kerry Taylor, Neil Tennant and Graham Williams also contributed a great deal. I am pleased to acknowledge financial support by the Australian National University and the Fujitsu Corporation. Thank you to my wife Salad and our son Sol for your patience and love.
Chapter 1

Introduction

We introduce a new, deductively complete subclass of natural deduction proofs called atomic normal form (ANF) proofs. Natural deduction offers several outstanding advantages as a base for mechanical reasoning:

- Proofs are readily understood as formal counterparts of informal (but rigorous) arguments constructed by humans. Powerful explanation, debugging and control facilities can be provided, based on simply inspecting the state of a proof construction computation.

- Strictly classical reasoning is inappropriate for many deduction problems faced by mechanical reasoners. Alternative logics are often formulated as natural deduction systems. For example, deduction systems for intuitionistic logic, and logics that can supply coherent answers in the presence of contradictory knowledge, are available.

Compared to resolution refutation, little is known about the efficient implementation of natural deduction based systems. The ANF scheme is designed to address this shortcoming. The special advantages of ANF are:

- The implementation technology of resolution refutation theorem provers is directly applicable to the task of constructing ANF proofs. The fundamental operation of the inference engine is the unification of two atomic formulae.

- The extended syntactic forms of the Prolog family of logic programming languages distance them from SLD resolution. ANF provides a direct proof theoretic account for these languages. The procedural semantics and implementation techniques of current languages are faithfully modeled in the new framework. Efficient implementations for more expressive languages are suggested.
CHAPTER 1. INTRODUCTION

- Each proof step is the application of a derived rule of inference justified by a particular input formula\(^1\). Reasoning in terms of derived rules results in increased efficiency. The correspondence between derived rules and input formulae facilitates explanation, debugging and control of inference engine behaviour.

The following sections present an account of some of the main themes of our work in short form. The preview here is intended to provide a supply of motivations, intuitions and examples to be drawn on by the more formal and detailed investigations of chapters 2 – 6.

The deduction systems to be described are intended for the language of first order predicate calculus formulae and its sublanguages. Computational procedures are expressed in the logic programming language Prolog. Prolog will also appear as an object of study here. The pure logic fragment of Prolog is one of the sublanguages of predicate calculus to be studied. In discussing language extensions, control constructs and implementation techniques Prolog is adopted as a convenient reference point. A basic familiarity with these languages is assumed.

1.1 Natural Deduction

We follow the usual practice for natural deduction formulations and enrich the language of (well formed) formulae slightly. The symbol \(\#\) (read contradiction) is admitted as a well formed atomic formula. Also we distinguish syntactically between occurrences of bound variables (denoted by \(\ldots, x, y, z\)) and free variables or parameters (denoted by \(\ldots, X, Y, Z\)).

Natural deduction rules of inference can be recognized as formal counterparts of methods used in common reasoning practice. A selection of these rules, sufficient for the purposes of this chapter, appears in figure 1.1. Putting the rules of the figure into words:

**And introduction** (\(\land I\)): The conjunction \(G \land H\) follows given both a proof \(\Pi_1\) for the formula \(G\) and a proof \(\Pi_2\) for the formula \(H\).

**And elimination** (\(\land E\)): Given a proof \(\Pi\) of the conjunction \(E \land F\) the formula \(E\) follows. Also, from the same premiss the formula \(F\) follows.

**Implication introduction** (\(\implies I\)): The implication \(F \implies G\) follows given a proof \(\Pi\) of the consequent \(G\). The antecedent \(F\) may be assumed as an axiom for the purposes of the proof.

\(^1\)This principle is strictly observed for minimal logic and its subsystems only.
CHAPTER 1. INTRODUCTION

introduction rules

\[
\begin{array}{c}
\Pi_1, \Pi_2 \\
\frac{G \quad H}{G \land H}
\end{array}
\]

elimination rules

\[
\begin{array}{c}
\Pi \\
\frac{E \land F}{E}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{E \land F}{F}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{[F]}{G}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{G}{\neg \exists G}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{G \lor H}{G}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{H}{G \lor H}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{\forall x F(x)}{F(t)}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{\neg \exists F}{\neg \forall x F(x)}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{\neg G}{\neg \Pi}
\end{array}
\]

\[
\begin{array}{c}
\Pi \\
\frac{F}{\neg \Pi}
\end{array}
\]

Figure 1.1: a selection of natural deduction rules

**Implication elimination** (\(\Rightarrow\)): Given a proof of an implication and a proof of its antecedent the consequent follows. This rule is also called *modus ponens*.

**Or introduction** (\(\lor\)): A disjunction follows given a proof of either disjunct.

**Universal quantifier elimination** (\(\forall\)): Given a proof of a formula \(\forall x F(x)\) the formula \(F(t)\) follows for any term \(t\).

**Negation introduction** (\(\neg\)): Given a proof of \(#\) (contradiction), using the formula \(G\) as an assumption, the formula \(\neg G\) follows. The name *reductio ad absurdum* is also used for this rule.

**Negation elimination** (\(\neg\)): Given the two proofs, \(\Pi\) of \(\neg F\) and \(\Pi_1\) of \(F\), we have detected \(#\) (a contradiction).

Consider a subset of the above rules consisting of just the two negation rules shown in figure 1.2. These two rules determine a *deduction system* intended for reasoning with formulae constructed from propositional atoms and the negation operator only. Given a set of formulae (axioms), the two rules of inference may be used to derive a new formula (conclusion), that depends on a subset of the axioms (premisses). We refer to such a derivation as a *proof* of the conclusion from the premisses.
Let us now focus on a particular deduction problem: Given the set of axioms \( \{a, \sim b\} \) prove the conclusion \( \sim \sim \sim a \) using just the rules of inference shown in figure 1.2. In symbols:

\[
\{a, \sim b\} \vdash_{1.2} \sim \sim \sim a
\]

In order to solve this problem, augment the deduction system with new constructs representing the axioms and query. Rules of inference with nothing standing above the inference stroke represent axioms — In this case figure 1.3 (a) and (b). A rule of inference with nothing standing below the inference stroke represents the query formula — In this case figure 1.3 (c).

\[
\begin{array}{c}
\text{AXIOM} \quad a \\
(\text{a})
\end{array}
\quad \begin{array}{c}
\text{AXIOM} \quad \sim b \\
(\text{b})
\end{array}
\quad \begin{array}{c}
\text{QUERY} \quad \sim \sim \sim a \\
(\text{c})
\end{array}
\]

The proof displayed in figure 1.4 is a solution For the above problem. It is a composition of instances of the available inference rules, including those representing the axioms and the query. In more detail, a proof is a bipartite (formula nodes and inference nodes) tree-form graph. The root formula node (drawn as the bottommost formula node) is the conclusion of the proof. The leaf formula nodes are either assumptions or premisses. Inference nodes and their associated edges are represented by horizontal inference strokes. The mapping from assumptions to the inference rule instances that discharge them (discharge function) is indicated by annotating inference strokes with numbers.

On its own the constraint that every step within a proof be an instance of a rule of inference is too weak to capture the notion that a proof should arrive at its conclusion without unnecessary detours. Consider the deduction problem:

\[
\{a, \sim b\} \vdash_{1.2} \sim b
\]

\[\text{Generalisation to directed acyclic graph will be considered in chapter 4 in connection with asserted disjunctions.}\]
CHAPTER 1. INTRODUCTION

As well as the very direct solution of 1.5 (a), the unnecessarily complicated solution of 1.5 (b) (just one representative drawn from an infinite class) is also admitted.

The normal form for natural deduction proofs [Prawitz 65] excludes a large class of such unnecessarily complex proofs by imposing a simple constraint on the application of inference rules. Taking this well known normal form as a starting point, we will impose additional constraints to arrive at proofs in atomic-normal-form (ANF).

1.2 A Meta Deduction Problem

The current section serves two purposes: Preliminary to our investigation of the control of deduction, it introduces the basic concepts of meta language and reflection. We also employ the example meta deduction problem of this section to illustrate the main ideas in the remainder of this chapter.

The deduction problem of figure 1.6 encodes at the meta level the disjunction of two object level deduction problems for the system of figure 1.2. That is, whether either the formula \( a \) or the formula \( \sim a \) is provable given the set of axioms \( \{ a, \sim \} \).

The formulae in figure 1.6 are expressions in a meta language, whereas the formulae of the deduction problem they talk about are expressions in an object language. Object language formulae are represented by terms in the meta language. For example, the object language formula \( \sim a \) is here represented by the meta language term \( \text{not}(\text{atom}(a)) \).
Apart from this reflection of object language items in the meta language, we will keep the two languages separate. In this case for instance, meta language expressions are formed using the operators $\land, \lor, \supset$ and $\forall$, while the object language admits the $\sim$ operator only.

The intended interpretation of the predicate $\text{goal}/1^3$ is provability of the argument formula. The predicate $\text{assertion}/1$ stands for a restricted notion of provability, in terms of which $\text{goal}/1$ is defined. The first axiom in the figure represents the two object level axioms $a$ and $\sim b$. The last two axioms in the figure represent the two rules of inference of the object deduction system. The notions of $\text{assertion}$ and $\text{goal}$ are explained in chapter 3. The reader may wish to return to this example once that material has been covered.

### 1.3 Atomic Normal Form Deduction

An outline of the ANF proof theory, by way of an example, follows. To contrast the two approaches, we run through the meta deduction problem of figure 1.6 in both the resolution refutation and ANF regimes.

\[
\begin{align*}
\text{assertion}(\text{atom}(a)) \\
\text{assertion}(\text{not}(\text{atom}(b))) \\
\forall x \text{ assertion}(\text{atom}(x)) \supset \text{goal}(\text{atom}(x)) \\
\forall y \text{ assertion}(\text{not}(y)) \land \text{goal}(y) \supset \text{assertion}(\text{atom}(\#)) \\
\forall z (\text{assertion}(z) \supset \text{goal}(\text{atom}(\#))) \supset \text{goal}(\text{not}(z)) \\
\end{align*}
\]

$\forall \text{goal}(\text{atom}(a)) \lor \text{goal}(\text{not}(\text{atom}(a)))$

Figure 1.6: meta deduction problem

The resolution refutation scheme requires the translation of the axioms and negated

---

3The notation $\text{name/arity}$ is used for predicates.
the breakdown of an axiom or query into its atomic components by the application of natural deduction rules of inference. The inference rules appearing in this example can be found in figure 1.1.

For resolution the problem of deducing the query from the axioms is re-represented as: Derive the null clause, using the resolution rule of inference, starting with the clausal translation of the axioms and negated query. In this case a number of derivations for the null clause are possible. Three of the simplest are shown in figure 1.9.

\[
\sim \text{goal}(\text{atom}(a)) \quad \sim \text{assertion}(\text{atom}(X)) \lor \text{goal}(\text{atom}(X))
\]

\[
\sim \text{assertion}(\text{atom}(a)) \\
\text{assertion}(\text{atom}(a)) \\
\]

(a)

\[
\text{assertion}(\text{atom}(a)) \quad \sim \text{assertion}(\text{atom}(X)) \lor \text{goal}(\text{atom}(X)) \\
\text{goal}(\text{atom}(a)) \\
\sim \text{goal}(\text{atom}(a)) \\
\]

(b)

\[
\text{assertion}(Z) \lor \text{goal}(\text{not}(Z)) \\
\sim \text{goal}(\text{not}(\text{atom}(a)))
\]

\[
\sim \text{goal}(\text{atom}(a)) \quad \sim \text{assertion}(\text{atom}(X)) \lor \text{goal}(\text{atom}(X)) \\
\sim \text{assertion}(\text{atom}(a)) \\
\text{assertion}(\text{atom}(a)) \\
\]

(c)

Figure 1.9: resolution refutations

The ANF representation of the deduction problem is more direct: Is it possible to construct a natural deduction proof for the query by “pasting” together substitution instances of proof components? The “glue” we use is the structural rule of inference called cut\(^5\) [Gentzen 35]. Only the one proof, displayed in 1.10 (a), is possible. The instances of the cut rule may be removed, and the associated equality assertions applied as substitutions, to reveal the cut free proof of figure 1.10 (b).

---

\(^5\)Cut is the rule of inference which allows a mathematician to use lemmas when constructing a proof.
Two important points in favour of the ANF proof theory are illustrated by this example:

- A natural deduction proof can be read directly as an argument for the query from the axioms. Each step in the proof is an instance of a rule of inference used in common reasoning practice.

- There are often more resolution proofs for a given deduction problem than necessary. The ANF scheme is better focused, without losing deductive completeness.

1.4 Computation

The search space for solutions to a given deduction problem is a bipartite graph, subgraphs of which are proofs. We extend the notation of the previous section by showing the edges associated with inference nodes — see figure 1.13. This addition enables us to indicate that a particular formula node is the conclusion or premise of more than one inference node.

Computation proceeds in two stages:

**Extend** the input formulae into proof components. We will refer to the set of proof components implicit in an axiom or query formula as the inferential extension of that formula.
CHAPTER 1. INTRODUCTION

(a) \[ \text{assertion}(\text{atom}(a)) \land \text{assertion}(\text{not}(\text{atom}(b))) \]

(b) \[ \forall x \ (\text{assertion}(\text{atom}(x)) \supset \text{goal}(\text{atom}(x))) \]

(c) \[ \forall y \ (\text{assertion}(\text{not}(y)) \land \text{goal}(y)) \supset \text{assertion}(\text{atom}(\#)) \]

(d) \[ \forall z \ (\text{assertion}(z) \supset \text{goal}(\text{atom}(\#))) \supset \text{goal}(\text{not}(z)) \]

(e) \[ \text{goal}(\text{atom}(a)) \lor \text{goal}(\text{not}(\text{atom}(a))) \]

Figure 1.11: inferential extensions
Compose instances of proof components, by applying the cut rule of inference.

The required instances of cut are computed by unifying the atomic premiss of one component instance with the atomic conclusion of another.

The inferential extension of an input formula is computed by an algorithm which breaks down the formula, step by step, into its atomic components. Two classes of formulae assertions and goals are recognised. A non-atomic assertion instantiates an elimination rule — A non-atomic goal instantiates an introduction rule. The inferential extensions of the axioms and query of figure 1.6 are shown in figure 1.11. In this figure, the input formulae are framed by a heavier outline than the other, derived formulae.

A proof component may be summarized as a derived rule of inference. Consequently, a solution may be viewed as a composition of parameter renaming instances of either proof components or derived rules. The derived rules for the inferential extensions of figure 1.11 are shown in figure 1.12.

\[
\begin{align*}
\text{assertion}(\text{atom}(a)) & \quad \text{assertion}(\text{not}(\text{atom}(b))) & \text{assertion}(Z) \\
\text{goal}(\text{atom}(X)) & \quad \text{assertion}(\text{not}(Y)) & \text{goal}(Y) & \text{goal}(\text{atom}(\#)) \\
\text{goal}(\text{atom}(X)) & \quad \text{assertion}(\text{atom}(\#)) & \text{goal}(\text{not}(Z)) \\
\text{goal}(\text{atom}(a)) & \quad \text{goal}(\text{not}(\text{atom}(a))) & \text{goal}(\text{atom}(a)) \lor \text{goal}(\text{not}(\text{atom}(a))) & \text{goal}(\text{atom}(a)) \lor \text{goal}(\text{not}(\text{atom}(a)))
\end{align*}
\]

Figure 1.12: derived rules of inference

The required instances of cut are computed by unifying the atomic conclusion of one proof component instance with the atomic premiss of another. For example, two steps of composition, given the derived rules in figure 1.12, are shown in figure 1.17. We denote the cut rule by a double inference stroke, and explicitly display the equality assertions (substitutions) required for successful unification.

A solution to a deduction problem is determined by the constraints:
A solution corresponds to a solution graph [Nilsson 80] of the AND/OR search space. A solution graph consists of:

- The query inference node.
- For every inference node all its premiss formula nodes.
- For every formula node exactly one inference node that has the formula node as conclusion.

The set of equality assertions, associated with the cut nodes of the solution graph, are to be consistent. Consistency is tested in the simple syntactic equality theory required for the well-formedness of natural deduction proofs. Closer to the implementation level, this corresponds to the well known composition of substitutions.
operation [van Vaalen 75].

Additional constraints will be imposed in chapter 5 for the proper discharge of assumptions and to ensure that the solution arrives at its conclusion without unnecessary detours.

For the example meta deduction problem, the search space generated by a breadth first backward chaining search strategy is illustrated in figure 1.13. The figure is drawn up in terms of derived rules. Only the leftmost branch of this AND/OR tree yields a solution, corresponding to the proof of figure 1.10. For all the other solution graphs the associated sets of equality assumptions are inconsistent.

Normally it is required that compose perform a complete search for solutions. The associated task of recording what portion of the search space remains unexplored at any given time is simplified by carrying out the search within the framework of a search strategy. A backward chaining strategy is one which only looks at partial solutions that include the query, as illustrated in figure 1.14. Each such partial solution corresponds to a conditional proof of the query — That is, assuming the set of open premisses the query follows. The search frontier (indicated by the dotted line in the figure) consists of the set of open atomic premisses of conditional proofs.

![Figure 1.14: backward chaining search](image)

### 1.5 Logic Programming

Let us examine the relationship between the pure Prolog language and ANF proof theory. The SLD resolution proof theory for Horn clause languages [Apt & vanEmden 82] is well known. The syntactically richer Prolog formulae need first to be translated into sets of Horn clauses for the SLD story to apply [Lloyd & Topor 84]. Most Prolog implementations, however, do not rewrite formulae in this way. A subset of ANF justifies the inferences performed by these inference engines.

The inferential extension of a Horn-clause consists of a single derived rule of inference as illustrated in figure 1.15. The prime notation here indicates that applications of the universal quantifier elimination rule have replaced the bound variables of the clause by parameters. The deduction system for Horn-clauses consists only of the elimination
rules for implication and the universal quantifier and introduction rules for conjunction and the existential quantifier. Prolog implementors have, however, recognized the relative simplicity of the deductive machinery required for a richer goal syntax. To include a particular logical connective in the goal syntax, just the introduction rule for that connective is required.

\[
\forall A_1 \land A_2 \land \ldots \land A_n \supset B \quad \text{extend} \quad \frac{A'_1 \ A'_2 \ \ldots \ A'_n}{B'}
\]

Figure 1.15: inferential extension of a Horn clause

The AND/OR tree notation with explicit substitutions, introduced in the preceding section, reflects the data structures found on the stacks of Prolog machines — see [Bruynooghe 82], [Hogger 84]. The Prolog equality predicate (=/2) is simply the syntactic equality theory required for well-formedness of proofs. The negation as failure (NAF) rule of inference fits neatly, as a negation introduction rule, into the ANF framework. Failure proofs, like success proofs, can be characterized by a deduction system.

1.6 Extended Languages

The subformula property of ANF proofs establishes a simple correspondence between the syntax of input formulae and the deductive machinery required for implementation. To include a particular connective in the goal syntax — augment the inference engine with just the introduction rule(s) for that connective. Similarly for assertions — just the corresponding elimination rule(s) are required.

Once all the usual operators (∧, ∨, ⊃, ∀, ∃ and ¬) have been admitted in this way for both assertions and goals, we have reached minimal logic. Any omissions and we have minimal logic with syntactic restrictions. Intuitionistic logic is reached by adding the ex falso quodlibet rule of inference to the minimal logic machinery. In the presence of this rule, no query can fail until the theory in question has been demonstrated contradiction free. Add excluded middle to the intuitionistic system and we have full classical logic.

Intuitionistic, rather than classical, deducibility is appropriate when the domain of discourse, as in the example problem of figure 1.6, is a deduction system. The problem of provability in such systems is in general undecidable [Gödel 31]. In the case of the example problem, the query \textit{goal}(F) ∨ \textit{¬goal}(F) would receive a positive answer even

\[^6\text{apart from the trail}\]
in the case that \( P \) is undecidable in the object system. Both classical and intuitionistic logic are inappropriate if the consistency of all the axioms in the problem statement is not guaranteed. In this situation relevance logics can deliver some answers.

Note that resolution refutation is tightly coupled to classical logic by requiring the rewriting of formulae using classical equivalences. Consequently neither minimal nor intuitionistic logic has a resolution refutation proof theory. ANF makes these logics accessible, while retaining the achievements of resolution refutation implementation techniques.

### 1.7 Control by Introspection

The search spaces for anything but very simple deduction problems are computationally intractable. [Hayes 73] and [Kowalski 79a] have advocated a separate control component to specify the order in which these search spaces are to be explored. This idea is realized in Prolog by the procedural reading and embedding of control annotations in formulae. We explore an alternative approach, still based on the control component being supplied by the user.

![Diagram](image)

Figure 1.16: a simple introspective architecture

A logic-based view of the control component of a deduction problem is as a separate control theory specifying the next action of the inference engine at any given computation state. That is, control decisions are made by the inference engine based on the introspection [Smith 86] of its own state. The major subcomponents required for a realization of this view are:

- The upward reflection of a part of the computation state as a logical theory accessible to introspection.
CHAPTER 1. INTRODUCTION

The downward reflection of a theory specifying inference engine actions as computational behaviour.

A control theory to be used to derive the specification of inference engine actions given a computation state.

We investigate a realization of these ideas based on the simple reflect and act model of [van Harmelen 88]. The chosen architecture is illustrated in figure 1.16 and discussed in more detail in the following paragraphs.

For our natural deduction based system, the current state of computation can be readily understood by the person who is to write control axioms. The first three
computation states (depth first traversal) for the example meta deduction problem are shown in figure 1.17. The conceptualization of computation state at a number of increasingly detailed levels is possible:

- a collection of subgoals (the search frontier),
- a collection of partial solutions,
- AND/OR search space.

The upward reflected computation state theory is generated by a collection of atomic axioms. These axioms are implemented as procedures with access to the internal data structures of the inference engine. The closed world assumption is appropriate for this theory. For example, the search frontier of 1.17 (b) is represented by the two axioms in figure 1.18. The two arguments of the \texttt{open/2} predicate represent the unique name of the formula node and a substitution instance of the formula occurring in that node respectively.

For the downward reflected action theory, a simple procedural model of the inference engine is desirable. Let us conceptualize the behaviour of the inference engine in terms of a reflect-and-act cycle, represented by the Prolog procedure \texttt{search/1} of figure 1.19. The single argument of \texttt{search/1} represents a computation state. Only the two procedures \texttt{final/1} (termination condition) and \texttt{select/2} (selection of subgoal on the search frontier) are evaluated at the meta level. The more complex notions of rule selection and backtracking are absent — All solutions are explored concurrently without backtracking.

\begin{verbatim}
search(State) :- final(State),
               display(State).

search(State) :- not final(State),
               select(State,Subgoal),
               next(State,Subgoal,NextState),
               search(NextState).
\end{verbatim}

Figure 1.19: procedural model of inference engine

The control theory specifying inference engine actions at a given computation state consists of a finite number of ground literals of the form \texttt{final}, \texttt{~final}, \texttt{select(Subgoal)} and \texttt{~select(Subgoal)}. The \texttt{final/0} and \texttt{select/1} predicates of the control theory are reflected down as the \texttt{final/1} and \texttt{select/2} procedures of the inference engine respectively. The closed world assumption applies to \texttt{final/0} but not \texttt{select/1}. 
CHAPTER 1. INTRODUCTION

In the case of our example meta deduction problem, we know that queries about the restricted provability relation represented by the assert/1 predicate are easily answered. This control knowledge is expressed by the control axioms in figure 1.20. Applied to the computation state of figure 1.17 (b) the result is state (c).

\[
\forall n \forall f \text{ open}(n, \text{assertion}(f)) \supset \text{select}(n).
\]

\[
\neg (\exists n \exists f \text{ open}(n, \text{assertion}(f))) \supset (\forall n \forall f \text{ open}(n, f) \supset \text{select}(n)).
\]

Figure 1.20: control assertions

The above is clearly not yet a practical control language. We need to simplify the syntax and introduce the notion of a default control theory. In the presence of a default theory: If answers are computed in acceptable time, problem-specific control assertions need not be supplied — In case of unacceptable performance, control assertions may be added incrementally.

1.8 Multiple Context Evaluation

The model presented in the preceding section promises a conceptually simple, yet expressive control language. A generalization of that architecture also supports unrestricted AND/OR parallel evaluation of deduction problems. Subgoals on the search frontier may be selected for composition concurrently. Two such subgoals may or may not occur as premisses of some common conditional proof, corresponding to AND and OR parallel evaluation respectively.

To achieve these benefits the procedure next/3 carries a heavy computational burden. Given a subgoal selected on advice from the control theory, all applicable proof components are composed with all conditional proofs having that subgoal as premiss. We consider an implementation based on ATMS\(^7\) [de Kleer 86] technology.

The computation state maintained by the inference engine is an AND/OR graph of formula nodes and inference nodes, with equality propositions associated with inference nodes representing instances of the cut rule (cut nodes). Every solution graph, for which the set of all associated equality propositions is consistent, corresponds to a conditional ANF proof. Each such conditional proof is represented by a label, being the set of cut nodes for that proof. Each time next/3 is called the AND/OR graph and set of labels are updated. The equality propositions generated by the unifier are tested for consistency in the environment of every label containing the selected goal. Any label containing an inconsistent subset of equality propositions (a nogood), is removed. Nodes that appear in no label are pruned.

\(^7\)ATMS = assumption based truth maintenance system
For the example search space of figure 1.13, there are five solution graphs represented by the five labels in figure 1.21. Cut nodes are numbered in order of their creation by a backward chaining breadth first search strategy. Each label consists of a set of these numbers. Only the first of the labels corresponds to a proof, the remainder containing nogoods as indicated.
Chapter 2

Natural Deduction

The task of this chapter is to introduce natural deduction systems, and to develop the notion of proof in atomic normal form (ANF proof) in this framework. A formal deduction system for ANF proofs is presented. The deductive completeness of that system is established.

2.1 Logical Preliminaries

The deduction systems to be described are intended for the language of first order predicate calculus formulae. The use of the special symbol # for contradiction, and the distinction of bound and free variables as two separate classes of syntactic objects, are the only unusual features of the following formulation of this language.

Syntactic Categories: A formula is built out of symbols drawn from the following classes:

- **logical constants** — just the following seven symbols:
  - connectives: $\land, \lor, \rightarrow$ and $\neg$
  - quantifiers: $\forall$ and $\exists$

- **formula constant**: #

- **predicate symbols** — for each $n$ ($n = 0, 1, 2, \ldots$), a denumerable supply of symbols of arity $n$. When we have no particular interpretation in mind, we will use the lowercase letters $p, q, r, s$ for predicate symbols.

- **variables** — a denumerable supply of bound (by a quantifier) variables. We will use lowercase letters towards the end of the Roman alphabet ($\ldots, x, y, z$) for variables.

- **parameters** — a denumerable supply of free variables. We will use the uppercase counterparts of variables ($\ldots, X, Y, Z$) to stand as parameters.
constant symbols — for each \( n \) \((n = 0, 1, 2, \ldots)\), a denumerable supply of symbols of arity \( n \). In the case that we have no particular interpretation in mind, we will use lowercase letters at the beginning of the Roman alphabet \((a, b, c, \ldots)\) as constant symbols.

auxiliary symbols — commas and parentheses used for grouping and to avoid ambiguity.

Objects in the problem domain of interest will be represented by syntactic elements called terms.

**Term:** The class of terms is determined by the inductive definition:

- A parameter is a term.
- Let \( f \) be an \( n \)-place constant symbol and let \( t_1, t_2, \ldots, t_n \) be terms then \( f(t_1, t_2, \ldots, t_n) \) is a term. In the case that \( n = 0 \) we write \( f \) rather than \( f() \).

Propositions about the problem domain are represented by formulae. Atomic formulae are basic. Compound formulae are built up by the recursive application of the logical operators (connectives and quantifiers) to atomic formulae.

**Atomic Formula:** The class of atomic formulae is determined by the two clauses:

- \# is an atomic formula.
- If \( p \) is an \( n \)-place predicate symbol and \( t_1, t_2, \ldots, t_n \) are terms then \( p(t_1, t_2, \ldots, t_n) \) is an atomic formula. In the case that \( n = 0 \), we usually just write \( p \) instead of \( p() \).

**Formula:** The class of (well formed) formulae is determined by the following clauses:

- An atomic formula is a formula.
- If \( F \) is a formula then so is \( \sim F \).
- If \( E \) and \( F \) are formulae then so are \( EF \), \( EVF \) and \( E \cup F \).
- If \( F \) is a formula containing one or more occurrences of a parameter \( X \) and \( F' \) is obtained from \( F \) by replacing all occurrences of \( X \) by a variable \( x \) then \( \forall xF' \) and \( \exists xF' \) are also formulae.

The above definition implies that all but atomic formulae may be decomposed into a collection of simpler subformulae. The notion of subformula we need incorporates this idea together with a slight extension for negative formulae.
CHAPTER 2. NATURAL DEDUCTION

Subformula: With reference to the above definition of formula, the subformula relation is determined by the following two clauses:

- $E$ is a subformula of $F$ if there is a construction of $F$ from $E$ together with some set of atomic formulae.
- If $F$ is of the form $\sim E$ then $\#$ is a subformula of $F$.

When talking about formulae, in the language of this thesis, the following conventions apply: The symbols $A$, $B$ and $C$ will stand for atomic formulae. $E$, $F$, $G$ and $H$ will stand for formulae more generally, with various syntactic constraints as required. A substitution (of terms for parameters) will be denoted by a set of equality assertions in solved form [Lassez et al. 88]. For the substitution instance of formula $F$ using substitution $\Theta$ we write $F\Theta$. For example:

$$p(X,Z) \{X=f(Y), Z=a\} \text{ evaluates to } p(f(Y), a)$$

When talking about objects containing formulae as constituents, the following conventions apply: The symbols $\Gamma$ and $\Delta$ will stand for sets of formulae. A deduction problem statement is denoted by $\Phi$. Proofs and solutions (graphs containing formula occurrences as nodes) will be denoted by $\Pi$ and $\Sigma$ respectively. The notion of substitution instance is extended, in the obvious way, to cover these composite objects also. For example, applying the substitution $\Theta$ uniformly to all formula occurrences in a proof $\Pi$ results in the proof $\Pi\Theta$.

2.2 Proof

The set of natural deduction rules characterizing first order classical logic ($C$) is displayed in figure 2.3. These rules constitute the deduction system $C_\Pi$, being an inductive definition for the notion of proof in classical logic.

Proof: The inductive definition of proof goes like this:

**base** An occurrence of a formula standing alone is a proof supporting the given formula as conclusion, and depending on just that same formula as premiss. Such a trivial argument, for formula $F$, is represented by the deducibility assertion:

$$\{F\} \vdash F$$

**step** The rules of inference provide the clauses of the inductive definition. Given proofs matching each of the premisses of the rule, a proof for
the instantiated conclusion may be constructed. The new proof may discharge some of the premisses of the component arguments as assumptions. Such a composite argument with conclusion $G$ and depending on the set of premisses $\{F_1, F_2, \ldots, F_n\}$ is summarized by the deducibility assertion:

$$\{F_1, F_2, \ldots, F_n\} \vdash G$$

In accordance with the above definition, proofs generated by single conclusion rules of inference are trees. The root of the tree is the conclusion, the leaves of the tree are either premisses or assumptions. A path is a sequence of formula occurrences, leading from a leaf, all the way down to the root. The branches of the tree are sequences of formula occurrences, having a leaf at the top and a minor premiss formula\(^1\) or the conclusion of the entire proof at the bottom. The branch terminating at the conclusion is also called the trunk. In any proof figure we draw, the trunk is the leftmost branch. For some simple examples of proofs see chapter 1.

![Figure 2.1: notation for proofs](image)

Figure 2.1 introduces some notation for proofs:

(a) Proof $\Pi$ has conclusion $G$.

(b) Formula $F$ occurs as a premiss in proof $\Pi$. The conclusion depends on the premiss.

(c) A proof $\Pi_1$ with conclusion $F$, has been grafted onto proof $\Pi_2$ at an occurrence of $F$ as premiss.

(d) Formula $F$ may occur as an assumption in proof $\Pi$. For each occurrence of $F$ as an assumption there is exactly one inference rule occurrence in $\Pi$ that discharges that assumption occurrence.

(e) The inference rule annotated with (i) discharges an occurrence of formula $E$ as assumption. That is, $F$ depends on the indicated occurrence of $E$ as premiss, whereas $G$ no longer does so.

---

\(^1\)See next section for definition of minor premiss formula.
2.3 Rules of Inference

We adopt a slightly more verbose notation for inference rules than is common. For example, instead of the conventional notation of figure 2.2 (a) for the implication elimination (modus ponens) rule, we write (b). The formula \( F \) here is the conclusion of the rule, \( G \supset F \) and \( G \) are premiss formulae, and \( \Pi_1, \Pi_2 \) are premiss proofs. The leftmost premiss is the major premiss, the remaining one the minor premiss. The rule is to be read as the inductive clause:

\[
\text{If } \frac{\Pi_1}{G \supset F} \text{ is a proof and } \frac{\Pi_2}{G} \text{ is a proof then } \frac{G}{F \supset G} \text{ is a proof.}
\]

The inverse of the implication elimination rule is the implication introduction rule shown in figure 2.2 (c). These rules are inverse in the sense that the introduction rule defines the conditions under which an implication may be derived as conclusion — The elimination rule unlocks the inferential resources of an implication that has already been proved.

For the systems discussed in this chapter inference rules involving assumptions are free to discharge any subset of the indicated formula occurrences as assumptions. In other words, it is not required that all, or even any, assumptions be discharged.

The rules of inference appearing in the natural deduction system for classical logic, system \( C_\Pi \) of figure 2.3, may be partitioned into four subsets:

**Introduction Rules:** The generic introduction rule, with premiss proofs \( \Pi_1 \ldots \Pi_n \) and conclusion formula \( G \), is expressed in the pattern:

\[
\frac{\Pi_1 \ldots \Pi_n}{G}
\]

The premiss formulae are subformulae of the conclusion.
### Chapter 2. Natural Deduction

#### Introduction rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{}{G}$</td>
<td>Introduction of $G$</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>$\frac{H}{G \land H}$</td>
<td>Introduction of $G \land H$</td>
</tr>
</tbody>
</table>

#### Elimination rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{E \land F}{\neg E}$</td>
<td>Elimination of $E \land F$</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{E \land F}{F}$</td>
<td>Elimination of $F$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{G \lor H}{G}$</td>
<td>Elimination of $G$</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{G \lor H}{H}$</td>
<td>Elimination of $H$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{G}{F \supset G}$</td>
<td>Elimination of $F \supset G$</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{G \supset F}{F}$</td>
<td>Elimination of $F$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{G(x)}{\forall x G(x)}$</td>
<td>Elimination of $\forall$</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{\forall x F(x)}{F(t)}$</td>
<td>Elimination of $F(t)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{G(t)}{\exists x G(x)}$</td>
<td>Elimination of $\exists$</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{\exists x F(x)}{G}$</td>
<td>Elimination of $G$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{G}{\neg G}$</td>
<td>Absurdity rule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{#}{\neg \neg G}$</td>
<td>Absurdity rule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{#}{\neg F}$</td>
<td>Absurdity rule</td>
</tr>
<tr>
<td>$\Pi_1, \Pi_2$</td>
<td>$\frac{\neg F}{#}$</td>
<td>Absurdity rule</td>
</tr>
</tbody>
</table>

#### Absurdity rule

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{#}{G}$</td>
<td>Absurdity rule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{#}{F \lor \neg F}$</td>
<td>Absurdity rule</td>
</tr>
</tbody>
</table>

#### Excluded middle

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi$</td>
<td>$\frac{#}{F \lor \neg F}$</td>
<td>Excluded middle</td>
</tr>
</tbody>
</table>

#### Variable and Parameter

- $x$ — variable
- $X$ — parameter
- $t$ — term
- $E, F, G, H$ — formula
- $\Pi, \Pi_1, \Pi_2$ — proof

---

Figure 2.3: System $C_{\Pi}$ — natural deduction proof for classical logic
Elimination Rules: The elimination rule with major premiss $\Pi_0$, minor premisses $\Pi_1 \ldots \Pi_n$ and conclusion $F$ is expressed in the pattern:

$$
\begin{array}{c}
\Pi_0 \\
E \\
\Pi_1 \ldots \Pi_n \\
\hline
F
\end{array}
$$

Except for the or elimination ($\lor E$) and existential elimination ($\exists E$) rules, all minor premiss formulae and the conclusion are subformulae of the major premiss formula. For the $\lor E$ and $\exists E$ rules, all assumptions are subformulae of the major premiss formula.

Together the introduction and elimination rules define a subsystem of classical logic called minimal logic ($\mathcal{M}$) [Johansson 36].

Absurdity Rule: This rule expresses the principle that given a proof of contradiction, any formula whatsoever follows. The addition of the absurdity rule to minimal logic yields intuitionistic logic ($\mathcal{I}$) [Dummett 77].

Excluded Middle: This rule expresses the principle that for any formula whatsoever either it or its negation holds. The addition of this principle to the intuitionistic system yields the system for classical logic.

Note that the universal introduction ($\forall I$) and existential elimination ($\exists E$) rules place restrictions on the occurrence of parameters in proofs. Discussion of these restrictions, and the role that parameters play in proofs more generally, is deferred until chapter 4.

Natural deduction proofs are constructed not only by application of the above rules of inference. It is common practice to "graft" one proof (lemma) that establishes a conclusion $F$, on top of a proof requiring $F$ as a premiss. This principle, called cut [Gentzen 35], is expressed by the inductive clause:

$$
\begin{array}{c}
\Pi_1 \\
F \\
\Pi_2 \\
\hline
F
\end{array}
$$

Given a proof $\Pi_1$ and a proof $\Pi_2$, then $\frac{\Pi_1}{F}$ to $\Pi_2$ is a proof.

We emphasize the presence of a cut in a proof by use of a double inference stroke. Cut is sound for the system $C_{\Pi}$ of natural deduction and primitive for the system of atomic normal form deduction to be presented shortly.
introduction rules

\[
\begin{align*}
\text{\textit{\Pi}_N}_1, \text{\textit{\Pi}_N}_2 &\quad \frac{G \quad H}{G \land H} \\
\text{\textit{\Pi}_N} &\quad \frac{G}{G \lor H} \\
\text{\textit{\Pi}_N} &\quad \frac{[F]}{G} \\
\text{\textit{\Pi}_N} &\quad \frac{G}{F \supset G} \\
\text{\textit{\Pi}_N} &\quad \frac{G(X)}{\forall x G(x)} \\
\text{\textit{\Pi}_N} &\quad \frac{G(t)}{\exists x G(x)} \\
\text{\textit{\Pi}_N} &\quad \frac{[G]}{\sim G} \\
\text{\textit{\Pi}_N} &\quad \frac{\#}{\#} \\
\text{\textit{\Pi}_N} &\quad \frac{\#}{G}
\end{align*}
\]

elimination rules

\[
\begin{align*}
\text{\textit{\Pi}_M} &\quad \frac{E \land F}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{E \land F}{E} \\
\text{\textit{\Pi}_M} &\quad \frac{[E]}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{[F]}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{E \lor F}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{E \lor F}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{G \supset F}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{G \supset F}{F} \\
\text{\textit{\Pi}_M} &\quad \frac{\forall x F(x)}{F(t)} \\
\text{\textit{\Pi}_M} &\quad \frac{\exists x F(x)}{G} \\
\text{\textit{\Pi}_M} &\quad \frac{\sim F}{F} \\
\text{\textit{\Pi}_M} &\quad \frac{\sim F}{\#} \\
\text{\textit{\Pi}_M} &\quad \frac{\sim F}{G}
\end{align*}
\]

absurdity rule

\[
\begin{align*}
\text{\textit{\Pi}_N} &\quad \frac{\#}{G}
\end{align*}
\]

excluded middle

\[
\begin{align*}
\text{\textit{\Pi}_N} &\quad \frac{\#}{F \lor \sim F}
\end{align*}
\]

\[
\begin{align*}
x &\quad \text{variable} \\
X &\quad \text{parameter} \\
t &\quad \text{term} \\
E, F, G, H &\quad \text{formula} \\
\text{\textit{\Pi}_M} &\quad \text{major premiss proof} \\
\text{\textit{\Pi}_N}, \text{\textit{\Pi}_N}_1, \text{\textit{\Pi}_N}_2 &\quad \text{normal form proof}
\end{align*}
\]

Figure 2.4: system $C_{\Pi N}$ — normal form proof for classical logic


2.4 Normal Form

Our notion of proof, so far, places no other constraint on proofs than that they be constructed from instances of rules of the given deduction system. In an attempt to ensure that a proof arrive at its conclusion without unnecessary detours, we now place further constraints on the overall form of a proof.

The inversion principle states that no deductive gain is to be had by using a formula, constructed by an introduction rule, as the major premiss for an elimination rule. Members of the subclass of proofs, that excludes such unproductive applications of introduction rules, are called normal form proofs [Prawitz 65].

Normal Form Constraint: A natural deduction proof is in normal form just in case no major premiss formula of an elimination rule is the conclusion of an introduction rule.

We can define a deduction system, call it $C_{IN}$, that incorporates the normal form constraint:

Normal Form Proof ($\Pi_N$): A major premiss proof ($\Pi_M$) is determined by the rules of inference on the right (elimination rules and excluded middle) of figure 2.4. A normal form proof ($\Pi_N$) is then determined by the rules of inference on the left (introduction rules and absurdity rule) of the figure, and a clause stating that a $\Pi_M$ is a $\Pi_N$.

The fact that we do not lose deductive power by confining our interest to normal form proofs is confirmed by the following well known result.

Lemma 1 $\Delta \vdash G$ \iff $\Delta \vdash G$

Proof:

$\Delta \vdash G$ $\iff$ $\Delta \vdash G$

Every $C_{IN}$ proof is a $C_I$ proof.

$\Delta \vdash G$ $\Rightarrow$ $\Delta \vdash G$

Every $C_I$ proof may be transformed into an $C_{IN}$ proof by removing all instances of introduced major premisses by exhaustive application of the reduction transformations below. Note that the absurdity rule is treated as an introduction rule here.
\( \Lambda \)-reduction

\[
\begin{array}{c}
\Pi_1 \\
F_1 \\
\hline
\Lambda F_1 \land F_2 \\
\Lambda e (F_1) \\
\hline
\Pi
\end{array}
\quad \rightarrow 
\begin{array}{c}
\Pi_1 \\
(F_i)
\end{array}
\]

\( \lor \)-reduction

\[
\begin{array}{c}
\Pi_2 \\
\Pi_1 \\
\Pi_2 \\
\hline
F_1 \lor F_2 \\
G \\
\hline
\lor e (G) \\
\Pi
\end{array}
\quad \rightarrow 
\begin{array}{c}
\Pi_2 \\
(F_i)
\end{array}
\]

\( \exists \)-reduction

\[
\begin{array}{c}
[F] \\
\Pi_2 \\
\hline
\exists x F(x) \\
(F(t)) \\
\hline
\exists e (G) \\
\Pi
\end{array}
\quad \rightarrow 
\begin{array}{c}
\Pi_2 \\
(F(t))
\end{array}
\]

\( \forall \)-reduction

\[
\begin{array}{c}
\Pi_2 \\
\hline
\forall x F(x) \\
(F(t)) \\
\hline
\forall e (G) \\
\Pi
\end{array}
\quad \rightarrow 
\begin{array}{c}
\Pi_2 \\
(F(t))
\end{array}
\]

\( \sim \)-reduction

\[
\begin{array}{c}
[F] \\
\Pi_2 \\
\hline
\sim F \\
F \\
\hline
\sim e (\#) \\
\Pi
\end{array}
\quad \rightarrow 
\begin{array}{c}
\Pi_2 \\
(\#)
\end{array}
\]
2.5 Cut Normal Form

In this section we present an alternative formulation of normal form proof. This new formulation emphasizes more of the structure and reflects an efficient method of construction for these proofs. A normal form proof has more structure than is explicit in the above statement of the normal form constraint. The constraint confers two important properties on any normal form proof $\Pi_N$:

**Subformula Property:** For the intuitionistic system, every formula occurrence in $\Pi_N$ is a subformula of one of the premisses of $\Pi_N$ or of the conclusion of $\Pi_N$. For the classical system, negations of premisses and of the conclusion may also occur.

**Minimal Formula Property:** Every branch of a $\Pi_N$ consists of two segments. Tracing a branch from the top premiss or assumption, a sequence of elimination rule occurrences is followed by a sequence of introduction rule occurrences. The two segments are separated by a minimal formula occurrence. The minimal formula is a subformula of both the formulae at the top and bottom of the branch.

The above properties, recognised by [Prawitz 65], form the basis for partitioning a proof into *introduction components* and *elimination components*. An introduction component represents the inferential contribution of a particular goal formula to the overall proof. The conclusion of a proof is an example of a goal formula. An elimination component represents the contribution of a particular assertion formula to the overall proof. Premisses and assumptions are assertion formulae.

A component consists of a single initial formula occurrence together with subformulae of that initial formula. Components are separated from one another by minimal formula occurrences. More precisely:

---

$\#$-reduction

\[
\begin{array}{c|c|c|c|c}
\Pi_0 & \Pi_1 & \cdots & \Pi_n & \Pi_n \\
\hline
\# & F_0 & F_1 & \cdots & F_n \\
\hline
\rightarrow & \hline
\end{array}
\]

\]

(G) II

(G) II

\[
\]

The notions of *goal* and *assertion* are made precise in the next chapter.
### Introduction Rules

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G \land H \quad \Pi_1$</td>
<td>$- \land E \quad F \quad E \land F \quad E \leftrightarrow F$</td>
</tr>
<tr>
<td>$- v_1 \quad (G \lor H) \quad \Pi_i$</td>
<td>$\Pi_E \quad [F] \quad [E] \quad \Pi_i \quad \Pi_2$</td>
</tr>
<tr>
<td>$G \quad (G \lor H) \quad \Pi_i$</td>
<td>$\Pi_E \quad \Pi_i \quad \Pi_1$</td>
</tr>
<tr>
<td>$- \exists ! \quad (F \supset G) \quad \Pi_i$</td>
<td>$\Pi_E \quad \Pi_i \quad \Pi_2$</td>
</tr>
<tr>
<td>$G(X) \quad (\forall x G(x)) \quad \Pi_i$</td>
<td>$\forall x F(x) \quad \Pi_1$</td>
</tr>
<tr>
<td>$- \exists ! \quad (\exists x G(x)) \quad \Pi_i$</td>
<td>$\Pi_E \quad \Pi_2$</td>
</tr>
<tr>
<td>$[G] \quad \Pi_i \quad \Pi_2$</td>
<td></td>
</tr>
<tr>
<td>$\not\exists ! \quad (\neg G) \quad \Pi_i$</td>
<td>$\Pi_E \quad \Pi_i \quad \Pi_2$</td>
</tr>
<tr>
<td>$\not\exists ! \quad (G) \quad \Pi_i$</td>
<td>$\Pi_E \quad \Pi_i \quad \Pi_2$</td>
</tr>
</tbody>
</table>

### Elimination Rules

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi_E \quad G \quad \Pi_1$</td>
<td>$\Pi_E \quad A \quad \Pi_1$</td>
</tr>
<tr>
<td>$\Pi_E \quad G \quad \Pi_1$</td>
<td>$\Pi_E \quad A \quad \Pi_1$</td>
</tr>
</tbody>
</table>

### Cut Normal Form Proof

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ - variable</td>
<td>$\Pi_i$ - introduction proof</td>
</tr>
<tr>
<td>$X$ - parameter</td>
<td>$\Pi_E$ - elimination proof</td>
</tr>
<tr>
<td>$t$ - term</td>
<td>$\Pi_C$ - cut normal form proof</td>
</tr>
<tr>
<td>$A$ - atomic formula</td>
<td>$\Pi_A$ - atomic normal form proof</td>
</tr>
<tr>
<td>$E, F, G, H$ - formula</td>
<td>$\Pi_1, \Pi_2$ - cut or atomic normal form proof</td>
</tr>
</tbody>
</table>

Figure 2.5: systems $C_{\Pi_C}$ and $C_{\Pi_A}$ — cut and atomic normal form proof
Introduction Proof Component (II): We refer to a component generated by a goal formula as an introduction component (II). A II is characterized by the inductive definition:

**base** A formula standing alone is a II.

**step** The introduction rules shown on the top left of figure 2.5 form the clauses of the definition. The assumptions generated by the application of ∨I and ¬I rules are initial formulae of elimination components.

The inference rules here reflect the decomposition (backward chaining) of a goal formula into subgoals and assumptions.

Elimination Proof Component (IE): We refer to a component associated with an assertion as an elimination proof component (IE). A IE is characterized by the inductive definition:

**base** A formula standing alone is a IE.

**step** The inference rules shown on the top right of figure 2.5 form the clauses of the definition. Notice that introduction components may occur as parts of an elimination component.

The inference rules here represent the decomposition (forward chaining) of an assertion into simpler subassertions and goals.

Notice that introduction and elimination components are finite. The subformula property limits the number of rule occurrences within a component to the number of logical operator occurrences in the initial formula from which the component is derived.

A cut normal form proof consists of instances of introduction and elimination components “pasted” together using the cut rule of inference as “glue”. The following definition reflects a backward chaining strategy for the construction of cut normal form proofs.

Cut Normal Form Proof (IC): A cut normal form proof (IC) is determined by the inductive definition:

**base:** A II is a IC.

**step:** Given a IC with premiss $F$, a IE with conclusion $F$ then we can compose the IC and IE. This step is symbolised by the CUT rule at bottom left of figure 2.5.

---

3 Proof strategies are discussed in detail in the next chapter.
CHAPTER 2. NATURAL DEDUCTION

The simple mapping between normal form and cut normal form proofs is exhibited by the following lemma.

**Lemma 2** \( \Delta \vdash G \) \iff \( \Delta \vdash G \)

**Proof:**
\( \Delta \vdash G \iff \Delta \vdash G \)

Every \( \Pi_{CUT} \) proof may be transformed into a \( \Pi_{ANF} \) proof by removing every cut rule occurrence by application of the cut-reduction transformation below.

\[
\frac{\Pi_2}{\neg F} \quad \frac{\neg \text{CUT}}{(F)} \quad \rightarrow \quad \frac{\Pi_2}{\neg \neg (F)} \quad \frac{\neg \neg (F)}{\Pi_1}
\]

\( \Delta \vdash G \Rightarrow \Delta \vdash G \)

Every \( \Pi_{ANF} \) proof may be transformed into an \( \Pi_{CUT} \) proof by adding the required instances of the cut rule by exhaustive application of the cut-expansion transformation below.

\[
\frac{\Pi_2}{\neg F} \quad \frac{\neg \text{CUT}}{(F)} \quad \rightarrow \quad \frac{\Pi_2}{\neg (F)} \quad \frac{\neg (F)}{\Pi_1}
\]

\( \square \)

2.6 Atomic Normal Form

Theorem prover technology has developed largely on the assumptions that the basic object language items to be manipulated are atomic formulae (with free variables). In particular, resolution theorem provers and logic programming systems employ clause indexing and unification mechanisms that incorporate this assumption. Atomic normal form proofs are tailored to meet these technological constraints.

**Atomic Normal Form Proof (IIA):** A cut normal form proof is in atomic normal form (ANF) \iff all occurrences of the cut rule have atomic formulae as premiss and conclusion. The ANF cut rule is shown at bottom right of figure 2.5.
The ANF scheme may seem wasteful in requiring that compound formulae always be broken down by elimination rules into their atomic components only to be reassembled by introduction rules. The two stage inference strategy presented in the next chapter effectively limits such waste by preprocessing (partial evaluation).

**Lemma 3** \( \Delta \vdash G \) iff \( \Delta \vdash G \)

**Proof:**
\[
\Delta \vdash G \iff \Delta \vdash G
\]

Every \( \Delta \vdash G \) proof is a \( \Delta \vdash G \) proof.

\[
\Delta \vdash G \implies \Delta \vdash G
\]

The transformations below may be applied to atomize the premiss and conclusion of any \( \text{CUT} \) rule occurrence.

**\( \land \)-expansion**

\[
\begin{array}{c}
\Pi_2 \\
\vdash F \land G \\
\text{E}\land \\
\Pi_1
\end{array} \quad \rightarrow \quad 
\begin{array}{c}
\Pi_2 \\
\vdash F \land G \\
\text{E}\land \\
\Pi_1 \\
\text{CUT} = \\
(F \land G)
\end{array}
\]

**\( \lor \)-expansion**

\[
\begin{array}{c}
\Pi_2 \\
\vdash F \lor G \\
\text{E}\lor \\
\Pi_1
\end{array} \quad \rightarrow \quad 
\begin{array}{c}
\Pi_2 \\
\vdash F \lor G \\
\text{E}\lor \\
\Pi_1 \\
\text{CUT} = \\
(F \lor G)
\end{array}
\]

**\( \Rightarrow \)-expansion**

\[
\begin{array}{c}
\Pi_2 \\
\vdash F \Rightarrow G \\
\text{E}\Rightarrow \\
\Pi_1 \\
\text{CUT} = \\
(F \Rightarrow G)
\end{array} \quad \rightarrow \quad 
\begin{array}{c}
\Pi_2 \\
\vdash F \Rightarrow G \\
\text{E}\Rightarrow \\
\Pi_1 \\
\text{CUT} = \\
(F \Rightarrow G)
\end{array}
\]

\( \text{CUT} = \)
The soundness and completeness of the natural deduction system \( \mathcal{C}_{\Pi} \) with respect to semantic accounts of first order classical logic is well known, see for example [Tennant 78]. Relying on this result, theorem 1 establishes soundness and completeness for the system \( \mathcal{C}_{\Pi A} \).

**Theorem 1** \( \Delta \vdash G \iff \Delta \vdash G \)

**Proof:** Immediately from lemmas 1, 2 and 3 and transitivity of iff. \( \square \)
Chapter 3

Deduction Problems

The notion of a deduction problem and of a proof as its solution are introduced. A formal deduction system is presented for solutions in atomic normal form (ANF solutions). This deduction system incorporates knowledge about the form of the natural deduction rules to reduce the size of the search space. Solutions are computed by a two stage procedure. Stage 1 maps the axioms and query of a deduction problem statement into sets of solution components. Stage 2 searches for a complete solution by composing instances of the components, supplied by stage 1, using resolution refutation technology.

3.1 Computational Preliminaries

Our interest in deductive reasoning is focused on solving deduction problems.

Deduction Problem: Given a logic (as deduction system $S$), a set of axioms $\Delta$ and a query formula $G$, what, if any, proofs of $G$ from $\Delta$ exist in $S$? We symbolise this problem as

$$\Delta \models_s G$$

The members of $\Delta$ (axioms) together with the formula $G$ (query) are the input formulae of the deduction problem.

Solution: Any proof in $S$ having conclusion $G$ and premiss set $\Gamma$, where $\Gamma$ is a subset of $\Delta$, is a solution. Such a solution is summarised by the deducibility assertion:

$$\Gamma \models_s G$$

Consider the example deduction problem:

$$\{p \land q, r\} \models_{c_{\land\lor}} p \lor (q \land r)$$
CHAPTER 3. DEDUCTION PROBLEMS

Notice that although this problem is posed for the system $C_{frN}$ (full first order normal form calculus), the resources of a simple propositional subsystem suffice. We represent the two axioms as shown in figure 3.1 (a) and (b), and the query as in (c). Just the two solutions shown in figure 3.2 exist.

\[
\begin{align*}
\text{- AXIOM} & \quad p \land q \\
\text{AXIOM} & \quad r \\
\text{- QUERY} & \quad p \lor (q \land r)
\end{align*}
\]

(a)  (b)  (c)

Figure 3.1: axioms and query

\[
\begin{align*}
\text{- AXIOM} & \quad p \land q \\
\text{- AXIOM} & \quad q \land r \\
\text{- QUERY} & \quad p \lor (q \land r)
\end{align*}
\]

(a)  (b)

Figure 3.2: solutions

We now introduce notation which reflects more clearly the graph structures employed in computation than does the traditional inference stroke notation. These graphs are bipartite, consisting of formula nodes and inference nodes. Directed edges, drawn down the page, denote the premiss and conclusion relations. The two example solutions are represented by the solution graphs in figure 3.3 (a) and (b). We have distinguished the axioms and query (the source and sink nodes of the directed graphs) by heavily outlined boxes.

\[
\begin{align*}
\text{p} & \quad \text{q} \\
\text{q} & \quad \text{r} \\
\text{p} & \quad \text{q} \\
\text{q} & \quad \text{r}
\end{align*}
\]

(a)  (b)

Figure 3.3: solution graphs

The advantage of the graph notation, over the inference stroke notation, is that it is possible to explicitly represent structure shared by multiple solutions. The way such
shared structure arises during computation is illustrated in the next section. The two example solutions share an axiom and conclusion as illustrated by the AND/OR graph of figure 3.3 (c). The formula nodes of the graph are OR nodes, in the sense that just one of the incoming conclusion edges is included in a solution. The inference nodes are AND nodes, as every incoming premiss edge is included in a solution.

In this chapter we specialize the deduction system $C_{HA}$ for ANF proofs, developed in the preceding chapter, so that it only admits proofs that are solutions. The aim of this new system $C_{T}(\Phi)$ is to reflect, as clearly as possible, an efficient method for computing solutions for a given deduction problem $\Phi$. The problem statement is represented by a set of (degenerate) rules of inference, while the (proper) rules of inference are as shown in figure 3.13. The rules in this figure are inherited from the system $C_{HA}$, with some modifications. A detailed examination of each of the rules of inference, particularly their computational properties, can be found in the next chapter. The present chapter sets up the framework for this study by presenting a more global picture of the computation.

3.2 Search Spaces and Strategies

The purpose of this section is to highlight the computational advantages of the ANF scheme. Both purely forward and purely backward chaining search strategies perform poorly given the natural deduction rules. The computational reading of the $C_{T}(\Phi)$ system is as a combination forward backward strategy.

The process of searching for solutions to deduction problems, by the direct application of natural deduction rules, is represented in outline by the Prolog procedure solve/3 in figure 3.4. This procedure returns a complete AND/OR graph as the Answer to a deduction Problem to be solved in a given Logic. Given the set of inference Rules for Logic and an initial representation of the problem as an AND/OR Graph, the Answer is constructed by the search/3 procedure. This procedure maintains state as an (possibly disconnected) AND/OR Graph. At each iteration the graph is extended by the application of Rules of inference to a part of the Graph called Focus. The search/3 procedure leaves open the search strategy to be used. In the following paragraphs the two common strategies forward chaining and backward chaining and their relationship to the ANF scheme are illustrated.

3.2.1 Forward Chaining Search

Forward chaining search is the computational equivalent of the kind of inductive definition given for natural deduction proofs by the system $C_{II}$ in chapter 2. The strategy starts with the axioms of the deduction problem as the initial (level 0) set of partial
solve(Problem,Logic,Answer) :-
   rules(Logic,Rules),
   initial(Problem,Graph),
   search(Graph,Rules,Answer).

search(Graph,Rules,Graph) :-
   complete(Graph).

search(Graph,Rules,Answer) :-
   not complete(Graph),
   focus(Graph,Rules,Focus),
   next(Graph,Rules,Focus,NextGraph),
   search(NextGraph,Rules,Answer).

Figure 3.4: procedure solve/3

solutions. A new partial solution is added to the set when it is recognised that the pre­
misses of an inference rule find matching partial solutions in the set. The new partial
solution is at level \( m + 1 \), where \( m \) is the maximum of the levels of the premisses. The
solution is complete once its conclusion matches the query.

At any one time a number of rules may match a partial solution at a number of
different foci. The following specializations of the forward chaining strategy reduce
this non-determinism a little. A search strategy is **breadth first** if it produces all partial
solutions at each level \( n \) before any at levels greater than \( n \). A strategy is **depth first**
if it never produces a partial solution at level \( n \), unless no partial solutions at levels
greater than \( n \) can be produced.

\[
\begin{array}{c|c|c|c|c}
\Pi_1 & \Pi_2 & \Pi & \Pi \\
\hline
G & H & E \land F & E \land F \\
\hline
G \land H & E & \neg E & \neg E
\end{array}
\]

Figure 3.5: example forward chaining deduction system

Given the forward deduction system of figure 3.5 consider the problem:

\[ \{ p \land q, r \} \ ?- q \land r \]

Let us walk through the execution of the **solve/3** procedure for this problem. The
initial/2 procedure sets up the disconnected AND/OR graph shown in figure 3.6. The level 0 set of partial solutions consists of just the two axioms $p \land q$ and $r$. The next two levels generated by a forward chaining, breadth first search strategy are displayed in figure 3.7.

```
 $p \land q$    $r$

 $q \land r$
```

Figure 3.6: initial AND/OR graph

From this example it is clear that the uniform forward chaining strategy results in an intolerably large search space. We place the blame on the and introduction rule, which generates new conjunctions irrelevant to the query at hand. The elimination rules, on the other hand, simply break down conjunctions into their conjuncts.

```

Figure 3.7: forward chaining search space
```

### 3.2.2 Backward Chaining Search

Backward chaining search corresponds to the kind of inductive definition given for introduction proof components in the preceding chapter. The strategy starts with the query as the only member of the level 0 set of partial solutions. A new partial solution is added to the set when it is recognised that the conclusion of a rule of inference matches a partial solution already in the set.

Backward chaining leaves the choice of rule and focus undetermined. As in the case of forward chaining, the breadth first and depth first constraints may be applied.

Figure 3.9 illustrates the search space generated by the backward deduction system.
CHAPTER 3. DEDUCTION PROBLEMS

\[
\frac{G \quad H}{\neg\neg (G \land H)} \quad \frac{E \land F}{\neg\neg (E)} \quad \frac{E \land F}{\neg\neg (F)}
\]

Figure 3.8: example backward chaining deduction system

of figure 3.8 for the problem:

\[
\{ p \land q, r \} \ ? \ q \land r
\]

Figure 3.9: backward chaining search space

The roles of the introduction and elimination rules in generating irrelevant partial solutions are now reversed. In the case of backward chaining, we blame the elimination rules for bringing irrelevant conjunctions into the picture. However, a smaller number of irrelevant conjunctions are produced by the single conclusion and elimination rules than by the dual premiss introduction rule. Also, every partial solution contains the query formula as conclusion, resulting in a more focused search.

3.2.3 ANF Search

The above analysis of the combinatorial weaknesses of the pure forward and backward strategies suggests a better approach. The ingredients for the new strategy are:

1. Forward chaining employing elimination rules only.
2. Backward chaining employing introduction rules only.
3. Use of the cut rule of inference to interface the conclusions of 1 with the open premisses of 2.

Figure 3.9 illustrates the search space generated by the combined system of figure 3.10 for the problem:

\[
\{ p \land q, r \} \ ? \ q \land r
\]
The applicability of introduction and elimination rules is here limited by the number of logical constants appearing in an input formula. The exhaustive application of these rules reduces the input formula into its atomic subformulae. The combinatorial search is performed in terms of the cut rule of inference alone. The computational counterpart of cut is the unification operation of [Robinson 65] for which efficient implementation techniques are known.

The above conjunction only example illustrates the computational motivation for the ANF scheme. The formalization and generalization of the scheme to encompass the full first order calculus is taken up next.

### 3.3 Atomic Normal Form Solutions

First we formalize the notion of an ANF solution as the deduction system $C_{\Sigma}(\Phi)$. A second definition, in graph theoretic terms, then more clearly reveals issues in search and representation. We also discuss some of the the major structural features of ANF solutions.
3.3.1 The Deduction System $C_\Sigma(\Phi)$

For a given problem $\Phi$ the deduction system $C_\Sigma(\Phi)$ determines what is to count as an ANF solution of $\Phi$. As already noted, the deduction system $C_\Sigma(\Phi)$ consists of both a problem independent and a problem specific set of inference rules.

Recall that an ANF proof consists of introduction and elimination proof components glued together by instances of the cut rule of inference. Analogously, an ANF solution consists of introduction and elimination solution components glued together by cuts. What distinguishes solution components from proof components is that each solution component is derived from either an axiom or a query. That is, solution components are derived from the input formulae of a particular deduction problem.

The problem independent rules of inference, that are part of any $C_\Sigma(\Phi)$ system, are displayed in figure 3.13. These proper rules are intended to reflect the process of proof construction in more detail than the formulations of the preceding chapter. The appropriate substitution of terms for the quantifier rules is no longer assumed. Consequently the quantifier rules $\forall i$, $\exists i$, $\exists E$ and the structural rule $CUT$ appear in modified form. The $CUT$ rule of inference here incorporates the notion of unification of two atomic formulae, and the subsequent substitution of terms for parameters. Notice also that the assumptions generated by the elimination rules $\forall E$ and $\exists E$ are written as conclusions for these rules. Each of the rules of inference is examined in detail in the next chapter. For now, let us consider the structure of solutions in more global terms.

The problem specific rules of inference of $C_\Sigma(\Phi)$ represent the particulars of the deduction problem $\Phi$ at hand. A problem statement

$$\{F_1, F_2, \ldots, F_n\} \vdash G$$

is represented as follows:

<table>
<thead>
<tr>
<th>introduction component</th>
<th>elimination component</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>$\text{AXIOM} -$</td>
</tr>
<tr>
<td>$\text{- QUERY -}$</td>
<td>$F_i$</td>
</tr>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

Figure 3.12: query and axiom rules

**Query:** The query $G$ is represented by a query rule (a rule of inference having the single premiss formula $G$, but no conclusion) — figure 3.12 (a).

**Axioms:** For every axiom $F_i$ an axiom rule (a rule of inference having conclusion $F_i$, but no premiss) is present — figure 3.12 (b).
CHAPTER 3. DEDUCTION PROBLEMS

The proper inference rules generate two kinds of solution components, using the given axiom and query rules as input. The contribution of a goal formula to the overall solution takes the following form.

**Introduction Solution Component (Σᵢ):** A Σᵢ is determined by the inductive definition:

- **base** The query rule is a Σᵢ. The minor premiss formula of any elimination rule is a Σᵢ.
- **step** The clauses of the definition are supplied by the introduction rules. Note that the assumptions thrown up by the ∨I and ∼I rules give rise to elimination solution components.
  
  Σᵢ is *complete* if none of the clauses apply, otherwise it is *partial*.

The contribution of an assertion formula to the overall solution takes the following form.

**Elimination Solution Component (Σₑ):** A Σₑ is determined by the inductive definition:

- **base** An axiom rule is a Σₑ. An assumption thrown up by either the implication or negation introduction rules is a Σₑ.
- **step** The clauses of the definition are supplied by the elimination rules. Note that the minor premisses of the ∨E and ∼E rules give rise to introduction solution components.
  
  Σₑ is *complete* if none of the clauses apply, otherwise it is *partial*.

We are now in a position to outline a definition for the notion of an atomic normal form solution.

**Atomic Normal Form Solution (Σₐ):** A Σₐ is characterized by the following backward chaining definition:

- **base** A Σᵢ is a Σₐ.
- **step** Given a Σₐ with atomic premiss A, a Σₑ with atomic conclusion B and that \( Aθ = Bθ \) then we may compose Σₐθ and Σₑθ to form a new solution. This computational form of the cut rule is shown at the bottom of figure 3.13.
  
  Σₐ is *complete* if it has no premisses and no conclusion, otherwise it is *partial*.
We also require that a complete $\Sigma_A$ satisfies the following:

**Conditions:** All assumptions must be discharged. Restrictions on the occurrence of parameters for the $\forall I$ and $\exists E$ rules must also be observed. These conditions are detailed in chapter 4.

### 3.3.2 Goals and Assertions

In the preceding section we implicitly partitioned the formula occurrences of a solution into two subsets. First, we distinguished between the query and axiom formulae of a problem statement by representing them as query and axiom rule occurrences respectively. Similarly, we partitioned the compound formula occurrences derived from the axioms and query as either ones to be introduced or eliminated by application of a rule of inference. Finally, any given atomic formula occurrence could be used as either the conclusion or premiss formula of the $\text{CUT}$ rule, but not both.

In the natural deduction literature, see for example [Prawitz 65], the two partitions are referred to as negative and positive (sub)formulae respectively. We adopt the more descriptive terms *goals* and *assertions* for the two kinds of formulae. The goal or assertion character of a formula arises from the role it plays in a solution. The following two definitions enable one to recognise the goal and assertion occurrences in a given solution or component.

**Goal:** All of the following are goal formula occurrences:

- The query.
- Any premiss formula of an introduction rule.
- The premiss $#$ of the absurdity rule.
- Any minor premiss formula of an elimination rule.

**Assertion:** All of the following are assertion formula occurrences:

- An axiom.
- An assumption.
- The conclusion formula of an elimination rule.
- The conclusion formula of the excluded middle rule.

One can easily adapt the above characterization of goal and assertion occurrences to partition the subformulae of a deduction problem statement into goal and assertion subformulae.
CHAPTER 3. DEDUCTION PROBLEMS

introduction component

\[
\frac{G \quad H}{(G \land H) \quad \Sigma I}
\]

\[
\frac{\neg v_1 \quad (G \lor H)}{G \quad (G \lor H) \quad \Sigma I}
\]

\[
\frac{[F]}{G \quad \Sigma I}
\]

\[
\frac{G(X^S)}{(\exists x G(x)) \quad \Sigma I}
\]

\[
\frac{\neg \exists x G(x)}{(\exists x G(x)) \quad \Sigma I}
\]

\[
\frac{[G]}{\Sigma I}
\]

\[
\frac{\neg \# \quad (\neg G)}{\Sigma I}
\]

elimination component

\[
\frac{\Sigma E}{E \land F \quad \Sigma E}
\]

\[
\frac{E}{E \land F \quad \Sigma E}
\]

\[
\frac{E}{E \land F \quad \Sigma E}
\]

\[
\frac{F}{E \land F \quad \Sigma E}
\]

\[
\frac{F}{E \land F \quad \Sigma E}
\]

\[
\frac{\Sigma E}{\Sigma E \quad \Sigma I}
\]

\[
\frac{\Sigma E}{G \quad F \quad \Sigma E}
\]

\[
\frac{\Sigma E}{\exists x F(x) \quad \Sigma E}
\]

\[
\frac{\Sigma E}{\exists x F(x) \quad \Sigma E}
\]

\[
\frac{\Sigma E}{\Sigma E \quad \Sigma I}
\]

\[
\frac{\Sigma E}{\neg F \quad \Sigma E}
\]

\[
\frac{\Sigma E}{\neg F \quad \Sigma E}
\]

\[
\frac{\#}{F \lor \#}
\]

structural rule

\[
\left( \frac{\Sigma E}{B} \right) \quad \overset{= \text{CUT}}{\Sigma A} \quad \text{where:} \quad A \Theta = B \Theta
\]

\[
x \quad \text{variable} \quad A, B \quad \text{atomic formula}
\]

\[
X \quad \text{parameter} \quad E, F, G, H \quad \text{formula}
\]

\[
X^S \quad \text{Skolem parameter} \quad \Sigma I \quad \text{introduction component}
\]

\[
t \quad \text{term} \quad \Sigma E \quad \text{elimination component}
\]

\[
\Theta \quad \text{substitution} \quad \Sigma A \quad \text{atomic normal form solution}
\]

Figure 3.13: system $C_{\Sigma}$—problem independent rules of inference
The distinction between goal and assertion formula occurrences will be used extensively in the subsequent discussion of implementation issues. The distinction is also important for systems, such as Prolog, that implement distinct syntax for the two kinds of formulae. Many more examples of such dual syntax languages are presented in chapter 4.

3.3.3 Search Graphs and Solution Graphs

Two computational issues not well reflected by the above inductive definition of solution are:

- The construction of a solution to a deduction problem involves search.
- How are solutions and search spaces to be represented?

A natural representation of proofs, solutions, search spaces and their fragments is as bipartite acyclic graphs. The two kinds of nodes, formula nodes and inference nodes, are connected by arcs representing the premiss and conclusion relations. By viewing a solution, as an AND subgraph of an AND/OR graph search space, we neatly address both of the above concerns. The OR nodes of the search graph correspond to choice points formed from sets of pairs of goal and assertion atoms satisfying the conditions of the CUT rule, as well as the choice of premiss for or introductions. The AND nodes arise from the set of premisses connected to each inference node occurrence. For foundational material on the AND/OR graph representation and associated search strategies refer to [Nilsson 80].

The following definitions are for the general notions of search and solution graphs. We will make use of them a little later in order to define ANF search graphs and ANF solution graphs.

**Search Graph:** A search graph is a directed acyclic bipartite graph. The two classes of nodes are formula nodes and inference nodes. The structure of the graph is constrained by the following two clauses:

- A formula node standing alone is a search graph.
- A formula node may be the conclusion of any number of inference nodes provided that for each inference node every one of its premisses is a search graph.

Inference nodes are the AND nodes of the graph in the sense that all the premisses are to be present. Formula nodes are OR nodes in the sense that for any solution subgraph they are the conclusion of just one inference node.
Solution Graph: A solution graph is a finite subgraph of a search graph. A solution graph may be either *complete*, in which case;

- Every formula node in the solution graph is the conclusion of exactly one inference node.

or *partial*, in which case;

- Every formula node in the solution graph is the conclusion of at most one inference node.

3.4 Extend and Compose

The deduction systems presented in chapter 2 incorporated inferential machinery common to all problem domains (the logical constants \&, \lor, \neg, \forall, \exists and \#). In order to solve deduction problems more efficiently, we are now ready to replace these fine grained, universal rules of inference by more powerful, problem specific rules.

The construction of an ANF solution proceeds in two distinct phases, extend and compose. There are a number of perspectives on what happens during the two phases and why this division into two phases is desirable:

- The inferential resources implicit in the query and axioms are made explicit as sets of solution components (inferential extensions) by extend. The task of compose is to build a solution by joining together instances of the components produced by extend by applying the CUT rule of inference.

- In the extend phase, the introduction and elimination rules for the logical operators are removed by combining rules, resulting in a reduced number of more powerful inference rules. During compose reasoning is carried out using these derived rules of inference.

- Current theorem prover technology has largely developed on the assumption that the fundamental operation of an inference engine is the unification of two atomic formulae. Extend allows us to partially evaluate the problem to be presented to compose, which is such an inference engine.

3.4.1 Extend

Extend is a computational realization of the inductive definitions of solution components. It maps each input formula into a set of solution components. We will refer to such a set of solution components as the *inferential extension* of its input formula.
Extend Algorithm: Starting with the axioms and query, repeatedly apply either of the two steps below, until neither step is applicable.

Goal Introduction: Select a compound (non-atomic) goal formula occurrence. Select an introduction rule with a matching conclusion formula. Instantiate the introduction rule. The premiss formulae of the new instance are goals. Any assumptions "thrown up" by the introduction rule are assertions.

Assertion Elimination: Select a compound (non-atomic) assertion formula occurrence. Select an elimination rule with a matching major premiss formula. Instantiate the elimination rule. The conclusion formulae of the new instance are assertions. Any minor premiss formulae are goals.

![Figure 3.14: inferential extension of a query](image)

The extend algorithm maps a query into its inferential extension, an AND/OR graph of the form illustrated in figure 3.14. This AND/OR graph consists of multiple connected components, call them search components. There are two kinds of search components here:

Query Search Component: This is a tree form AND/OR graph with the query as root and atomic goal formulae as leaves. The solution graphs of this search component is a set of introduction solution components.

Assumption Search Components: The reduction of a goal, that is either a negated formula or an implication, by an introduction rule "throws up" assumptions. These assumptions are to be made available for the proof of a
subset of the premisses of the *parent* introduction rule only. An assumption search component consists of a set of elimination solution components, being the solution graphs of this AND/OR graph.

The extend algorithm maps an axiom into an AND/OR graph of the form illustrated in figure 3.15. This inferential extension of an axiom consists of two kinds of search components:

**Axiom Search Component**: This is a directed AND/OR graph with an axiom and a set of atoms as source nodes and a set of atoms as sink nodes. Each solution graph consists of a single elimination component together with a set of introduction components rooted at minor premisses of elimination rules.

**Assumption Search Components**: These are found in the inferential extensions of both axioms and queries. Assumption components are of the same form as axiom components. However, the availability of assumption components is restricted to proving subgoals arising from its parent introduction rule occurrence only.

The notions of inferential extension, search components and solution components have now been outlined for the full first order classical calculus. In their present form these notions are not adequate for the practical implementation of an inference engine. For example, the MX (excluded middle) rule cannot be treated the way axioms are, lest we drown in an ocean of inferential extensions. These computational issues are examined in subsequent chapters.
3.4.2 Compose

**Compose** is a computational realization of the inductive definitions of ANF solution. It is confronted with the task of constructing solutions given the set of inferential extensions produced by **extend**. The search space for solutions is a graph consisting of search components connected by choice points (sets of CUT rule instances). A solution is a subgraph of the above, consisting of solution components connected by CUT rule instances.

As well as inferential extensions derived from distinct input formulae, multiple copies of the one inferential extension and its various components may occur as part of the one search space or solution. To ensure that parameters in one occurrence are not captured by a substitution computed for another, all inferential extension occurrences, and within these multiple copies of assumption components, need to be renamed apart. We will refer to such renamed copies as *inferential extension instances* and *component instances*.

**Renaming Apart:** A set \( \{\sigma_1, \sigma_2, \ldots, \sigma_n\} \) of component occurrences is renamed apart by the set of substitutions \( \{\Theta_1, \Theta_2, \ldots, \Theta_n\} \) with respect to the set of parameters \( \{X_1, X_2, \ldots, X_m\} \) if:

- **Renaming:** Every substitution \( \Theta_i \) is a full renaming substitution. That is, \( \Theta_i \) is of the form \( \{X_1 = Y_1^i, X_2 = Y_2^i, \ldots, X_m = Y_m^i\} \), where every \( Y_j^i \) is a parameter that does not occur in \( \{X_1, X_2, \ldots, X_m\} \).

- **Apart:** Every substitution \( \Theta_i \) renames every \( X_j \) uniquely. That is, every new name \( Y_j^i \) is different from every other new name.

Multiple occurrences of the one inferential extension are renamed apart with respect to the set of parameters appearing in its axiom or query search component. Within a single inferential extension occurrence, multiple occurrences of the one assumption search component are renamed apart with respect to the parameters that appear in that component but do not appear in its parent component.

When we draw search spaces and solution graphs we indicate renaming by subscripting parameter occurrences. Implementations of theorem proving systems commonly allocate a unique memory cell for each distinct parameter, making explicit renaming by substitution unnecessary.
CHAPTER 3. DEDUCTION PROBLEMS

By a \textsc{cut} rule instance we mean the following:

\textbf{Cut Instance:} A \textsc{cut} instance is a triple $\langle A, B, \Theta \rangle$ where $A$ is an atomic goal occurrence, $B$ an atomic assertion occurrence and $\Theta$ the most general unifier of $A$ and $B$ [Robinson 65], [Lassez et al. 88].

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (A) at (0,0) {$A$};
  \node (B) at (1,1) {$B$};
  \node (Theta) at (2,0) {$\Theta$};
  \node (A') at (4,0) {$A$};
  \draw[->] (B) -- (Theta);
  \draw[->] (Theta) -- (A');
\end{tikzpicture}
\caption{Cut rule instance}
\end{figure}

The notation of figure 3.16 is used for \textsc{cut} instances. The substitution is explicitly represented, reflecting in part the structure sharing (non-copying) implementation technique for inference engines [Boyer & Moore 72]. This is an important point for the economical representation of multiple (partial) solutions, as required by the multiple context evaluator discussed in chapter 5.

For search spaces we group sets of \textsc{cut} instances sharing a common conclusion formula into \textit{choice points}.

\textbf{Choice Point:} A choice point is a triple $\langle A, \overline{B}, \overline{\Theta} \rangle$ where $A$ is an atomic goal occurrence, $\overline{B}$ is a tuple of atomic assertion occurrences and $\overline{\Theta}$ a tuple of the most general unifiers of the corresponding element of $\overline{B}$ and $A$.

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (A) at (0,0) {$A$};
  \node (B1) at (-1,1) {$B_1$};
  \node (Bn) at (-1,-1) {$B_n$};
  \node (Theta1) at (1,1) {$\Theta_1$};
  \node (Thetan) at (1,-1) {$\Theta_n$};
  \draw[->] (B1) -- (Theta1);
  \draw[->] (B1) -- (A);
  \draw[->] (Bn) -- (Thetan);
  \draw[->] (Bn) -- (A);
\end{tikzpicture}
\caption{Choice point}
\end{figure}

In the interest of computational efficiency, it is important to minimize the number of \textsc{cut} instances in choice points. In the absence of any context for a goal atom $A$,
one cannot exclude any unifiable assertion \( B \) of any search component. Two kinds of context for \( A \) enable some pruning of choice points. Call the \( B \)s that remain the \emph{admissible assertions} for \( A \).

\textbf{Subgoal Context:} If the path from \( A \) down to the query is known, only assumption search components arising from \( \forall \) and \( \neg \exists \) rule applications on the path need be included.

\textbf{Case Argument Context:} Admissible assertions arising from \( \forall \exists \) rule applications are determined by a case argument mechanism, described in section 4.5.3.

We now have on hand the ingredients needed to characterize the search space for ANF solutions for a given deduction problem \( \Phi \).

\textbf{ANF Search Graph:} The ANF search graph for problem \( \Phi \) is a search graph (as defined in section 3.3.3) satisfying the following properties:

\textbf{Composition:} The graph consists of a set of inferential extensions derived from the input formulae of \( \Phi \). These inferential extensions are renamed apart.

\textbf{Form:} The system \( \mathcal{C}_E \), figure 3.13, incorporates a backward chaining form of the \( \text{CUT} \) rule. Application of this rule generates a tree form search graph rooted at a single query search component instance.

\textbf{Completeness:} Every atomic goal, occurring in the graph, has a complete choice point. A choice point is complete if every admissible assertion is in the choice point.

A search space, as defined above, may be infinite in extent. We plan to countenance only finite subgraphs of the search graph as solutions. Some nonterminating paths may be pruned on the grounds that they cannot be part of any solution. Clearly any proposed solution of the form shown in figure 3.18 (a) may be discarded, as an unnecessarily convoluted form of (b). More generally, we have the notion of a loop free solution:

\textbf{Loop Free Solution:} Any proposed solution of the form shown in figure 3.18 (c), where \( A_1 = A_2 \Theta \) and \( \Sigma_1 \) does not discharge any assumptions in \( \Sigma_2 \), should be discarded in favour of the simpler form shown in (d).

A definition for an ANF solution for deduction problem \( \Phi \) follows.
ANF Solution Graph: An ANF solution graph for $\Phi$ is a finite subgraph of an ANF search graph for $\Phi$ satisfying these further constraints:

**Query Relevance:** The query is included as the root (single sink node) of the solution.

**Axiom Relevance:** The source nodes of the solution graph are either axioms or discharged assumptions.

**Resolved Choice:** Any formula node, in the solution, is the conclusion of exactly one inference node. This implies that only a single $\text{CUT}$ instance from any choice point is included in the solution.

**Substitution Consistency:** The entire set of substitutions (composition of substitutions [van Vaalen 75]) associated with $\text{CUT}$ rule instances in the solution is consistent.

**Loop Freeness:** Only loop free solutions, as defined above, are admitted.

The computational interpretation of the above, as a constraint satisfaction problem, is taken up in chapter 5. A large body of research addresses the problem of efficiently implementing resolution refutation systems. Like compose these systems are required to recognise subgraphs, with consistent substitutions generated by a unifier, while navigating within large AND/OR search spaces. Much of this work carries over directly to inference engines for ANF natural deduction. This idea is pursued further in chapter 5.
Chapter 4

Expressive Power and Inference

Normal form natural deduction exhibits a simple correspondence between the expressive power of a language and the deductive machinery required for its implementation. A hierarchy of deduction systems properly contained in the deduction system for classical logic is explored incrementally. The important languages encountered along the way are identified. A short detour, to survey negation as failure and relevant deduction, concludes the chapter.

4.1 Containment and Conservative Extension

The normal form and atomic normal form formulations of deducibility for classical logic exhibit two important containment properties. The first of these is the simple correspondence between the expressive power of a language and the introduction and elimination rules required to solve deduction problems stated in that language.

Sublanguage Property: For any solution \( \Sigma \) of the deduction problem \( \Phi \):

- \( \Sigma \) contains instances of introduction rules only for those operators that appear as the primary operator of a goal subformula of \( \Phi \).
- \( \Sigma \) contains instances of elimination rules only for those operators that appear as the primary operator of an assertion subformula of \( \Phi \).

The second containment property extends the first, centering on the acceptance or rejection of the absurdity rule and excluded middle as acceptable principles of reasoning.

Sublogic Property: The rejection of the rule of excluded middle (\( MX \)) from the classical system yields a system for intuitionistic logic [Dummett 77]. The rejection of the absurdity rule (\( #X \)) from the intuitionistic system yields minimal logic [Johansson 36].
In this chapter, we consider the implementation of inference engines for a hierarchy of properly contained subsystems of $C_\Sigma$, the system for atomic normal form solution in classical logic. The containment of systems and corresponding languages is illustrated in figure 4.1. This hierarchy consists of the languages:

- $\mathcal{H}$: Horn language
- $\mathcal{E}$: positive Edinburgh Prolog language
- $\mathcal{D}$: positive definite language
- $\mathcal{P}$: positive language
- $\mathcal{M}$: minimal logic
- $\mathcal{I}$: intuitionistic logic
- $C$: classical logic

The Horn language system is the simplest, requiring just four rules of inference. Each of the following systems is a conservative extension of the preceding one obtained by adding the inference rules indicated in figure 4.1.

### 4.2 The Horn Language

The deduction system $H_\Sigma$, for ANF solutions for problems posed in the Horn language, is shown in figure 4.2. The Horn language occupies a special niche in resolution refutation proof theory [Kowalski 79]. This is also the case for the ANF formulation, which supports the reading of a Horn formula as a rule of inference, as discussed below.
CHAPTER 4. EXPRESSIVE POWER AND INFERENCE

introduction component elimination component

\[
\frac{G}{(G \land H)} \quad \frac{H}{\Sigma_I} \quad \frac{G \supset F}{F} \quad \frac{\Sigma_E}{\Sigma_I}
\]

\[
\frac{G(X)}{(\exists xG(x))} \quad \frac{\forall xF(x)}{F(x)} \quad \frac{\Sigma_E}{\Sigma_I}
\]

structural rule

where:

\[A \Theta = B \Theta\]

\[x\] variable
\[X\] parameter
\[\Theta\] substitution
\[A, B\] atomic formula
\[E, F\] assertion formula
\[G, H\] goal formula
\[\Sigma_I\] introduction component
\[\Sigma_E\] elimination component
\[\Sigma_A\] atomic normal form solution

Figure 4.2: system \(H_\Sigma\) — atomic normal form solution for the Horn language

4.2.1 Horn Formulae and Their Extensions

A feature of the Horn language, and many of the subsequent languages, is that assertion and goal subformulae have distinct syntax. For these dual syntax languages we use the syntactic variables \(E\) and \(F\) to stand for assertion formulae, and \(G\) and \(H\) for goal formulae. The syntax of Horn axioms and queries is illustrated in figure 4.3. The large prefix universal (existential) quantifiers in this figure denote the universal (existential) closure of the prefixed formula — that is, the formulae are in prenex form. Notice that although this conventional notion of Horn formulae requires prenex quantification, the deduction system \(H_\Sigma\) does not.

The inferential extension of a Horn language axiom is illustrated in figure 4.3 (a), the inferential extension of a query in (d). For the purpose of AND/OR graph search
we can prune away the input formula resulting in the simpler forms (b) and (e). For rule based inference, (b) and (e) may be represented as the derived rules of inference (c) and (f) respectively. The prime notation in the figure indicates that applications of the universal elimination ($\forall E$) and existential introduction ($\exists I$) rules have replaced the bound variables of the input formula by parameters.

\[
\text{Horn Axiom } F: \quad \forall A_1 \land A_2 \land \ldots \land A_n \supset B
\]

\[
\text{Horn Query } G: \quad \exists A_1 \land A_2 \land \ldots \land A_n
\]

The simple correspondence between Horn formulae and derived rules of inference, illustrated in figure 4.3, supports a proof theoretic view of a Horn problem. Read the input formulae, not as formulae, but as rules of inference or the clauses of an inductive definition of provable atomic formulae. This view is proposed and extended towards more expressive languages by [Hallnäs & Schroeder-Heister 90].

4.2.2 CUT and The Quantifier Rules $\forall E$ and $\exists I$

It is time to consider in detail the role played by parameters and unification in the process of constructing solutions. The story begins here and is continued, when the other two quantifier rules $\forall I$ and $\exists E$ are adopted.

A solution is a composition, by application of the CUT rule, of renaming instances of search components. Applications of $\forall E$ and $\exists I$ replace the bound variables of input formulae by parameters, so that only quantifier free atoms appear as premisses and conclusions of components. The cut principle requires that its premiss and conclusion formulae be syntactically identical. The new, more procedural version of the principle

Figure 4.3: inferential extensions of Horn axioms and queries
CHAPTER 4. EXPRESSIVE POWER AND INference

is expressed by the clause:

Given the two components \( \frac{\Sigma_1}{B} \) and \( \frac{(A)}{\Sigma_2} \) and a substitution \( \Theta \) such that \( (A\Theta = B\Theta) \)

then \( \left( \frac{\Sigma_1}{B} \right)_{\text{cUT}} \) is a component.

The \( \forall \land \exists \) rules allow for any term whatever to replace a parameter. Given the two quantifier free atoms \( A \) and \( B \) we want to find a substitution (of terms for parameters) \( \Theta \) such that \( A\Theta \) and \( B\Theta \) are the most general syntactically identical substitution instances of \( A \) and \( B \). That is, \( \Theta \) is the most general unifier (mgu) of \( A \) and \( B \).

**Most General Unifier (mgu):** The mgu \( \Theta \) of the two atoms \( A \) and \( B \) is a set of equality assertions:

\[
\{ X_1 = t_1, X_2 = t_2, \ldots, X_n = t_n \}
\]

\( \Theta \) satisfies the following constraints:

**Unifier:** \( A\Theta \) and \( B\Theta \) are syntactically identical.

**Most General:** Any common substitution instance of \( A \) and \( B \) is also a substitution instance of \( A\Theta \) (\( B\Theta \)).

**Solved Form:** Each \( X_i \) is a distinct parameter that occurs in either \( A \) or \( B \).
Each \( t_i \) is a term containing parameters that occur in either \( A \) or \( B \) but none that occur as an \( X_i \).

The above constraints enable efficient composition of mgu's, a question considered in detail in chapter 5. The computation of an mgu given \( A \) and \( B \) has been extensively studied since the pioneering work of [Robinson 65], see for example [Lassez et al. 88].

**4.3 The Positive Edinburgh Prolog Language**

The logic programming language Prolog developed in the proof theoretic context of resolution refutation for the Horn language. Prolog implementors have, however, recognised the relative simplicity of the deductive machinery required for a richer goal syntax. Two syntactic extensions, negated and disjunctive goals were admitted by the classic Edinburgh dialect [Clocksin & Mellish 81]. The negation as failure (NAF) extension is discussed separately in section 4.9. The disjunctive goals extension is taken up here.
4.3.1 The Or Introduction Rules \( \lor \)

The sublanguage property states that just the introduction rules for the logical operators appearing in the goal syntax are required for a complete normal form deduction system. Thus to extend the deduction system \( \mathcal{H}_E \) for disjunctive goals, we simply add the two or introduction rules of figure 4.4 to the existing rules.

\[
\frac{G}{(G \lor H)} \quad \frac{H}{(G \lor H)}
\]

Figure 4.4: or introduction rules \((\lor)\)

The form of the inferential extensions for the extended syntax is illustrated in figure 4.5 (a) and (d). An inferential extension still consist of a single search component. The search component still has a single atomic conclusion. However, a search component may now contain OR branches, giving rise to multiple solution components. Each solution graph of the AND/OR graphs (b) or (e) is a derived rule of inference (c) or (f), featuring a subset of the atomic premisses.

Edinburgh Axiom \( F \): \( \forall A_1 \circ A_2 \circ \cdots \circ A_n \supset B \) (where: \( \circ \) is \( \land \) or \( \lor \))

\[
\frac{\neg\text{AXIOM} \quad \text{AND/OR tree} \quad \text{AND/OR tree} \quad \text{AND/OR tree} \quad \text{AND/OR tree}}{F \quad A_1' \quad A_2' \quad \cdots \quad A_n' \quad A_1' \quad A_2' \quad \cdots \quad A_n' \quad A_{i_1}' \quad A_{i_2}' \quad \cdots \quad A_{i_m}'}
\]

(a) (b) (c)

Edinburgh Query \( G \): \( \exists A_1 \circ A_2 \circ \cdots \circ A_n \) (where: \( \circ \) is \( \land \) or \( \lor \))

\[
\frac{\neg\text{QUERY} \quad \text{AND/OR tree} \quad \text{AND/OR tree} \quad \text{AND/OR tree} \quad \text{AND/OR tree}}{A_1' \quad A_2' \quad \cdots \quad A_n' \quad A_1' \quad A_2' \quad \cdots \quad A_n' \quad A_{i_1}' \quad A_{i_2}' \quad \cdots \quad A_{i_m}'}
\]

(d) (e) (f)

Figure 4.5: inferential extensions of Edinburgh axioms and queries

The procedural semantics of Prolog dictate that search component AND/OR trees be traversed left to right with backtracking to the most recent OR node on goal failure. For a more focused discussion on the relationship between logic programming and
atomic normal form natural deduction see [Keronen 91].

4.4 The Positive Definite Language

In this section we consider the deductive machinery required for a full positive goal syntax. The universal quantifier introduction ($\forall i$) and implication introduction ($\exists i$) rules are added to the Edinburgh system. This language is definite in the sense that disjunctive and existentially quantified assertion formulae are not admitted. The conclusion of a rule derived from a positive definite axiom is still an atomic formula.

4.4.1 The Quantifier Introduction Rule $\forall i$

\[
\frac{G(X^S)}{\forall i \frac{(\forall xG(x))}{\Sigma i}}
\]

Figure 4.6: universal quantifier introduction rule ($\forall i$)

The universal quantifier introduction rule $\forall i$ replaces the bound variable $x$ in the goal $\forall xG(x)$ by a parameter $X^S$, resulting in the subgoal $G(X^S)$, see figure 4.6. The superscript $S$ is used to distinguish the parameter generated by application of this rule as a Skolem parameter. Unlike the parameter generated by an application of $\exists i$, a renamed Skolem parameter $X_i^S$ is subject to the following two constraints on its use:

**Skolem Constraint:** $X_i^S$ is to appear literally in the solution. The mechanism to enforce this constraint is simply to treat the parameter as if it were a constant symbol, identical only to itself [Skolem 28]. That is, a Skolem parameter may only appear on the right hand side of any element $X_i = t_i$ of an mgu. As an example, the mgu in figure 4.7 (a) violates this constraint.

**Dependency Constraint:** $X_i^S$ may not appear in any assumption on which $G(X_i^S)$ depends. Assumptions may be present once any of the rules $\exists i$, $\neg i$, $\exists \vee i$ or $\forall \vee i$ are admitted. In general terms, enforcing the dependency constraint requires that mgu elements of the form $X_i = Y_j^S$ be checked to determine that the $\forall i$ rule responsible for $Y_j^S$ occurs low enough in the solution, so that all assumptions in which $X_i$ occurs have been discharged. As an example, the mgu in figure 4.7 (b) violates this constraint.
4.4.2 The Implication Introduction Rule \( \vdash \)

The deduction problem \( \Delta \models (F \supset G) \) is reduced by the implication introduction rule, shown in figure 4.8, to the problem \( \Delta \cup \{F\} \models G \). That is, the antecedent \( F \) is an assumption or temporary axiom that may be used for the purpose of deriving the conclusion \( G \) only.

\[
\frac{[F]}{G} \quad (\vdash)
\]

Figure 4.8: implication introduction rule \( (\vdash) \)

Inferential extension may now contain assumption search components arising from the antecedents of goal implications, as illustrated in figure 4.9 (a) and (d). For each inferential extension there is a set of derived rules of inference of the form (c) or (f). The intended reading of these rules is: For each premiss \( A_{kx} \) of the derived rule a set of derived rules \( R_{kx} \) is available as assumptions. This generalization of the notion of a rule of inference is explained in more detail in section 5.1.

Notice that the inferential extension of a formula still consists of search components with a single atomic conclusion. Hence the natural deduction formulation retains the definite character of the deduction problem. In contrast the resolution refutation proof theory is more severely affected. While any Edinburgh formula can be rewritten as a logically equivalent set of Horn clauses, once implications as goals are admitted we are outside Horn clause resolution. As an example, the axiom \( (F \supset G) \supset H \) rewrites to the set of clauses \( \{F \vee H, \neg G \vee H\} \). The multiple positive literals of the resulting clauses call for a full resolution refutation strategy [Chang & Lee 73].

Though less severe than in the case of resolution refutation, there is still a computational price to be paid for the expressive power of implications as goals. The search process is complicated by the presence of assumptions. The set of search components...
CHAPTER 4. EXPRESSIVE POWER AND INference

Positive Definite Axiom $F$: $\forall G \supset B$

![Diagram of Axiom F](image)

Positive Definite Query $G$: any formula constructed using operators $\exists, \forall, \wedge, \vee$ and $\supset$

![Diagram of Query G](image)

Figure 4.9: inferential extensions of positive definite axioms and queries

available for constructing the choice point for a given goal atom now depends on its subgoal context. Recall that this context is determined by the path from the query to the goal atom in question. This raises the following challenges for inference engine implementations:

- A choice point cannot be completely constructed until the path to the query is known. A simple approach to this problem is to employ backward chaining search in the compose phase of the inference engine.

- Efficient logic programming engines construct choice points, as far as possible at compile time. In the presence of implications as goals, such a mechanism needs to be extended to incorporate the lookup of search components from a tree structured database at run time.

- The various search components making up an inferential extension may share common parameters, as well as containing parameters to be renamed for each
4.5.2 The Existential Elimination Rule $\exists E$

The existential elimination rule $\exists E$, shown in figure 4.12, reduces an assertion $\exists x F(x)$ to the quantifier free assertion $F(X^S)$. Like the $\forall t$ rule, this rule generates a Skolem parameter, subject to both the Skolem and dependency constraints.

$$\Sigma E$$

$$\exists x F(x) \quad F(X^S)$$

Figure 4.12: existential quantifier elimination rule ($\exists E$)

As we moved from the orthodox formulation of natural deduction proof (chapter 2) to the more computational notion of a solution (chapter 3), we adopted new notation for existential elimination. Figure 4.13 (a) illustrates the orthodox notation for an application of existential elimination, and (b) our computational notation for the same. The transformation from the form (a) to the form (b) can always be performed, provided the existential elimination discharges its assumption. That is, the new notation does not permit vacuous applications of the rule. The new notation is also more convenient in connection with the AND/OR graph search paradigm.

Figure 4.14 displays an example solution using the orthodox notation (a) and the computational notation (b). As a disadvantage of the new notation, the assumption does not stand out as well here as it does in the orthodox notation. Figure 4.15 illustrates the need to carefully discharge assumptions and to check the dependency constraint to avoid unsound inference.

The discharge of the assumption is a simple deterministic operation. To ensure completeness one must discharge the assumption as high up in the solution graph as possible. A simple implementation may traverse down the solution, applying substitutions, until a formula occurrence that does not contain the Skolem parameter in
CHAPTER 4. EXPRESSIVE POWER AND INference

4.5.3 The Or Elimination Rule $\forall E$

The or elimination rule $\forall E$ of figure 4.16 may be read as: The assertion $E \lor F$ gives rise to two possible worlds, one contains $E$, the other $F$. More generally, $n$ binary disjunctions give rise to $2^n$ possible worlds. Any goal formula $G$ is to be demonstrated.
for all worlds. In the presence of disjunctive assertions, a solution for query $G$ consists of a set of case arguments. Each case argument establishes $G$ for a subset of worlds.

$$
\Sigma_E \\
E \lor F \\
\neg \lor \Sigma_E \\
E \quad F
$$

Figure 4.16: or elimination rule ($\lor E$)

For the same reasons as in the case of the $\exists E$ rule, we employ an alternative notation for the computational notion of or elimination. The transformation between applications of the orthodox natural deduction rule and our computational rule is illustrated in figure 4.17. Note that natural deduction (even in normal form) permits vacuous applications of the $\lor E$ rule, but that such application cannot be expressed in the new notation. For a different perspective on multiple conclusion rules of inference see [Shoesmith & Smiley 78].

$$
\Sigma \\
E \lor F \\
\neg \lor \Sigma \\
(E) \quad (F) \\
\Sigma_1 \quad \Sigma_2
$$

Figure 4.17: notation for or elimination

In the presence of the $\lor E$ rule, search components contain AND related atomic conclusions. We have now reached the most general form for search components, illustrated in figure 4.18.

The word "AND", used above to describe the relationship between conclusion atoms, is not totally satisfactory. It is true that a complete set of case arguments corresponds to a solution graph of the search space when just one disjunctive assertion occurs in the solution. There are two ways in which this model fails to reflect the application of disjunction elimination more generally:

- Any one case argument may use only one of the disjuncts from any one solution component instance. The example in figure 4.19 illustrates an unsound solution, resulting from a failure to observe this condition.
A solution must contain case arguments to cover all worlds generated by disjunctive assertions. Figure 4.20 illustrates a failure on this count.

The set of worlds to be covered by case arguments is generated as the cartesian product of sets of disjunctive conclusions. For the example of figure 4.21 there are the four worlds:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>w1</td>
<td>w2</td>
</tr>
<tr>
<td>d</td>
<td>w3</td>
<td>w4</td>
</tr>
</tbody>
</table>

There are three case arguments in figure 4.21. The leftmost case argument establishes w1 as inconsistent. The middle case argument concludes f for the world w2. The rightmost case argument concludes f for the two worlds w3 and w4. In contrast, the
unsound case argument in figure 4.19 used more than one disjunct from an axis of such a diagram, while the set of case arguments in figure 4.20 failed to cover the two worlds w2 and w3.

\[
\begin{align*}
\{ & a \lor b \\
 & c \lor d \\
 & \neg(a \land c) \\
 & (b \land c) \supset f \\
 & d \supset f \\
\} \quad \vdash f
\end{align*}
\]

Figure 4.21: or elimination example

The above discussion suggests that we recognise the supervision of case arguments as a separate subtask for the inference engine. This supervisory level of the inference engine sets up the case argument context, being a set of single conclusion derived rules, and calls for a case argument search in that context.

### 4.6 Minimal Logic

The addition of the introduction and elimination rules for the negation operator (\(\neg\)) to the positive system results in a system for minimal logic [Johansson 36]. The elimination rule for negation constitutes a simple definition of the notion of contradiction. The subsequent use that is made of contradiction in deriving new conclusions is more controversial. The introduction and elimination rules for negation highlight the inadequacy of pure forward or backward chaining search strategies for compose.

\[
\begin{align*}
\frac{[G] \quad \#}{\neg \vdash \#} & \quad & \frac{\Sigma_E \quad \Sigma_t}{\neg F \quad \#} \\
(\neg G) & \quad & (\neg F)
\end{align*}
\]

Figure 4.22: negation introduction (\(\neg i\)) and elimination (\(\neg E\)) rules

#### 4.6.1 The Negation Rules \(\neg i\) and \(\neg E\)

The elimination rule for negation, shown in figure 4.22 (b), detects a contradiction (\(\#\)), given that both a formula and its negation have been established. The corresponding introduction rule, shown in (a), can be viewed as an amalgam of two principles:
Reductio ad Absurdum: A familiar method of argument to establish that a negation formula \( \neg G \) holds is by demonstrating that a contradiction can be derived from the assumption \( G \) together with other current assumptions and axioms.

Absurdity Principle: Adherence to the semantics of classical logic demand that any formula whatsoever be derivable from a contradiction.

The reductio ad absurdum principle provides us with two points, the assumption \( G \) and the conclusion \( \# \), around which to construct a solution. Let us distinguish this as a new kind of deduction problem.

Relevant Deduction Problem: Given a set of required axioms \( \Gamma \) and a set of ordinary axioms \( \Delta \), is there a proof of \( G \) in the system \( S \)? In symbols:

\[
\Gamma : \Delta \vdash^S G
\]

Any solution to the deduction problem

\[
\Gamma \cup \Delta \vdash^S G
\]

that features every member of \( \Gamma \) as a premiss is a solution for the corresponding relevant deduction problem.

Neither the pure backward nor forward chaining search strategy makes full use of the constraints on premisses and conclusion. Notice that this point can also be made for the implication introduction rule. The absurdity principle is discussed in the next section.

4.7 Intuitionistic Logic

Intuitionistic deducibility requires an implementation of the absurdity principle for any goal formula, not just the negated ones. A first reading of the absurdity rule, shown in figure 4.23, might then be as a kind of introduction rule to be applied for all goal formula occurrences.

\[
\frac{-\#x \quad \#}{\Gamma \vdash^S G \quad \Sigma_i}
\]

Figure 4.23: absurdity rule (\( \#x \))

Given that the task is to find solutions to a deduction problem \( \Delta \vdash^S G \), application of the absurdity rule can, however, be reduced to the following cases:
• Check the consistency of the original problem theory \( \Delta \). In the case that an inconsistency is found all queries receive an affirmative answer, until the theory is repaired.

• Given the consistency of \( \Delta \), a contradiction may still be derivable in some subgoal or case argument contexts. For each additional assumption we call for a search for contradiction derivable using the assumption in question. Notice that this is another example of a relevant deduction problem.

Many (most) theorem prover implementations do not perform the first of the above checks, preferring to assume the consistency of \( \Delta \). An example of this is the set of support strategy for resolution refutation systems [Chang & Lee 73].

### 4.8 Classical Logic

A system for classical logic results if any one of the constructs shown in figure 4.24 is added to the intuitionistic system. These constructs are:

\[
\begin{array}{c|c|c|c}
F \lor \neg F & \neg \neg \neg G & \neg \neg G & F \\
\hline
G & \# & G & G \\
\hline
(a) & (b) & (c) & (d)
\end{array}
\]

Figure 4.24: excluded middle

(a): Axiom schema for excluded middle,

(c): rule of classical reductio,

(d): rule of double negation.

(b): rule of dilemma,

In the presence of any of these alternatives the subformula property is not strictly observed. To see this, consider the deduction problem

\[
\{ \} \models (a \supset b) \lor (b \supset a)
\]

There is no intuitionistic solution for this problem. The classical solution therefore must include at least one application of the excluded middle principle, and therefore a negated formula occurrence. No negated subformulae, however, occur in the statement of the problem. One of the possible classical solutions is shown in figure 4.26.
rules for moving in negations

\[ \frac{\neg G}{\neg \neg (G \land H)} \]

\[ \frac{\neg H}{\neg \neg (G \land H)} \]

\[ \frac{\neg (G \land H)}{\neg E \neg F} \]

\[ \frac{\neg G \neg H}{\neg (G \lor H)} \]

\[ \frac{\neg (G \lor H)}{\neg E \neg F} \]

\[ \frac{G \neg H}{\neg (G \lor H)} \]

\[ \frac{\neg (G \lor H)}{\neg E \neg F} \]

\[ \frac{\exists x (\neg G(x))}{\neg \forall x G(x)} \]

\[ \frac{\neg \forall x G(x)}{\exists x (\neg F(x))} \]

\[ \frac{\forall x (\neg G(x))}{\neg (\exists x F(x))} \]

\[ \frac{\neg (\exists x G(x))}{\forall x (\neg F(x))} \]

\[ \frac{\neg G}{\neg \neg G} \]

\[ \frac{\neg \neg G}{\neg E} \]

\[ \frac{\neg (G \rightarrow H)}{\neg E} \]

\[ \frac{\neg (G \rightarrow H)}{\neg E} \]

\[ \frac{\exists x (G(x))}{\forall x F(x)} \]

\[ \frac{\forall x (F(x))}{\exists x G(x)} \]

\[ \frac{\forall x (F(x))}{\exists x G(x)} \]

\[ \frac{\neg (\exists x G(x))}{\forall x (\neg F(x))} \]

\[ \frac{\forall x (\neg F(x))}{\exists x G(x)} \]

\[ \frac{\neg (\exists x G(x))}{\forall x (\neg F(x))} \]

\[ \frac{\forall x (\neg F(x))}{\exists x G(x)} \]

rules for negative literals

\[ \frac{(A)}{\Sigma A} \]

\[ \frac{\#}{\neg \neg (A)} \]

\[ \frac{\neg \neg (A)}{\neg B} \]

\[ \frac{\#}{\neg \neg B} \]

\[ \frac{(A \neg B)}{\Sigma A_1 \Sigma A_2} \]

\[ \frac{G G}{A \Theta = B \Theta} \]

where:

\[ A \Theta = B \Theta \]

\[ x \quad \text{variable} \]

\[ \Theta \quad \text{substitution} \]

\[ A, B \quad \text{atomic formula} \]

\[ E, F \quad \text{assertion formula} \]

\[ G, H \quad \text{goal formula} \]

\[ \Sigma I \quad \text{introduction component} \]

\[ \Sigma E \quad \text{elimination component} \]

\[ \Sigma A \quad \text{(partial) solution} \]

Figure 4.25: extended rules for negation
The direct computational interpretation of any of the rules (b)–(d), as a kind of introduction rule for any formula whatever, suffers from serious combinatorial problems. Also, a literal implementation of alternative (a), that is the presence of all formulae of the form \( F \lor \neg F \) as axioms, is not possible.

An incomplete implementation relying on recognising occurrences of complementary subgoal literals is suggested by consideration of the RGR rule of [Nilsson 80]. Well known equivalences of classical logic, enable any formula to be rewritten so that the negation operator applies only to atoms. This operation of moving in negations is part of the process for translating formulae into clausal form for resolution. These equivalences may be included in our system as rules of inference, as shown in figure 4.25. As a result of this extension the absurdity reasoning called for by negation introduction is confined to atoms, as indicated by the special \( \neg \) rule in the figure. The special MX rule may be applied whenever two complementary, unifiable atomic goals arise in distinct case arguments.

4.9 Alternative Languages

As seen above, classical and even intuitionistic natural deduction proof theories suffer from severe combinatorial problems. We place the blame for this on the following two features of these systems:

- The complexity of the search for contradictions.
- The complexity of applying excluded middle.

The Prolog family of logic programming languages avoids these problems. Negated goals but not assertions are admitted, removing contradictions altogether. Also, these languages are commonly not expressive enough to require application of excluded middle. For example the deduction problem \{ \} ?– \((a \supset b) \lor (b \supset a)\), discussed in the preceding
section, cannot be expressed. Many proposed extensions of these languages, for example [Gabbay & Reyle 84], are based on intuitionistic logic, again avoiding the need for excluded middle.

Prolog has adopted the negation as failure (NAF) rule of inference [Clark 78] as the proof theoretic device for negated goals. The NAF rule of inference fits neatly, as a negation introduction rule, into a natural deduction framework. According to this rule, the deduction problem $\Delta \vdash \neg G$ receives an affirmative answer given that there is a failure demonstration, denoted by $\Sigma_f$ in figure 4.27, for the problem $\Delta \vdash G$.

$$
\Sigma_f \\
\frac{G}{\neg G}
$$

Figure 4.27: negation as failure rule (NAF)

An alternative to the Prolog approach is to still admit negated assertions, but to limit the search for contradictions. This can be done by rejecting the absurdity principle. The rejection involves removing the absurdity rule and requiring that other rules discharge assumptions. Conceived on the basis of philosophical objections to classical logic, the so called relevant (relevance) logics follow this scheme. The systems of [Anderson & Belnap 75] reject the absurdity rule, but retain excluded middle. The intuitionistic relevant logic of [Tennant 87] rejects both principles.
Chapter 5

Inference Engines

The search for solutions is viewed within two paradigms:

- deduction employing derived rules of inference
- AND/OR graph search

The derived rules of inference paradigm provides a simple conceptual model of the inference engine. AND/OR graph search, on the other hand, exposes issues relevant to efficient search. The impressive implementation technology of the logic programming language Prolog is examined. The extension of this technology to more expressive languages and full AND/OR parallel evaluation is considered.

5.1 AND/OR Graphs and Derived Rules of Inference

We began chapter 3 with a brief examination of search strategies in the context of the natural deduction rules of inference. The fact that this set of rules is systematic and fixed (for a given logic) enabled us to eliminate them, and replace each axiom and query by its inferential extension. In chapter 4 we saw that it is possible to think of the AND/OR graph, that is an inferential extension, as representing a set of derived rules of inference. Consequently, we now have two further perspectives on the search task:

**Deduction employing derived rules of inference:** Subgraphs of inferential extensions correspond to derived rules of inference. Hence, search can be viewed in the context of reasoning within a deduction system consisting of a set of such derived rules.

**AND/OR graph search:** The search space for solutions consists of AND/OR graph fragments (renamed search components) connected by \texttt{cut} rule instances. A solution to the deduction problem at hand is a solution subgraph of the AND/OR search space.
A derived rule of inference is a generalisation of the notion of inference rule, as presented in chapter 2. The natural deduction rules are all instances of the schema shown in figure 5.1 (a). Such a rule leads from a set of premiss formulae \( \{F_1 \ldots F_n\} \) to a single formula \( G \) as conclusion. The rule may also discharge assumption formulae \( E_1 \ldots E_n \). In contrast, a derived rule, figure 5.1 (b), leads from a set of premiss formulae \( \{A_1 \ldots A_n\} \) to a set of conclusion formulae \( \{B_1 \ldots B_m\} \). The set of conclusions being read disjunctively. Further, the assumptions that may be discharged are not formulae but sets of derived rules \( R_1 \ldots R_n \).

\[
\begin{array}{c}
E_1 \\
\vdots \\
F_1 \ldots F_n \\
G \\
\end{array}
\quad (a)
\quad \begin{array}{c}
R_1 \\
\vdots \\
A_1 \ldots A_n \\
B_1 \ldots B_m \\
\end{array}
\quad (b)
\]

Figure 5.1: inference rule schemas

The set of derived rules produced by extend depends on the particular deduction problem statement at hand. Further, as the search proceeds, the set of available rules varies depending on subgoal and case argument context. The best we can do, a priori, is to distinguish the three subclasses of derived rules illustrated in figure 5.2. Rules drawn from each of these subclasses occupies a distinct niche in a complete solution:

\[
\begin{array}{c}
R_1 \\
\vdots \\
A_1 \ldots A_n \\
\end{array}
\quad (a)
\quad \begin{array}{c}
R_1 \\
\vdots \\
A_1 \ldots A_n \\
B_1 \ldots B_m \\
\end{array}
\quad (b)
\quad \begin{array}{c}
B_1 \ldots B_m \\
\end{array}
\quad (c)
\]

Figure 5.2: subclasses of derived rules

(a): A query rule has an empty conclusion, and is derived from the query formula. All paths in a solution are terminated at the bottom by a query rule instance.

(b): A proper rule is derived from an axiom or query that contains implications or negations as assertion subformulae.

(c): A fact rule has an empty set of premises, and may be derived from an axiom or query. All paths of a solution are terminated at the top by a fact rule instance.
Deduction systems, such as the above, that include extended forms of inference rules have received some attention recently. [Shoesmith & Smiley 78] investigate the extension to rules with multiple conclusions. [Schroeder-Heister 84] argues that rules that discharge other rules as assumptions are a "natural extension of natural deduction".

The account of implementation techniques in this chapter relies on both the derived rules and AND/OR graph search paradigms. The AND/OR graph view is strong for many issues in search and representation. The derived rule view comes into its own when we wish to present a simple user view of the inference engine, see chapter 6.

5.2 Search as a Constraint Satisfaction Problem

We take the definition of ANF solution graph of section 3.4.2 as a starting point for the exploration of implementation issues. That definition consisted of the following five constraints on the form of a solution graph:

- Query Relevance
- Axiom Relevance
- Resolved Choice
- Substitution Consistency
- Loop Freeness

As a first step towards computational realization we read this definition as a constraint satisfaction problem. The remainder of this chapter deals with the problem of applying these constraints constructively to the task of finding solutions.

Traditionally inference engines apply either forward chaining (axiom relevance) or backward chaining (query relevance) elaborating a single partial solution at a time (resolved choice) while maintaining full substitution consistency for the current partial solution. The loop freeness constraint is often ignored. The Prolog inference engine, to be described a little later, conforms to these conventions, and provides us with a good point of reference. The following subsections examine the five constraints in more detail.

5.2.1 Axiom and Query Relevance

Much of the discussion in this chapter will assume a backward chaining search strategy. The factors that speak in favour of this approach are briefly these:
• As illustrated in section 3.2, the query relevance constraint is built into backward chaining strategies. Only those subgraphs of the search space containing the query are explored.

• In cases where the search space is too large to permit exhaustive, uninformed search, the generic backward chaining scheme can be specialised to incorporate control knowledge. Backward chaining is conceptualised as simple goal reduction with choice of subgoal and rule. This point is expanded in section 6.2.

• Backward chaining simplifies some implementation issues. For example the construction of complete choice points is possible, as the path to the query is always known. Implementation techniques developed in the logic programming context may be applied.

Backward chaining does not apply the axiom relevance constraint to limit search. This problem is painfully obvious in the case of relevant deduction problems when the conclusion is a contradiction, as is the case with reductio ad absurdum. Haridi [Haridi 81] suggests that forward chaining should be adopted for these kinds of subproblems. Note, however, that a forward chaining strategy does not make effective use of query relevance. What is called for is a search regime that is sensitive to all available constraints. A first step in this direction would seem to be to apply the relatively expensive substitution consistency constraint incrementally. For some suggestions in this direction see the work [Sickel 76] on clause interconnection graphs.

5.2.2 Resolved Choice

A spectrum of search strategies from depth first to breadth first is characterized by the number of partial solutions maintained at any one time. Common practice is to simplify implementation and maintain resolved choice by choosing the extreme depth first end of the spectrum. The full breadth first strategy, at the other extreme, is often impractical on combinatorial grounds. In section 5.5.3 we consider the implementation of strategies in the middle ground, enabling the concurrent exploration of a number of promising partial solutions. The following paragraphs set the scene in the context of backward chaining search strategies.

The resolved choice constraint requires the selection of a single CUT instance from each choice point. The term “backward chaining” refers to an abstract search process, leaving unspecified the order in which either choice points or CUT rule instances within them are to be selected. In the preceding sentence we have distinguished two kinds of choice:
**Subgoal Choice:** Select a subset of the subgoals for expansion from the current set of partial solutions. In other words, given a set of partial solutions, select a set of choice points. For sequential, one solution at a time implementations both of these sets are singletons.

**Rule Choice:** Given a subgoal, select the derived rules to be applied from the set of candidate rules. In other words, given a choice point, select a set of `cut` rule instances.

The above conceptual model of the choices faced by a search algorithm is commonly found in accounts of the Prolog language. Prolog relies on the programmer to exploit his understanding of a fixed choice algorithm to control the search process. In section 6.2, encouraged by the success of this approach, we explore an alternative control paradigm based on the same conceptual model.

### 5.2.3 Substitution Consistency

A simple extension of the *composition of substitutions* operation of [van Vaalen 75] replaces a set of mgu's in solved form (see section 4.2.2) \( \{\Theta_1, \Theta_2, \ldots, \Theta_m\} \) with a single mgu in solved form \( \Theta \), such that for any term \( t \)

\[
((t\Theta_1)\Theta_2)\ldots\Theta_m \quad = \quad t\Theta
\]

**Composition of MGUs:** Given a set of mgu's \( \{\Theta_1, \Theta_2, \ldots, \Theta_m\} \) in solved form the following algorithm finds their composition if one exists and otherwise halts with fail status.

**step 1:** Let \( \Theta \) be \( \Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_m \). That is \( \Theta \) is a set of equality assertions \( \{X_1 = t_1, X_2 = t_2, \ldots, X_n = t_n\} \).

In order to reduce \( \Theta \) into solved form repeat step 2 until no longer applicable.

**step 2:** Choose any pair of equality assertions \( (X_i = t_i) \) and \( (X_j = t_j) \), such that the two parameters \( X_i \) and \( X_j \) are identical. If the two terms \( t_i \) and \( t_j \) unify then replace the two assertions in \( \Theta \) by the set of assertions that is the mgu of \( t_i \) and \( t_j \), otherwise halt with failure status.

Return \( \Theta \) as the answer.

Notice that the above operation is associative, implying that there is no restriction on the order in which unifiers from a solution graph are composed. The operation is also incremental, in the sense that applications of step 2 of the above algorithm may
be postponed. These freedoms are often not exploited by implementors to improve performance. Composition of unifiers is typically the most expensive operation of an inference engine implementation.

As discussed in chapter 4, the quantifier rules $\forall i$ and $\exists E$ impose two further constraints on substitutions:

Skolem Constraint: This condition can be maintained by restricting Skolem parameters to appear only on the right hand side of mgu equality assertions.

Dependency Constraint: To maintain this condition it is necessary to check that any $\forall i$ rule occurrences do not rely on undischarged assumptions. A very simple implementation can check the well formedness of a complete candidate solution.

5.2.4 Loop Freeness

The normal form for natural deduction proofs constrains the form of subproof for major premisses of elimination rules only. This leaves the door open for paths through minor premisses containing multiple occurrences of the same subproblem. In the case of a “no assumptions” language like Prolog a subgoal that is identical or subsumes a subgoal lower down on the path to the query signals the presence of a loop. In the case that assumptions are present, a sufficient additional condition for a loop is that the upper subproblem not have more premisses available to it than does the lower one.

![Figure 5.3: proof loops](image)

(a)  

(b)  

The potential for loops exists when occurrences of the elimination rules with minor premisses, that is $\exists E$ and $\sim E$ are present. Figure 5.3 illustrates the simplest loop elements. Notice that in the case of $\sim E$, the absurdity rule $\# X$ is also involved. The first step in minimizing the cost of loop detection is to perform a loop check only when multiple instances of components containing applications of these rules are present.
CHAPTER 5. INFERENCE ENGINES

5.3 Implementation Technology

The purpose of this section is to establish a point of reference for inference engine implementation technology. Programs for solving deduction problems have been developed in a number of distinct settings:

- Resolution refutation theorem provers.
- Question answering systems for deductive databases.
- Logic programming language implementations.
- Expert systems inference engines.
- Verification of the correctness of programs and hardware designs.

Despite the various demands of the intended area of application, many current systems are based on a set of common elements:

- Backward chaining search is used to maintain query relevance. Some systems incorporate a carefully limited forward chaining preprocessor.

- A single partial solution is explored at any one time, with backtracking on failure or depth bound. An important exception is the processing of ground atoms by database operations.

- Substitution consistency is maintained by a unification algorithm together with clever representation schemes for terms instantiated by unification. Substitution consistency is fully maintained for the single current partial solution.

Conforming to the above model, the logic programming community has largely focused its attention on the development of implementation techniques for the Horn language. First, recall that Horn clauses are mapped into Horn rules by extend. Second, recall that case arguments are derivations consisting of instances of Horn rules only. These two relationships suggest that we adopt this work as a point of reference. The remaining sections of this chapter explore implementation issues from this perspective.

5.3.1 The Prolog Inference Engine

The following paragraphs point out the key elements of current logic programming systems implementations. We focus on the structure sharing implementation technology developed for the Prolog language. This section deals with the the pure Horn language only. This discussion is used as a starting point for a subsequent investigation
of implementation techniques for more expressive languages. For more comprehensive descriptions of current techniques see for example [Aït-Kaci 90], [Bruynooghe 82] and [Campbell 84].

Prolog exploits the reading of sets of formulae as programming language procedure definitions. This *procedural semantics* of Prolog determines that the AND/OR search space be explored sequentially in left to right and depth first order. The stack based representation of computation state, developed for procedural programming language implementations, is used.

As an illustration, the partial solution of figure 5.4 (a) is represented by the data structures in (b). These data structures can be divided into a static and dynamic component:

**Structure Store:** The static structure of the Horn rules is kept here.

**Stack:** The structure and substitutions for the current (partial) solution are maintained as a sequence of stack frames.

The current (partial) solution tree is mapped onto the stack in chronological order. There is one stack frame for each occurrence of a Horn rule instance in the solution being represented. A stack frame consists of a pointer to the Horn rule structure, a pointer to a "parent" stack frame, a vector of bindings, as well as other information left out of the figure for clarity. The vector of bindings contains one entry for each distinct parameter occurring in the rule. The result of applying this vector as a substitution to the rule structure is the desired rule instance.

Each binding, an equality assertion of the form $X_i = t$, is represented by a binding vector entry. The renaming $i$ is implicit in the context of the binding as part of a stack frame. The name $X$ is associated with the vector offset. The term $t$ is explicitly represented as a pair of pointers, one pointing to the term structure, the other to the stack frame where bindings for parameters occurring in the term structure are to be found.

The fact that only a single partial solution is represented at any one time implies that just a single set of bindings, being the composition of unifiers for the current partial solution, is required. The composition of unifiers is represented by the entire set of non-null bindings on the stack. In the case of the example of figure 5.4, the composition of the three unifiers $\Theta_1, \Theta_2$ and $\Theta_3$ is represented by four binding vectors.

The above representation of the composition of unifiers has more structure than our textual representation as a set of equality assertions. The binding pointers form a graph consisting of a set of connected components. Each connected component corresponds to an equivalence class of terms. A pair of equivalence classes is unified by connecting
two components. That is, the Prolog unifier applies the well known UNION-FIND algorithm (see for example [Aho, Hopcroft & Ullman 74] for a description of union-find). For future reference note that the unifier is free to apply the path compression optimization of UNION-FIND.

Figure 5.4 is a simplification. The structure of the current partial solution is represented, while the information required for conducting the search is left out. This information consists of choice points and a trail. Two kinds of choice points are recorded on the stack:

**Rule Choice:** This corresponds to the cut choice point of our AND/OR graph model. Just a single pointer is needed to step through the sequence of rules.

**Subgoal Choice:** The antecedent of a Horn clause is a flat sequence of atoms. A single pointer is again sufficient to maintain the state of the left to right traversal of this sequence.

The trail is a chronological record of binding operations, consulted when undoing bindings on backtracking. The trail also fits in with the stack discipline.

Apart from the simplicity of the above scheme, the compilation of unification and
choice points, together with clever indexing schemes, contribute to the efficiency of Prolog implementations.

5.4 Extended Logic Programming

A logic programming language consists of two sublanguages, as recommended by the slogan

Algorithm = Logic + Control

of [Kowalski 79a]. We shall refer to these two component languages as the problem language and the control language. In this section we consider the prospect of extending the expressive power of these two languages as well as the issue of exploiting parallel hardware for solving deduction problems.

Let us first look at current proposals for extending the Prolog inference engine to deal with larger subsets of the full first order language as the problem language. It is instructive to consider goal and assertion syntax separately:

**Goal Syntax:** [Gabbay & Reyle 84] and [Bollen 88] have demonstrated implementations incorporating the implication introduction rule. Implementations of the full positive goal syntax, based on the transformations of [Lloyd & Topor 84], also exist [Thom & Zobel 88]. Even negated goals, implemented by the negation as failure mechanism, may be regarded as intuitionistic negation with respect to a completed program [Clark 78], [Shepherdson 88].

**Assertion Syntax:** Recall that the only operators admitted by the Prolog assertion syntax are $\forall$ and $\exists$. The author knows of no implementations that extend the inference engine to deal directly with enriched assertion syntax. Several meta interpreters have been proposed for asserted disjunctions and negations, see for example [Smith & Loveland 88].

In terms of the language hierarchy of chapter 4, direct implementations have not reached beyond the positive definite language $\mathcal{P}$. We suggest that the reason why logic programming systems do not offer the expressive power of the full assertion syntax are at least threefold:

**Procedural Semantics:** It is not clear what the procedural semantics should be once inferential extensions have more than a single atomic conclusion. Also, the procedural semantics of Prolog dictate a statically ordered backward chaining search, which is very inefficient in the case of relevant deduction subproblems.
Negation As Failure: Both disjunctive and existentially quantified assertions tend
to "block" negation as failure results. The semantics of a language incorporating
these constructs is not clear.

Proof Theory: No resolution refutation proof theories are known for languages
intermediate between the Horn language and full first order classical logic. The
natural deduction formulation now informs us what proofs in these languages
look like.

Concerning the expressive power of the control language: The procedural semantics
of Prolog determine a backward chaining, depth first, single solution search strategy.
The knowledgeable programmer escapes these restrictions using meta programming
techniques. That is, the control language is expressively inadequate for many application
areas. Again, the procedural semantics blocks extension of expressive power.

Recently, the exploitation of parallel computing hardware has become a major focus
for logic programming research, see for instance [Gregory 87], [Kacsuk 90] and
[Wise 86]. The procedural semantics of Prolog enable the efficient implementation of
the language on sequential machines. While some parallelism is available within this
model, a fuller exploitation of parallelism cannot tolerate a sequential execution model.

Perhaps it is obvious by now that we propose an approach to logic programming,
that does not rely on the procedural reading of formulae. Our aim is to unblock
the development of more expressive languages and implementations that can exploit
full AND/OR parallelism. The prospect of an efficient inference engine based on the
AND/OR graph paradigm is explored in the next section. Top level language design
issues are taken up in the next chapter.

5.5 Implementation Techniques for Extended Languages

The following subsections present refinements of the Prolog data structure to support
implementation of more expressive and parallel languages. The AND/OR graph representation
for solutions and search spaces suggests data structures for implementation
of extended languages.

5.5.1 More Expressive Sequential Languages

We saw in chapter 4 that the increased expressive power of a language is reflected
as an increase in the complexity of the inferential extensions of input formulae. The
preceding discussion of Prolog implementation techniques assumed the Horn language,
and consequently dealt with the corresponding simple Horn rule form of inferential
extensions only. The question we address in this section is this: Can we extend the
stack based scheme, with each stack frame representing a derived rule of inference, for the more expressive languages?

The generalisation of the implementation to the Edinburgh language is very simple. The only point of change concerns the subgoal choice pointer. For a Horn rule this pointer traverses left to right through a sequence of atoms. For the Edinburgh language this is generalised to a left to right, depth first enumeration of the solution trees of an AND/OR tree with atoms as leaves.

The next step up in complexity of inferential extensions is the presence of assumption search components. Recall that assumption components are generated by implications and logical negations as goals (the \( \exists I \) and \( \sim I \) rules of inference), present in the positive definite language. The set of inferential extensions available for the construction of a solution to a subproblem depends now on the context in which the subproblem occurs.

The necessary extension of data structures is illustrated in figure 5.5. A context pointer is allocated in each stack frame. A context being a set of assumption search components, each sharing some of their bindings with the stack frame that created the assumption. The representation of an assumption component in the structure store includes a list of common parameters the component shares with other components of the inferential extension. This information is needed to initialize binding vectors for the component's stack frame. Contexts are stored in a tree structured data base.

The most general form of search component contains an AND/OR tree of conclusions, as well as premisses. A stack based inference engine can still be used to construct the case arguments from which a solution is built. A case argument supervisor issues a
sequence of calls to the inference engine, each resulting in either a case argument being returned or in failure.

The data structure illustrated in figure 5.5 is adequate for representing a case argument. The context mechanism, however, needs to be extended to provide a wider range of services. A non-empty initial context may be supplied to the inference engine to direct the search for a case argument. As well as a context of assumptions, a case argument is associated with an “anti-context” of assumptions. The anti-context is the set of assumptions not available in the world of the case argument.

5.5.2 Single Solution AND Parallelism

While appropriate for sequential implementations, a stack based representation cannot be maintained when a set of asynchronous processes co-operate to build up a solution. If we abandon the stack discipline, we arrive at the data structure displayed in figure 5.6. This data structure is a graph, maintained in a heap store, perhaps distributed across a number of processors. This AND graph data structure is also appropriate for implementations designed to avoid unnecessary recomputation of subgoals on backtracking.

Notice that this data structure still represents just a single (partial) solution, and therefore can only support single solution AND parallelism. It is common in the literature to distinguish two forms of single solution AND parallelism:

**Restricted AND Parallelism:** This form occurs when the partial solution graph is extended concurrently at a number of subgoals that do not share variables.
Stream AND Parallelism: This form occurs when concurrent subgoals share variables.

The distinction shows up in our AND graph model in two ways: Firstly, concurrent access to shared bindings must be controlled to maintain the integrity of the composition of unifiers operation. Secondly, a bindings dependency analysis is required to determine the consequences of a failure on concurrent subgoals. For a more detailed discussion of AND parallelism see [Gregory 87].

5.5.3 Multiple Solutions AND/OR Parallelism

A further refinement of the data structure is required to represent multiple solutions. Let us suppose that the premisses $A_1^1$ and $A_2^1$, of the example in the preceding section, also unify with $R5$ and $R6$, as shown in figure 5.7 (a). Depending on the success of the composition of unifiers operation, there are from zero to four well formed partial solutions here. The data structure, shown in (b), represents the four candidate solutions.

The reader may have noticed already that, unlike in the simple motivational presentation of chapter 1, bindings are not associated with unifiers but with derived rule occurrences. The advantage of the current scheme is that the binding for any specified parameter is readily located at a fixed vector offset.

When multiple partial solutions are represented, a number of binding vectors may be associated with a single derived rule occurrence. That is, a rule occurrence in this data structure may stand for a number of distinct substitution instances of the rule.

The data structure of figure 5.7 stands for a set of candidate solutions. We need a second level representation to pick out the well formed (substitution consistent and loop free) solutions from among these candidates. The representation we propose here is a refinement of the ATMS labelling scheme introduced in chapter 1. The graph structure of a solution is represented explicitly by a label, while the composition of substitutions is not, as explained below.

The graph structure of a solution is uniquely determined by its set of binding vectors. Further, only the ambiguity of multiple binding vectors for the one subgoal needs to be resolved. For the example of figure 5.7, the structures of the candidate solutions are picked out by the labels shown in figure 5.8. Only labels corresponding to the well formed (partial) solutions are to be kept. Any label for a (partial) solution containing an inconsistent set of bindings or a loop (a nogood) is removed. Search effort should not be wasted on those portions of the AND/OR data structure that do not appear in any label. Many implementations would garbage collect such structures.

The set of non-null bindings associated with a solution can no longer be maintained
as the composition of substitutions in solved form, since a number of bindings may exist for the one parameter. Also, the path compression optimization cannot always be applied. Two options for implementing the composition of substitutions are:

- Call on the unifier to determine the composed binding for a parameter dynamically. In this case, the degree to which the set of bindings approximates solved form is critical to performance.
• Maintain an explicit representation of the composition with the label. This may well be feasible when restricted to a critical subset of parameters.

The above description of AND/OR parallel evaluation omits discussing mechanisms to support backtracking search. The reason for this omission is that the author's experimental implementation work has focussed on the non-backtracking language described in the introduction. It has also assumed that only Horn rules are present. A comprehensive description of feasible implementation techniques for more expressive parallel languages has to wait on further experimental work.
Chapter 6

Exploiting the Representation

A natural deduction solution can be readily understood as an argument leading from a set of axioms, by way of simple principles of deduction, to the query. Atomic normal form extends this explanatory power of natural deduction. The very detailed steps of reasoning are replaced by derived rules of inference, each justified by a particular input formula. This perspicuous representation can be exploited as follows:

- As a graphic display, it may be used for purposes of explanation, testing and debugging.

- Reflected as a theory accessible to introspection, it may be used for purposes of control and to meet other practical demands placed on the reasoner.

6.1 Visualization

Having presented a mechanical reasoner with a deduction problem $\Delta ? \neg G$ and a finite amount of time for computation, we expect to receive as the answer a set (possibly empty) of proofs, together with an indication of whether this set contains all the proofs there are. Each of the proofs is to be a solution for the given deduction problem, that is they are proofs of $\Gamma \vdash G$ (where: $\Gamma \subseteq \Delta$).

In this setting, we can think of a proof as explaining which subset of the axioms, and by what methods of reasoning, lead to the conclusion $G$. The following two subsections treat explanation in this sense only. The aim is to display a solution in such a way that it can readily be grasped as an explanation. The third subsection extends this treatment to the display of partial solutions, for the purposes of testing and debugging. The aim being to observe the progress being made in covering the search space.
6.1.1 Flat Explanation

Given that some primitive principles of reasoning, their representation as rules of inference and the instantiation and composition of these rules are understood and accepted, a proof in any formal system may claim to explain its conclusion. In addition, the natural deduction rules claim to represent principles actually used when the most detailed account of an argument is presented by a human mathematician. Why don’t we just present the user with the ANF natural deduction proof as explanation?

A weakness of the ANF form of natural deduction is illustrated in figure 6.1. The ANF solution for the deduction problem \( \{a \land b\} \vdash a \land b \) is shown in (a), whereas the very simple solution in (b) is clearly better as an explanation. The atomization transformations (Lemma 3 in Chapter 2) can be applied in reverse to remove such unnecessary elimination-introduction pairs for any of the connectives.

![Figure 6.1: atomization example](image)

The person reading the explanation is likely to be familiar with many sound rules of inference, which need to be derived when using the natural deduction rules. For example, the commutativity result, established in figure 6.2, may be displayed as in (b). Such transformations for the commutative and associative operators \( \land \) and \( \lor \) can significantly simplify the presentation of a solution.

![Figure 6.2: commutativity example](image)

In chapter 1 resolution proofs were criticized on the grounds that refutations are not as perspicuous as direct proofs. Yet, the negation introduction rule (reductio ad absurdum) calls on a kind of refutation for the proof of a negated goal. Some applications of reductio may be removed by transformations. For example, the solution in figure 6.3 (a) may be simplified into the form shown in (b), being a single application.
CHAPTER 6. EXPLOITING THE REPRESENTATION

of modus tollens. More generally however, the reductio rule remains.

\[
\begin{array}{c}
\text{AXIOM} \quad a \supset b \\ \sim b \\
\hline
\sim b \\
\end{array}
\quad \text{---(1)}
\begin{array}{c}
\text{AXIOM} \quad \sim a \\
\hline
\sim a \\
\end{array}
\]

Figure 6.3: reductio example

In the presence of disjunctive assertions, a solution consists of a set of case arguments for the query. For such solutions, the case arguments may be presented separately. Recall, from chapter 4, that some of the cases may lead to absurdity, requiring a terminal application of the absurdity rule. The example of figure 6.4 (a) illustrates this complication. As in this example, some applications of or elimination may be presented as disjunctive syllogism, as shown in (b).

\[
\begin{array}{c}
\text{AXIOM} \quad a \lor b \\
\sim a \\
\hline
\sim a \\
\end{array}
\quad \text{---(1)}
\begin{array}{c}
\text{AXIOM} \quad a \lor b \\
\sim a \\
\hline
\sim a \\
\end{array}
\]

Figure 6.4: proof by cases example

Even after the above simplifications, natural deduction solutions for all but the most trivial problems are too large and detailed to have much more than a curiosity value. In the next subsection we exploit the notions of solution fragment and derived rule of inference to improve the situation.

6.1.2 Structured Explanation

It is possible to partition any given ANF solution into a set of fragments, each justified by a particular axiom or query. Alternatively, the solution can be seen as being composed of applications of derived rules of inference, again justified by a particular input formula. For these reasons, the ANF scheme can also be characterized as input form natural deduction. We now claim that this feature of the ANF scheme extends the explanatory power of natural deduction.

The structure of ANF solutions, as a composition of derived rules of inference or so-
Figure 6.5: $\Delta_{\text{alps}}$ — example problem theory

The mapping of formulae into derived rules of inference for the Horn language is a very simple one. Many current systems exploit this mapping implicitly for their explanation facilities. What are the issues raised by the more expressive languages?

Consider the problem theory $\Delta_{\text{alps}}$, for the so called alpinist puzzle\(^1\), of figure 6.5. As examples of structured explanations, figure 6.6 (a) and (b) offer solutions to the two deduction problems

$$\Delta_{\text{alps}} \vdash \exists z \sim \text{skier}(z)$$

and

$$\Delta_{\text{alps}} \vdash \exists z \text{ climber}(z)$$

respectively. Each derived rule instance is displayed here as an inference stroke annotated with the input formula that justifies it. In the event that the derived rule involves assumptions (contains applications of $\exists I$ or $\sim A$), both the inference stroke and the assumption are annotated with a unique number. In (a) the formula $\text{skier}(\text{mike})$ is such an assumption. The solution in (b) consists of two case arguments.

For existentially quantified queries it is often important that we be able to extract from the solution the so called answer substitution. For natural deduction solutions the answer substitution is simply a set of pairs, each pair $\langle x, t \rangle$, extracted from an occurrence of the existential introduction rule:

$$\begin{align*}
\frac{G(t)}{\exists x G(x)}
\end{align*}$$

\(^1\)This puzzle appeared in the comp.lang.prolog group of the internet news distribution.
As illustrated by the examples in Figure 6.6, the substitution is not always easily spotted in the structured display. For both examples, the answer substitution is just \{ (z, mike) \}. Where the solution consists of case arguments, the answer substitution may differ between arguments. The answer substitution may even be absent, as in (b), where a case argument terminates in an application of the absurdity rule.
6.1.3 Testing and Debugging

In the event that an unexpected solution, caused by an erroneous axiomatization of the problem, is found, an explanation display can reveal the error. Where the solution is not found within acceptable time or fails to be found altogether, displays of the search space and partial solutions can be useful. The following discussion is limited to these issues only. For a thorough treatment of testing and debugging, in the logic programming context, see [Shapiro 83].

\[
\begin{align*}
\forall v \forall w \text{edge}(v, w) & \supset \text{path}(v, w) \\
\forall x \forall y \forall z \text{path}(x, y) \land \text{path}(y, z) & \supset \text{path}(x, z)
\end{align*}
\]

(a)

(b) [Diagram of edge(V, W) → path(V, W)]

(c) [Diagram of path(X, Y) → path(X, Z)]

Figure 6.7: Δ\text{path} — path axioms

We will use the axiomatization Δ\text{path}, shown in figure 6.7 (a), to illustrate the discussion in the remainder of this chapter. The \text{path}/2 predicate is intended to be interpreted as path in a directed graph. The example problems may also be read at the meta level — think of the directed graph as representing a search space for solutions. The two rules of inference derived from the axioms are shown in (b) and (c). In this section we diagnose a number of problems in applying these derived rules to solve problems. In section 6.2 we express meta knowledge needed to apply the rules more intelligently.

The search space generated by the two derived rules is illustrated by the connection graph\(^2\) display of figure 6.8. Circuits in this figure represent recursive application of rules. Solutions are obtained by creating fresh renaming instances of nodes, “unrolling” such circuits. For the example problem the warning is clear — Rules need to be applied carefully to avoid wasted computation on \text{path}/2 subgoals, due to violation of the loop freeness constraint.

The normal form for natural deduction solutions does not prevent the construction

\(^2\)Although the connection graph paradigm was developed by [Kowalski 75] for the resolution refutation proof theory, many of the ideas are equally applicable here. We go no further in this direction than to point out that this kind of display can be very useful in analysing the computational characteristics of a set of rules.
of multiple solutions for the one query relying on identical premisses. A set of axioms $\Delta_{\text{chain}}$, and its intended model, for use in conjunction with $\Delta_{\text{path}}$ are shown in figure 6.9. Given the deduction problem

$$\Delta_{\text{path}} \cup \Delta_{\text{chain}} \Rightarrow \text{path}(a, d)$$

two solutions, as shown in figure 6.10 (a) and (b), are possible. Once we notice that these two solutions rely on the same set of axioms, we are likely to be disappointed by this state of affairs. We will suggest remedies in section 6.2.

$$\begin{cases} \text{edge}(a, b) \\ \text{edge}(b, c) \\ \text{edge}(c, d) \end{cases}$$

Figure 6.9: $\Delta_{\text{chain}}$ — simple directed graph

Even when the inference engine enforces the loop freeness constraint, as best it can, solutions may fail to appear when expected. Two kinds of failure are commonly distinguished:

**Finite Failure:** An inference engine can, at least in principle, complete the search in finite time without finding any solutions.

**Non Termination:** The search does not terminate within any finite interval of time.

From a pragmatic point of view we distinguish two kinds of finite failure:

**Constructed Non-demonstrability:** The inference engine completes the search within acceptable time without finding any solutions.
CHAPTER 6. EXPLOITING THE REPRESENTATION

Inefficient Search: Solutions are not returned within acceptable time, although in principle the search completes in finite time.

As an example of constructed non-demonstrability consider the annotated fragment of search space, shown in figure 6.11, for the query $\text{path}(b, a)$, given the set of axioms $\Delta_{\text{path}} \cup \Delta_{\text{chain}}$. There are three partial solutions here, each of which incorporates, as a leaf, a goal atom $A$ that is either:
No Match: No derived rule of inference has a conclusion that matches $A$.

Loop: $A$ subsumes another goal that occurs on the path from $A$ to the query.

Notice again that care is required in selecting which goal to expand. Expansion of either of the two "OPEN" goals in the figure is wasted effort.

![Diagram](attachment://figure_6_12.png)

Both solutions and finite failure demonstrations represent the final state of a computation. In the presence of either inefficient search or non-termination we need to understand the progress of the computation in covering the search space. Consider the path problem for the tree form directed graph shown in figure 6.12 (a). The partial solution shown in (b) displays an intermediate state for a search that traverses the tree left to right bottom up. Clearly a more efficient regime for this problem would traverse down the tree, and if possible in parallel starting from the two end points.

### 6.2 Introspection

As well as the nominated purposes, the discussion in the first half of this chapter was intended to support the claim that search spaces, solutions and perhaps even the process of search can be readily conceptualized and understood. In the remainder we argue that such conceptualization and understanding can be harnessed to solve many of the problems that arise in practical applications of computational logic.
6.2.1 An Extended Introspective Architecture

Chapter 1 introduced the idea of introspection, and its application to the task of controlling the selection of subgoals on behalf of the object level inference engine. The application was described as a two level architecture, being:

**Object Level:** An axiomatization of the object problem domain used by an object level inference engine to construct solutions in response to deduction problems.

**Meta Level:** An axiomatization of the choice of subgoal problem used by a meta level inference engine to choose a subgoal in response to a query from the object level.

Figure 6.13 illustrates an extension of the introspective architecture, of chapter 1. This extended architecture is designed to allow control of performance critical activities by meta language assertions, as well as enabling the exploitation of parallel hardware. The current state of the search is maintained as an AND/OR graph in a blackboard [Waterman & Hayes-Roth 78] memory, accessible to inspection and change by a number of agents. A parallel implementation will need to support concurrent access to the blackboard.

For the subsequent discussion of this model, we once again rely on Prolog as a point of reference. A Prolog program is more than just a set of assertions about the problem
domain. The assertions are organized as a set of procedures, each procedure consisting of a sequence of statements. A Prolog statement is more than just a logical formula. A statement is an expression constructed recursively as an operator applied to a sequence of expressions. Some of the operators may be read as logical connectives, others have only a procedural reading. Some primitive expressions may be read as atomic formulae of either the object or meta language with the remainder again having just a procedural reading.

The introspective architecture attempts to build on the successes of Prolog, while addressing its shortcomings. In broad terms, the issues are these:

**Language:** In Prolog knowledge about the problem domain is expressed in logic, while the knowledge that directs subgoal and rule selection, search space pruning, input/output etc. is not. Also, the failure to separate knowledge about the various domains can make it difficult to understand, modify and reuse programs.

We suggest that knowledge about each distinct domain be regarded as a distinct theory. The problem domain theory being axiomatized in the object language, the multiple other theories in the meta language.

**Search:** The range of available search strategies in Prolog is limited. The statically determined search strategy is not sensitive to the instantiation of parameters and other runtime context. Recent implementations feature a range of “meta predicates” in an effort to overcome this limitation. Also, search is restricted to a single solution at a time with chronological backtracking on failure.

In principle, the introspective architecture suffers from none of these limitations. The identification of practical alternatives is, however, a challenging problem. As a starting point for such an investigation, we can retain backward chaining and adopt the Prolog “meta predicates” as part of the meta language.

**Parallelism:** The sequential procedural semantics of a Prolog program locks away parallelism. Many attempts have, however, been mounted in an effort to identify useful non-sequential operational readings.

In contrast the introspective architecture of figure 6.13 suggests a parallel implementation, based on a set of co-operating processes.

Figure 6.13 represents just one intermediate point in a range of possible architectures for introspective computation. There is no a priori assignment of functionality between the object and meta levels. At the one extreme, every action is encoded as meta level
assertions. At the other extreme, every action is performed by a monolithic object level inference engine. The former extreme gives total control of every action to the meta level assertions, while the latter provides none. While very efficient implementation techniques are known for object level engines, expressive power at the meta level is bought at a relatively high cost in computation speed. Experience with theorem proving and logic programming systems suggests a compromise, where the following issues are addressed by meta language assertions:

- Ordering the Search
- Detecting Loops
- Pruning the Search Space
- Negation As Failure
- Allocating Computational Resources
- Exploiting Models
- Specifying Communication

These issues are discussed in a little more detail in the remaining sections. We focus on meta language assertions of the form:

\[
\text{condition } \supset \text{ action}
\]

Recall from chapter 1, that the condition is tested by introspecting the current computation state, while the action reflects down, specifying a computation to be performed. In the examples that follow, a condition is expressed in terms of a goal/2 predicate, which picks out the atomic goal formula nodes in the current computation state. Some of the condition and action predicates are borrowed from the Prolog language, the remainder being proposed new constructs.

6.2.2 Ordering the Search

For any real machine the speed of computation is limited, the size of partial solutions is bounded by available storage and the number of concurrent operations is bounded by the number of available processors. These limitations imply that the order in which the search space is explored is often crucial to performance. This order may be specified declaratively by assertions in the meta language.

Often we have only limited knowledge (perhaps none) to bring to bear on the problem of deciding which, of a number of available expansions of the AND/OR graph,
to pursue next. For a computation state for which the user supplied theory is mute, the choice may be determined by a default theory. If it turns out that the defaults lead to difficulties, the user supplied theory may be incrementally strengthened. We can go further and recognise a number of useful knowledge sources:

**Catchall:** A simple, uniform search strategy enables one to reliably predict the effects of overriding assertions. Prolog's left to right, depth first choice order is an example of such a catchall theory for a sequential implementation.

**Static:** Search advice computed from the static structure of the problem (analysis of connection graph for instance), can reduce the amount of overriding user supplied knowledge required for acceptable performance.

**Dynamic:** Search decisions may depend on an analysis of the dynamic behaviour of the system. Such "learned" strategies may further reduce the amount of user intervention.

**User:** The user may be in possession of knowledge about the intended interpretation of the problem axioms and the range of queries likely to be encountered. This knowledge may be put to use as search advice.

The kind of default theory determines, to a large extent, the kind of overriding assertions needed. For example, a depth first strategy is easily trapped by infinite branches, while space can quickly become a problem for a breadth first strategy.

As an illustration of the formulation of search advice, consider the path axioms $\Delta_{\text{path}}$ of figure 6.7. For this problem breadth first search is a reasonable catchall theory. A static analysis of the problem can reveal that the $edge/2$ relation is defined entirely by a set of atomic axioms. We may thus regard $edge/2$ goals as relatively tractable, and specify that they be selected whenever they occur. The syntax for this piece of advice might be:

$$\forall g \forall n_1 \forall n_2 \text{goal}(g, edge(n_1, n_2)) \supset \text{select}(g)$$

If we know that $\Delta_{\text{path}}$ is to be applied to the kind of tree shown in figure 6.12 (a), we may specify preferential selection of $path/2$ goals that have an instantiated second argument.

$$\forall g \forall n_1 \forall n_2 \text{goal}(g, path(n_1, n_2)) \land \text{ground}(n_2) \supset \text{select}(g)$$

Many current logic programming languages provide constructs to suppress the selection of a goal atom that contains non-ground terms. The preferential selection of goals, illustrated above, is not available in any of the languages known to the author.
6.2.3 Detecting Loops

Advice about the conditions under which loops may occur and the frequency of checks may be specified by meta language assertions. The knowledge sources for a loop detection strategy may be diverse:

**Catchall:** A simple strategy is to check for a loop in the event that a predetermined depth bound is exceeded, or even, as a last resort, when memory space is exhausted.

**Static:** An analysis of circuits and the associated substitutions in the connection graph can identify potential loops.

**User:** The user may wish to override decisions derived from the above sources.

For the example \( \Delta_{path} \) axioms, \( path/2 \) goals that do not have both arguments ground can recur as part of a loop. We might make use of this knowledge by specifying that loop checking be performed for such goals. The assertion might look like this:

\[
\forall g \forall n_1 \forall n_2 \text{goal}(g,\text{path}(n_1, n_2)) \land (\text{var}(n_1) \lor \text{var}(n_2)) \supset \text{loopcheck}(g)
\]

We propose that controlled loop detection, as illustrated above, be incorporated into logic programming systems.

6.2.4 Pruning the Search Space

Large portions of search space can often be removed by careful application of knowledge about the problem axiomatization and the current state of search. In Prolog such pruning is effected by use of the ! (cut) and once constructs. We suggest that pruning be specified by assertions in the meta language.

Where a computationally expensive subproblem occurs more than once, an opportunity exists to reduce the size of the search space by sharing results. Whenever a new subproblem arises, two knowledge sources may be consulted:

**Introspect:** In the event that two identical subproblems are concurrently represented, results may be fully shared. Partial sharing may be possible when one of the subproblems subsumes the other.

**Memorize:** In the normal course of events, subproblems fail due to the no match or loop conditions, as was illustrated in figure 6.11. The failure of a subgoal implies the failure of any partial solution that incorporates that subgoal. In terms of the AND/OR graph paradigm, the failure propagates to siblings and the parent node at AND nodes and to the parent node at exhausted OR nodes.
The space taken up by these data structures is normally reclaimed, making them inaccessible to introspection. The retention of crucial failure results may be specified by meta language assertions.

The checking of every new subgoal against these knowledge sources is likely to be infeasible. The user may identify subgoals to be checked by declarations in the meta language.

An opportunity for pruning the search space exists when duplicate solutions, like the ones illustrated in figure 6.10, occur. This situation is commonly referred to as don't care nondeterminism in the logic programming literature. For our path example, we might phrase the request for a single solution like this:

$$\forall g \forall n_1 \forall n_2 \text{goal}(g, \text{path}(n_1, n_2)) \land \text{ground}(n_1) \land \text{ground}(n_2) \supset \text{once}(g)$$

More generally, we can provide constructs for the arbitrary pruning of choice points. For the example path problem we may confine the search to proceed as a sequential left-to-right edge following search thus:

$$\forall g_1 \forall g_2 \forall n_1 \forall n_2 \forall n_3 \text{goal}(g_1, \text{path}(n_1, n_2)) \land \text{goal}(g_2, \text{path}(n_2, n_3)) \supset \text{chop}(g_1, \text{path}/2)$$

The chop/2 construct here is a generalization of the Prolog ! (cut). In this case any element of goal $g_1$'s choice point that would reduce the goal to a further path/2 subgoal is removed.

Ideally, each meta language assertion that specifies pruning of the search space can be read as a theorem about proof search for the intended problem domain. Failure on this point results in the loss of solutions.

### 6.2.5 Negation As Failure

Our knowledge about a problem axiomatization $\Delta$ may include the fact that it is complete for a particular predicate $p/n$. That is,

$$\Delta \vdash p(a_1, \ldots, a_n)$$

if and only if $p(a_1, \ldots, a_n)$ is true in the intended domain, and

$$\Delta \not\vdash p(a_1, \ldots, a_n)$$

if and only if $\neg p(a_1, \ldots, a_n)$ is true. In this case the negation as failure (NAF) rule of inference

$$\frac{\Delta \not\vdash p(a_1, \ldots, a_n)}{\Delta \vdash \neg p(a_1, \ldots, a_n)}$$
CHAPTER 6. EXPLOITING THE REPRESENTATION

is sound. Recall that in section 4.9 we suggested that the deduction system for the object language could be extended to include an inductive definition of the notion of a failure demonstration. We now propose that negation as failure reasoning be applied whenever it is sound, and reductio reasoning otherwise to answer negative goals.

The knowledge that the axiomatization of the example path/2 predicate is complete might be expressed in the meta language like this:

\[ \forall n_1 \forall n_2 \text{complete}(\text{path}(n_1, n_2)) \]

Within the first order language the knowledge of completeness of predicate p/n may be expressed by the syntactic transformation of completing the axiomatization for p/n. Clark [Clark 78] introduced this transformation for the Horn language. Although this work generalizes easily to the positive definite language, the extension to disjunctive assertions is more problematic. As an example of the completion transformation see the axiomatization \( \Delta_{\text{comp}} \) of figure 6.14, being the result of completing \( \Delta_{\text{path}} \cup \Delta_{\text{chain}} \). Arguably the completed axiomatization is less readable and modular than the original.

\[
\begin{align*}
\forall v \forall w \text{edge}(v, w) & \equiv ((v = a) \land (w = b)) \lor ((v = b) \land (w = c)) \lor ((v = c) \land (w = d)) \\
\forall x \forall y \forall z \text{path}(x, z) & \equiv \text{edge}(x, z) \lor (\text{path}(x, y) \land \text{path}(y, z))
\end{align*}
\]

Figure 6.14: \( \Delta_{\text{comp}} \): completion of \( \Delta_{\text{path}} \cup \Delta_{\text{chain}} \)

A solution for the query \( \sim \text{path}(b, a) \) for the completed axiomatization is shown in figure 6.15. The form of the solution, a set of case arguments embedded in an application of reductio ad absurdum, is the expected response for negated queries from completed axiomatizations. Recall that a backward chaining search strategy is not well suited to the task of finding such solutions.

Comparing the reductio solution with the finite failure demonstration of figure 6.11 we note that: The failure demonstration is simpler and therefore likely to be more readily understood as an explanation. Further, the failure demonstration appears as a subgraph of the reductio solution. We conjecture that this is the case generally, and that a procedure for translating failure demonstrations into reductio solutions is feasible.

6.2.6 Allocating Computational Resources

Meta language assertions may address the problem of allocating limited computational resources:

\textbf{Time:} The user may wish to impose a time limit on a computation, or perhaps specify a time dependent search strategy.
**CHAPTER 6. EXPLOITING THE REPRESENTATION**

**Space:** Once memory is exhausted, rollback may be specified for the less promising partial solutions. For distributed memory implementations, advice for memory allocation may be given.

**Processors:** Concurrent AND/OR search can exhibit genuine superlinear speedup. In practice worthwhile computational tasks need to be identified and allocated to the various processors with care.

As an example consider $\Delta_{\text{comp}}$ with an arbitrary directed graph. We may wish to specify concurrent search by two processes, working in from the two endpoints of any given goal path.

$$\forall g \forall n_1 \forall n_2 \text{goal}(g, \text{path}(n_1, n_2)) \land \text{ground}(n_1) \land \text{ground}(n_2) \supset$$

$$\exists p_1 \exists p_2 \text{process}(g, p_1) \land \text{strategy}(p_1, \text{LeftToRight}) \land$$

$$\text{process}(g, p_2) \land \text{strategy}(p_2, \text{RightToLeft})$$

The two search strategies *LeftToRight* and *RightToLeft* are simple variants of the edge following search illustrated in section 6.2.4 above. Once either of the processes reaches a decision, the other may be terminated on the grounds of duplication. This may be achieved by using the *once/1* construct, also discussed in section 6.2.4.

**6.2.7 Exploiting Models**

Models for a problem domain can be used to speed up computation in two ways:
CHAPTER 6. EXPLOITING THE REPRESENTATION

Counterexamples: In his pioneering work, [Gelernter 59] used diagrams as counterexamples for proposed theorems of geometry. This idea generalizes to models for any domain. Such testing against models may be specified in the meta language.

Procedural Attachment: Efficient algorithms are known for many problems. As an example, many arithmetic functions are commonly provided for directly in machine hardware. Such procedural attachment may be specified declaratively.

For the example path problem, it may be the case that even carefully controlled deduction does not yield acceptable performance. As a last resort, we can write a procedure, call it PathFinder, to decide these goals. We then specify a procedural attachment in the meta language.

\[ \forall g \forall n_1 \forall n_2 \text{goal}(g, \text{path}(n_1, n_2)) \land \text{ground}(n_1) \land \text{ground}(n_2) \supset \text{attach}(g, \text{PathFinder}) \]

6.2.8 Specifying Communication

The relationship between the computation state and any interaction with the system’s environment may be specified by assertions in a meta language. The facilities that may be provided include:

Read/Write: Prolog programmers have found it useful to embed various input and output requests in their programs. We can specify that a given input/output action take place once the computation state satisfies a given condition.

Debugging: The idea of declarative debugging can be realized in the introspective framework. A debugging action is specified to occur in response to the given condition being met by the current computation state.

Carelessness in pruning the search space may result in unexpected failures. In the case of our example path problem we can attempt to diagnose the problem thus:

\[ \forall g \forall n_1 \forall n_2 \text{goal}(g, \text{path}(n_1, n_2)) \land \text{ground}(n_1) \land \text{ground}(n_2) \land \text{failed}(g) \supset \text{display}(g) \]

The display/1 construct will generate a failure demonstration display, such as the one illustrated in figure 6.11.
Chapter 7

Conclusion

The formalization of the notion of a logically sound argument, as a natural deduction proof, offers the prospect of a computer program capable of constructing such arguments in response to queries. We have presented a constructive definition for a new subclass of natural deduction proofs, called atomic normal form (ANF) proofs. We have argued that this is the right framework for mechanical reasoning on both proof theoretic and computational grounds.

7.1 Proof Theory

ANF is a well motivated normal form for natural deduction. In chapter 2, we demonstrate that ANF proofs form a deductively complete subclass of the normal form proofs of [Prawitz 65]. In subsequent sections we propose that both these normal forms be strengthened as follows:

No Vacuous Applications of Inference Rules: Every occurrence of the existential elimination, or elimination and negation introduction rules must discharge assumptions. See sections 4.5.2, 4.5.3 and 4.6.1.

Absurdity Rule: The absurdity rule may only occur as the terminal rule application for:

- the entire deduction,
- a case argument (minor premiss of disjunction elimination),
- premiss of implication introduction

See section 4.7.

These remarks address the intuitionistic and classical systems. Some modifications are required for minimal logic and other subsystems.
**CHAPTER 7. CONCLUSION**

**Discharging Assumptions:** Assumption discharge is to occur as early in the proof as is permitted by the discharge constraints. See sections 4.4.1 and 4.5.2.

**Loop Free:** The proof must be loop free. See section 5.2.4.

These additional constraints do not affect what is deducible in the intuitionistic or classical systems. Further, any proof that does not observe these constraints, violates the claim:

"A deduction in normal form proceeds from the assumptions of the deduction to the conclusion in a direct and rather perspicuous way without detours" — [Prawitz 65] p 8.

We therefore submit that there is a need for a strong normal form for natural deduction, and that these constraints be incorporated.

In section 3.4.1 we propose that, for the purpose of deduction, an assertion or query formula be represented by its inferential extension. Further, each inferential extension may be read as a set of derived rules of inference. These derived rules take on an interesting form that incorporates the extensions of [Shoesmith & Smiley 78] and [Schroeder-Heister 84], as described in section 5.1.

The notion of constructed non-demonstrability, introduced in section 6.1.3, is an important contribution of logic programming research to proof theory. In section 6.2.5 we conjecture that failure demonstrations can be translated to reductio proofs.

### 7.2 Languages

A wide range of languages and logics are available as natural deduction systems. In chapter 4, we present a spectrum of subsystems of the classical first order calculus. For these systems the ANF formulation exhibits a simple tradeoff between the expressive power of the language in which a problem is expressed and the deductive machinery required to solve that problem. This analysis offers simple, natural deduction based accounts for many current logic programming languages. It also reveals the deductive machinery required for the implementation of more expressive logic programming languages.

In chapter 4 we also raise possible objections to the application of classical principles of reasoning in automated theorem proving:

**Excluded Middle:** The rejection of excluded middle distinguishes the intuitionistic from the classical reasoner.
**Absurdity Rule:** Rejection of the absurdity rule is required for coherent reasoning in the presence of contradictions.

The application of these principles can be computationally extremely expensive. This point is implicitly acknowledged by the many mechanical reasoners that fail to implement them. Much more research on computationally tractable logics in this neighbourhood is required.

### 7.3 Computation

ANF inference engines make use of well known computational techniques. We introduce the computation, as AND/OR graph search, in section 3.3.3. An alternative view of the computation, as deduction employing derived rules of inference, is presented in section 5.1. The fundamental operation of the ANF inference engine is the unification of two atomic formulae, see sections 4.2.2 and 5.2.3.

Chapter 5 investigates the application and extension of logic programming implementation technology for ANF inference engines. The application of truth maintenance techniques is developed in sections 1.8 and 5.5.3.

The exploitation of parallel computing hardware for logic programming is an area of much current research. The AND/OR graph search model is related to the popular AND/OR process model of [Conery 83]. Implementation data structures, based on the AND/OR graph search model are analysed in section 5.5.3. The introspective architecture, described in section 6.2, is designed to support parallel evaluation.

Our investigation is confined to the classic forward and backward chaining search strategies. As pointed out in section 5.2, such strategies do not constitute the best possible use of all the available search constraints. Relevant deduction problems, are particularly poorly served by these strategies. Work is needed to identify more appropriate search strategies for these problems.

The representation of arguments as natural deduction proofs provides a good foundation for research on efficiency gain by emulating human reasoning abilities. A characteristic of human reasoning in a particular domain is the incremental accumulation of reasoning expertise for problems in that domain. Two related aspects of this expertise are:

**Lemmas:** derived rules of inference, carefully selected for their expected utility in solving problems.

**Analogy:** the recognition of a class of problems which may be solved by the instantiation of a common proof schema.
7.4 Visualization

The visualization of proofs, failure demonstrations and search spaces is considered in section 6.1. A natural deduction proof can be understood as an argument that leads from a set of premisses, by way of simple rules of inference, to the conclusion of interest. ANF extends this explanatory power of natural deduction. The argument may be presented in terms of derived rules of inference, each justified by a particular input formula.

In section 6.2 we propose that control and other pragmatics be formulated as deduction problems at the meta level. An advantage of this approach is that the work on visualization can be carried over to explanation, testing and debugging of these meta level functions also.

Visualization of the process of search is discussed only very briefly. Much more work is needed to identify useful schemes here.

7.5 Introspection

We present an introspection based architecture for ANF inference engines, see sections 1.7 and 6.2. The architecture is aimed to exploit both parallel computing hardware and the perspicuous natural deduction representation of reasoning to overcome the combinatorial and other practical problems faced by computational logic applications. The model represents an extension of the schema [Kowalski 79*]

\[
\text{Algorithm} = \text{Logic} + \text{Control}
\]

The new schema looks something like this

\[
\text{Algorithm} = \text{Deduction Problem} + \\
\quad \text{Search Control} + \\
\quad \text{Resource Allocation} + \\
\quad \text{Computational Models} + \\
\quad \text{Input/Output}
\]

Each of the components on the right hand side represents a distinct logical theory. The deduction problem consists of a set of axioms and a query formula expressed in an object language. The remaining theories are expressed as sets of axioms in a meta language.

Our experiments have been confined to a Horn meta language and the simple conceptualization of the computation state as a frontier of atomic goals. The conceptualization and some of the meta predicates were borrowed from current logic programming
languages. Even within this restricted framework we were able to identify several useful new constructs.

The efficient implementation of meta level inference is crucial for achieving acceptable performance for implementations of the architecture. The introspective model is based on the notoriously expensive operations of pattern matching, associative recall and logical deduction. More work is needed to determine the practicality of this approach.
Bibliography

[Aho, Hopcroft & Ullman 74]

[Aït-Kaci 90]

[Anderson & Belnap 75]

[Apt & vanEmden 82]

[Bollen 88]

[Bowen & Kowalski 82]

[Boyer & Moore 72]

[Bruynooghe 82]
Maurice Bruynooghe. ‘The Memory Management of Prolog Implementations’.

[Campbell 84]  

[Chang & Lee 73]  

[Clark 78]  

[Clocksin & Mellish 81]  

[Conery 83]  

[Curry 77]  

[Davis76]  

[de Kleer 86]  

[Doyle 79]  

[Dummett 77]  
[Dyckhoff 91]

[Fuller & Abramsky 88]

[Gabbay & Reyle 84]

[Gallaire & Lasserre 82]

[Gelernter 59]

[Genesereth & Nilsson 87]

[Gentzen 35]

[Gregory 87]

[Gödel 31]

[Hallnäs & Schroeder-Heister 90]
[Haridi 81]

[Hayes 73]

[Hogger 84]

[Johansson 36]

[Kacsuk 90]

[Keronen 91]

[Kowalski 75]

[Kowalski 79a]

[Kowalski 79b]

[Lassez et al. 88]
BIBLIOGRAPHY

[Lloyd 84]

[Lloyd & Topor 84]

[Maes & Nardi 88]

[McCarthy & Hayes 69]

[Minker 88]

[Nakamura 84]

[Nilsson 80]

[Paulson 87]

[Pereira 84]

[Prawitz 65]

[Robinson 65]
BIBLIOGRAPHY

[Schroeder-Heister 84]

[Sergot 83]

[Shepherdson 88]

[Shapiro 83]

[Shoesmith & Smiley 78]

[Sickel 76]

[Skolem 28]

[Smith 86]

[Smith 89]

[Smith & Loveland 88]
BIBLIOGRAPHY

[Tennant 78]

[Tennant 87]

[Thom & Zobel 88]

[van Harmelen 88]

[van Vaalen 75]

[Waterman & Hayes-Roth 78]

[Weyhrauch 80]

[Whitehead & Russell 1910-1913]

[Wise 86]