Computational Natural Deduction

Seppo R. Keronen

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

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.

- 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 ($\wedge$I): The conjunction $G \wedge H$ follows given both a proof $\mathbf{\Pi}_1$ for the formula G and a proof $\mathbf{\Pi}_2$ for the formula H.

And elimination ($\wedge$E): Given a proof $\mathbf{\Pi}$ of the conjunction $E \wedge F$ the formula E follows. Also, from the same premiss the formula F follows.

Implication introduction ($\supset$I): The implication $F \supset G$ follows given a proof $\mathbf{\Pi}$ 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.

introduction rules

$$\text{—}{\scriptstyle \wedge\text{I}}\,\dfrac{\dfrac{\mathbf{\Pi}_1}{G} \quad \dfrac{\mathbf{\Pi}_2}{H}}{G \wedge H}$$

$$\text{—}{\scriptstyle \supset\text{I}}\,\dfrac{\dfrac{\overset{[F]}{\mathbf{\Pi}}}{G}}{F \supset G}$$

$$\text{—}{\scriptstyle \vee\text{I}}\,\dfrac{\dfrac{\mathbf{\Pi}}{G}}{G \vee H} \qquad \text{—}{\scriptstyle \vee\text{I}}\,\dfrac{\dfrac{\mathbf{\Pi}}{H}}{G \vee H}$$

$$\text{—}{\scriptstyle \sim\text{I}}\,\dfrac{\dfrac{\overset{[G]}{\mathbf{\Pi}}}{\#}}{\sim G}$$

elimination rules

$$\text{—}{\scriptstyle \wedge\text{E}}\,\dfrac{\dfrac{\mathbf{\Pi}}{E \wedge F}}{E} \qquad \text{—}{\scriptstyle \wedge\text{E}}\,\dfrac{\dfrac{\mathbf{\Pi}}{E \wedge F}}{F}$$

$$\text{—}{\scriptstyle \supset\text{E}}\,\dfrac{\dfrac{\mathbf{\Pi}}{G \supset F} \quad \dfrac{\mathbf{\Pi}_1}{G}}{F}$$

$$\text{—}{\scriptstyle \forall\text{E}}\,\dfrac{\dfrac{\mathbf{\Pi}}{\forall x F(x)}}{F(t)}$$

$$\text{—}{\scriptstyle \sim\text{E}}\,\dfrac{\dfrac{\mathbf{\Pi}}{\sim F} \quad \dfrac{\mathbf{\Pi}_1}{F}}{\#}$$

Figure 1.1: a selection of natural deduction rules

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

Or introduction ($\vee$I): A disjunction follows given a proof of either disjunct.

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

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

Negation elimination ($\sim$E): Given the two proofs, $\mathbf{\Pi}$ of $\sim F$ and $\mathbf{\Pi}_1$ 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.

$$
{\scriptstyle \sim\mathrm{I}}\,\dfrac{\dfrac{\overset{[G]}{\Pi}}{\#}}{\sim G}
\qquad\qquad
{\scriptstyle \sim\mathrm{E}}\,\dfrac{\dfrac{\Pi_1}{\sim F}\quad\dfrac{\Pi_2}{F}}{\#}
$$

Figure 1.2: example natural deduction system

Let us now focus on a particular *deduction problem*: Given the set of axioms $\{a, \sim b\}$ prove the conclusion $\sim\sim\sim\sim a$ using just the rules of inference shown in figure 1.2. In symbols:

$$\{a, \sim b\} \ \underset{1.2}{?\!-}\ \sim\sim\sim\sim a$$

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).

$$
\begin{array}{ccc}
\text{— AXIOM —} & \text{— AXIOM —} & \sim\sim\sim\sim a \\
a & \sim b & \text{— QUERY —} \\
 & & \\
\text{(a)} & \text{(b)} & \text{(c)}
\end{array}
$$

Figure 1.3: axioms and query

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[2]. 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:

$$\{a, \sim b\} \ \underset{1.2}{?\!-}\ \sim b$$

[2]Generalisation to directed acyclic graph will be considered in chapter 4 in connection with asserted disjunctions.

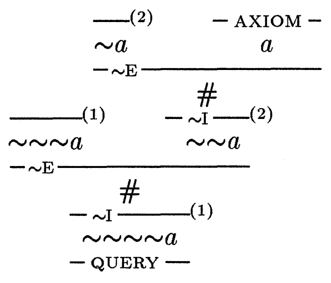


Figure 1.4: natural deduction proof

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.

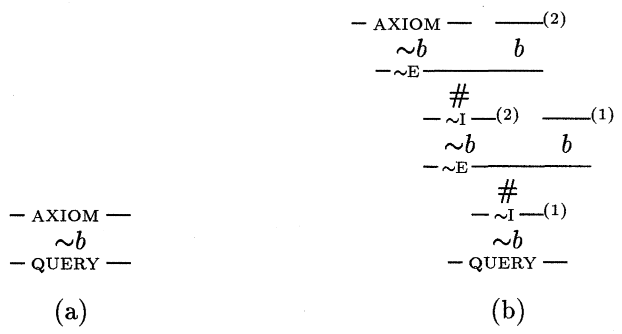


Figure 1.5: normal and abnormal proof

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 $\{a,\ \sim b\}$.

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 $not(atom(a))$.

$$\left\{\begin{array}{l} assertion(atom(a)) \wedge assertion(not(atom(b))) \\ \forall x\; assertion(atom(x)) \supset goal(atom(x)) \\ \forall y\; assertion(not(y)) \wedge goal(y) \supset assertion(atom(\#)) \\ \forall z\; (assertion(z) \supset goal(atom(\#))) \supset goal(not(z)) \end{array}\right\} \;?\!-\; \begin{array}{l} goal(atom(a)) \vee \\ goal(not(atom(a))) \end{array}$$

Figure 1.6: meta deduction problem

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 $\wedge$, $\vee$, $\supset$ and $\forall$, while the object language admits the $\sim$ operator only.

The intended interpretation of the predicate *goal*/1[3] is provability of the argument formula. The predicate *assertion*/1 stands for a restricted notion of provability, in terms of which *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 *assertion* and *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.

$$\begin{array}{l} assertion(atom(a)) \\ assertion(not(atom(b))) \\ \sim assertion(atom(X)) \vee goal(atom(X)) \\ \sim assertion(not(Y)) \vee \sim goal(Y) \vee assertion(atom(\#)) \\ assertion(Z) \vee goal(not(Z)) \\ \sim goal(atom(\#)) \vee goal(not(Z)) \\ \sim goal(atom(a)) \\ \sim goal(not(atom(a))) \end{array}$$

Figure 1.7: clausal form

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

[3]The notation *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.

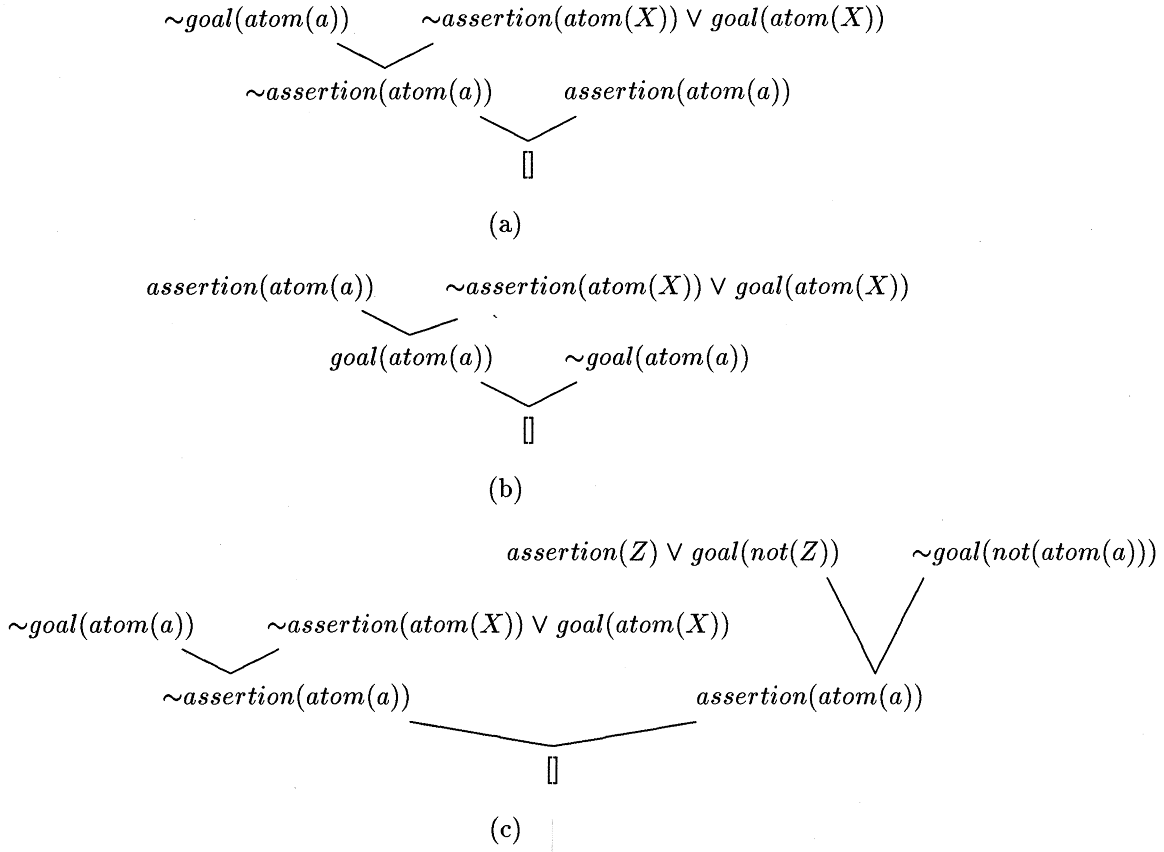


Figure 1.9: resolution refutations

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.

$$
{\scriptstyle\text{QUERY}}\dfrac{{\scriptstyle\vee\text{I}}\dfrac{{\scriptstyle =\text{CUT}}\dfrac{{\scriptstyle\supset\text{E}}\dfrac{{\scriptstyle\forall\text{E}}\dfrac{{\scriptstyle\text{AXIOM}}\dfrac{}{\forall x\ assertion(atom(x)) \supset goal(atom(x))}}{assertion(atom(X_1)) \supset goal(atom(X_1))} \qquad {\scriptstyle =\text{CUT}}\dfrac{{\scriptstyle\wedge\text{E}}\dfrac{{\scriptstyle\text{AXIOM}}\dfrac{}{assertion(atom(a)) \wedge assertion(not(atom(b)))}}{assertion(atom(a))}}{assertion(atom(X_1))}\ X_1{=}a}{goal(atom(X_1))}}{goal(atom(a))}\ X_1{=}a}{goal(atom(a)) \vee goal(not(atom(a)))}}{}
$$

(a)

$$
{\scriptstyle\text{QUERY}}\dfrac{{\scriptstyle\vee\text{I}}\dfrac{{\scriptstyle\supset\text{E}}\dfrac{{\scriptstyle\forall\text{E}}\dfrac{{\scriptstyle\text{AXIOM}}\dfrac{}{\forall x\ assertion(atom(x)) \supset goal(atom(x))}}{assertion(atom(a)) \supset goal(atom(a))} \qquad {\scriptstyle\wedge\text{E}}\dfrac{{\scriptstyle\text{AXIOM}}\dfrac{}{assertion(atom(a)) \wedge assertion(not(atom(b)))}}{assertion(atom(a))}}{goal(atom(a))}}{goal(atom(a)) \vee goal(not(atom(a)))}}{}
$$

(b)

Figure 1.10: ANF 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 premiss 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.

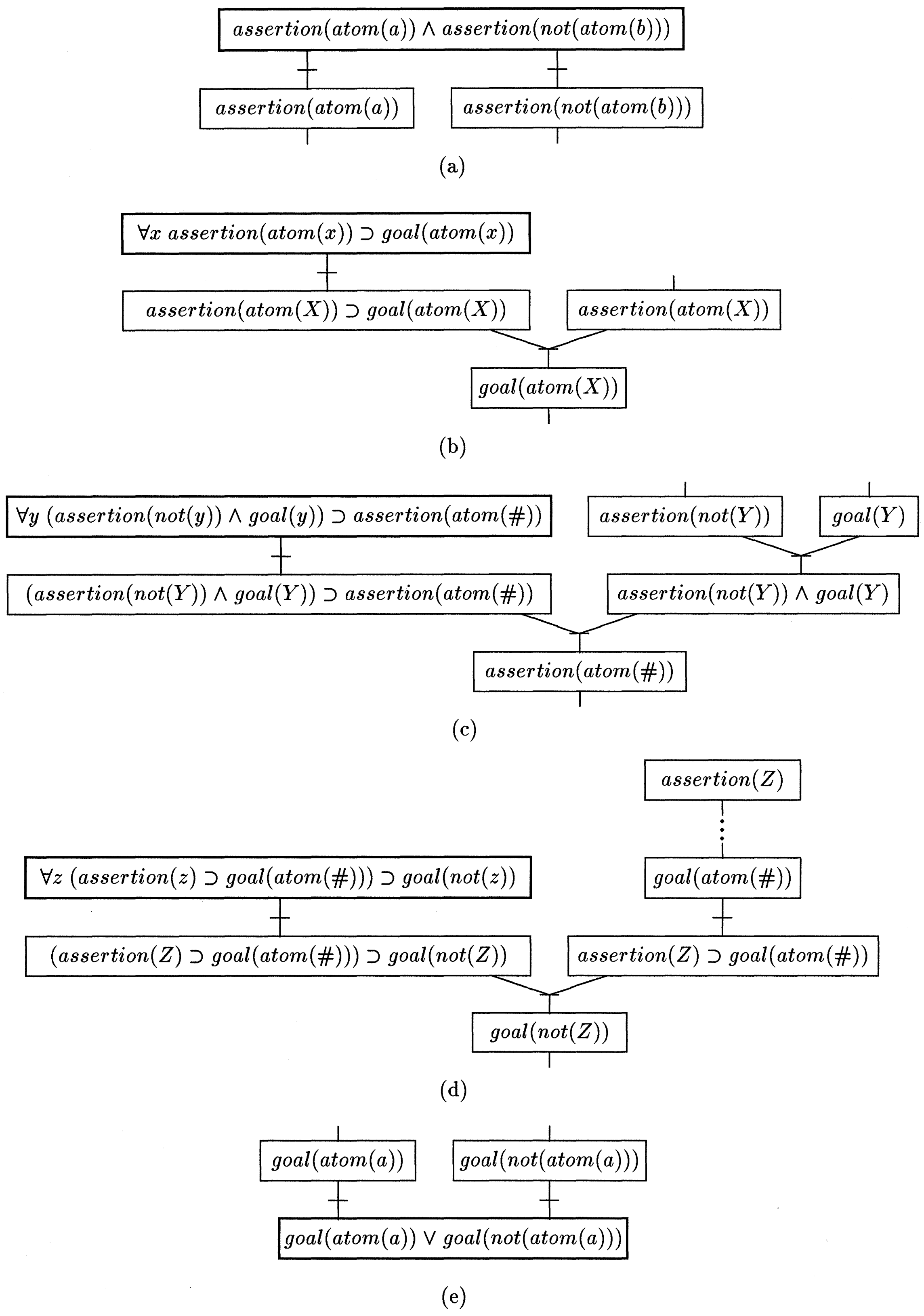

assertion(atom(a)) ∧ assertion(not(atom(b)))
assertion(atom(a))
assertion(not(atom(b)))
(a)
∀x assertion(atom(x)) ⊃ goal(atom(x))
assertion(atom(X)) ⊃ goal(atom(X))
assertion(atom(X))
goal(atom(X))
(b)
∀y (assertion(not(y)) ∧ goal(y)) ⊃ assertion(atom(#))
assertion(not(Y))
goal(Y)
(assertion(not(Y)) ∧ goal(Y)) ⊃ assertion(atom(#))
assertion(not(Y)) ∧ goal(Y)
assertion(atom(#))
(c)
assertion(Z)
∀z (assertion(z) ⊃ goal(atom(#))) ⊃ goal(not(z))
goal(atom(#))
(assertion(Z) ⊃ goal(atom(#))) ⊃ goal(not(Z))
assertion(Z) ⊃ goal(atom(#))
goal(not(Z))
(d)
goal(atom(a))
goal(not(atom(a)))
goal(atom(a)) ∨ goal(not(atom(a)))
(e)

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.

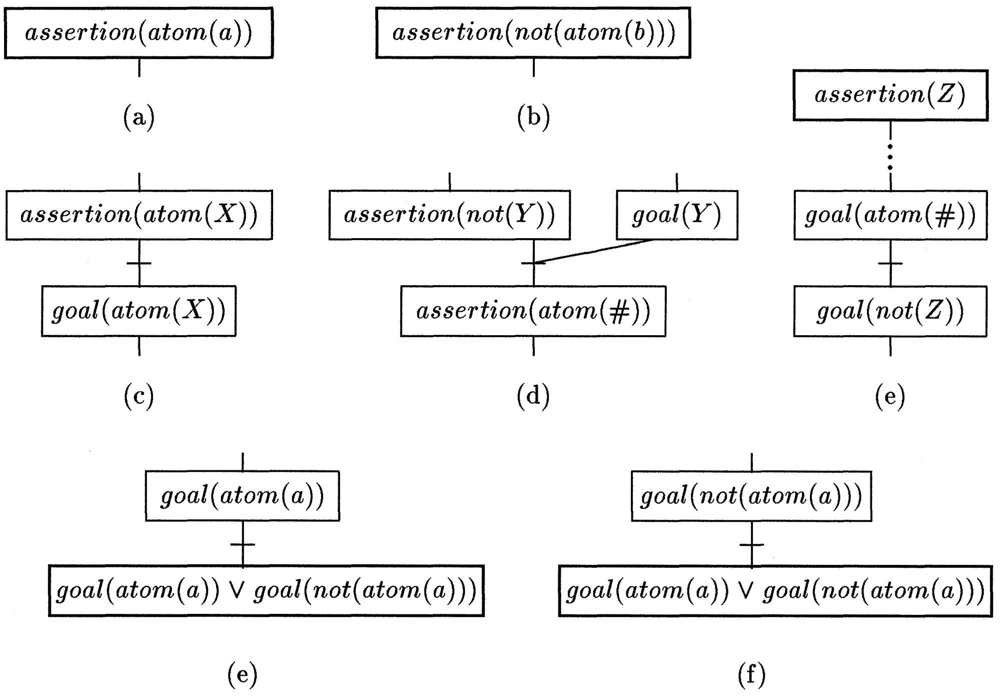


Figure 1.12: derived rules of inference

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:

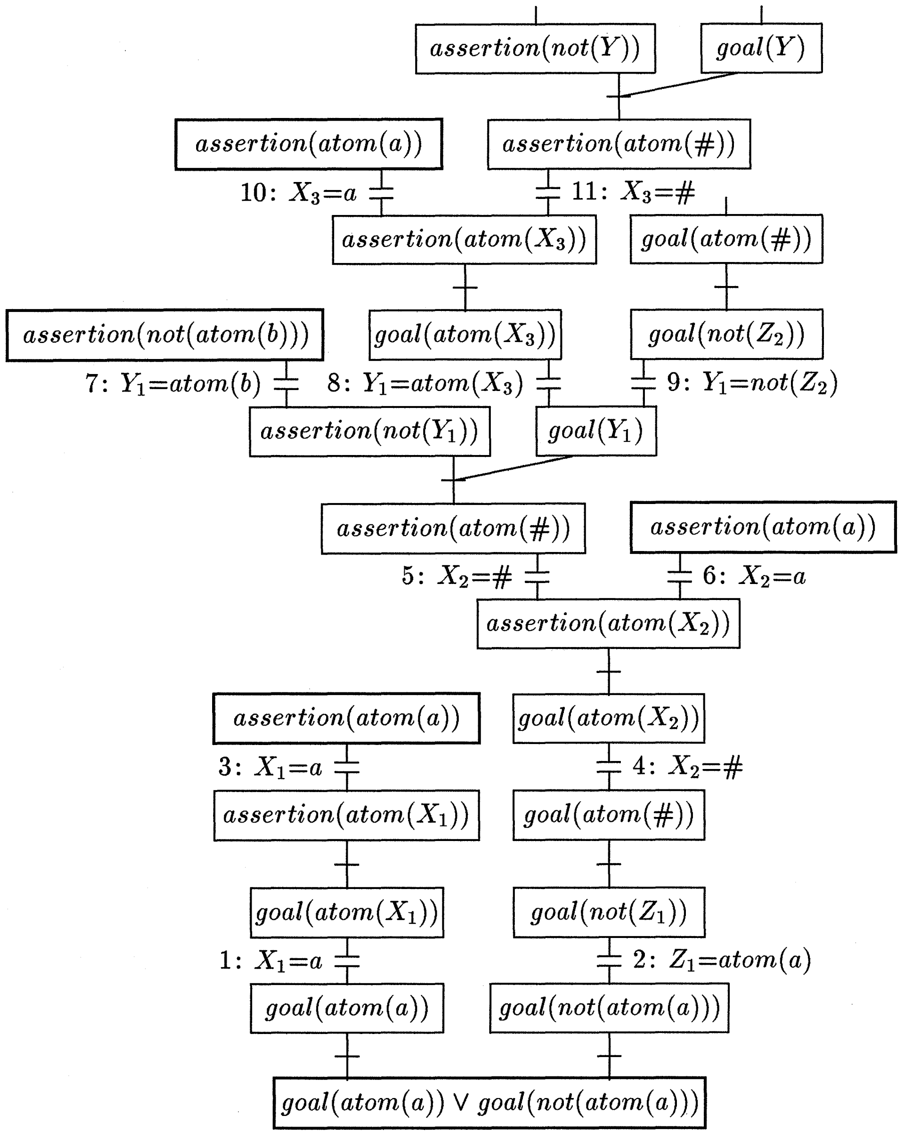


Figure 1.13: search space

- 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.

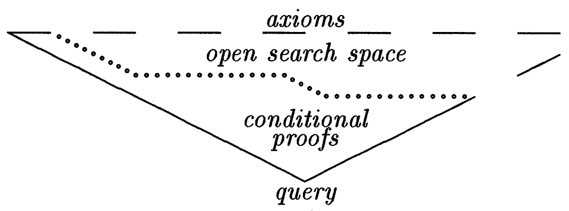


Figure 1.14: backward chaining search

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, recognised 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.

$$\forall\, A_1 \wedge A_2 \wedge \ldots \wedge A_n \supset B \quad \xrightarrow{\text{extend}} \quad \frac{A_1' \quad A_2' \quad \ldots \quad A_n'}{B'}$$

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[6] — 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 ($\wedge$, $\vee$, $\supset$, $\forall$, $\exists$ and $\sim$) 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 $goal(F) \vee {\sim} goal(F)$ would receive a positive answer even

[6] apart from the trail

in the case that F 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 79^a] 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.

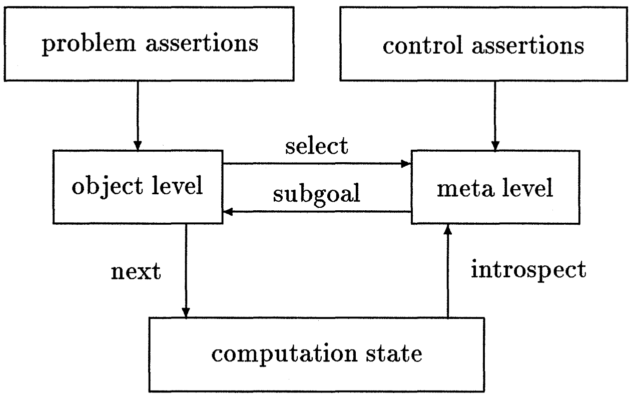


Figure 1.16: a simple introspective architecture

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.

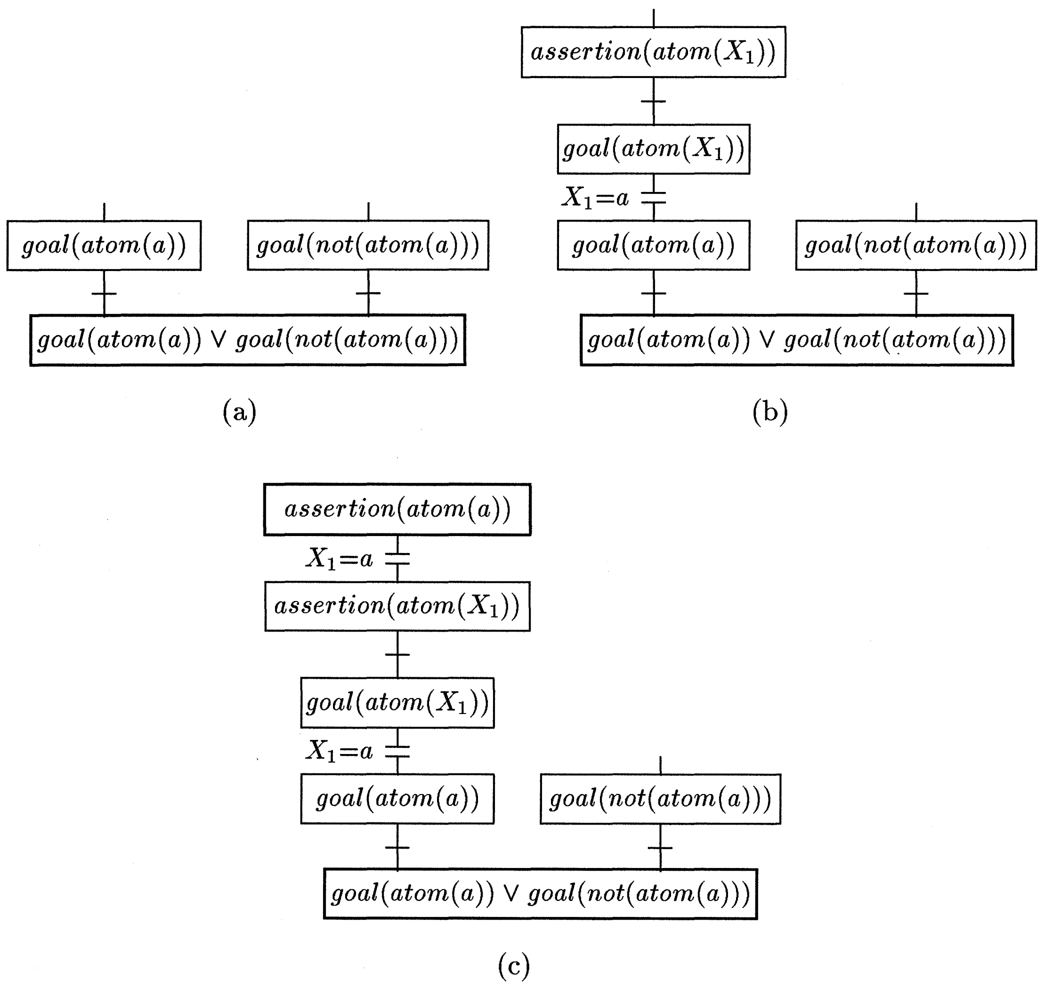


Figure 1.17: computation state

$$open(fnode(5), assertion(atom(a)))$$
$$open(fnode(3), goal(not(atom(a))))$$

Figure 1.18: computation state axioms

- 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 *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 `search/1` of figure 1.19. The single argument of `search/1` represents a computation state. Only the two procedures `final/1` (termination condition) and `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.

```
search(State)  :-  final(State),
                   display(State).

search(State)  :-  not final(State),
                   select(State,Subgoal),
                   next(State,Subgoal,NextState),
                   search(NextState).
```

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 *final*, ∼*final*, *select*(*Subgoal*) and ∼*select*(*Subgoal*). The *final*/0 and *select*/1 predicates of the control theory are reflected down as the `final/1` and `select/2` procedures of the inference engine respectively. The closed world assumption applies to *final*/0 but not *select*/1.

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).

$$\forall n\, \forall f\ open(n, assertion(f)) \supset select(n).$$
$$\sim\!(\exists n\, \exists f\ open(n, assertion(f))) \supset (\forall n\, \forall f\ open(n, f) \supset select(n)).$$

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

$$\{1, 3\}$$

$$\{2, \underbrace{4, 6}_{nogood}\}$$

$$\{2, 4, 5, \underbrace{7, 8, 10}_{nogood}\}$$

$$\{2, 4, 5, \underbrace{7, 8, 11}_{nogood}\}$$

$$\{2, 4, 5, \underbrace{7, 9}_{nogood}\}$$

Figure 1.21: labels

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: $\wedge$, $\vee$, $\supset$ and $\sim$

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 (…, x, y, z) for variables.

parameters — a denumerable supply of free variables. We will use the uppercase counterparts of variables (…, 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, ...) 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 $E \wedge F$, $E \vee F$ and $E \supset F$.
- 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 $\forall x F'$ and $\exists x F'$ 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.

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 $\boldsymbol{\Theta}$ we write $F\boldsymbol{\Theta}$. For example:

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

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

2.2 Proof

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

Proof: The inductive definition of proof goes like this:

base An ocurrence 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 $\{F1, F2, \ldots, Fn\}$ is summarized by the deducibility assertion;

$$\{F1, F2, \ldots, Fn\} \vdash G$$

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.

$$\begin{array}{ccccc}
\dfrac{\mathbf{\Pi}}{G} & \begin{matrix}(F)\\ \mathbf{\Pi}\end{matrix} & \begin{matrix}\dfrac{\mathbf{\Pi}_1}{(F)}\\ \mathbf{\Pi}_2\end{matrix} & \begin{matrix}[F]\\ \mathbf{\Pi}\end{matrix} & \begin{matrix}\overline{\;E\;}^{\,(i)}\\ \vdots\\ \dfrac{F}{G}\,(i)\end{matrix} \\
(a) & (b) & (c) & (d) & (e)
\end{array}$$

Figure 2.1: notation for proofs

Figure 2.1 introduces some notation for proofs:

(a) Proof $\mathbf{\Pi}$ has conclusion G.

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

(c) A proof $\mathbf{\Pi}_1$ with conclusion F, has been grafted onto proof $\mathbf{\Pi}_2$ at an occurrence of F as premiss.

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

$$\frac{G\supset F \quad G}{F} \qquad\qquad {}_{-\supset\mathrm{E}}\frac{\dfrac{\mathbf{\Pi}_1}{G\supset F} \quad \dfrac{\mathbf{\Pi}_2}{G}}{F} \qquad\qquad {}_{-\supset\mathrm{I}}\frac{\dfrac{\overset{[F]}{\mathbf{\Pi}}}{G}}{F\supset G}$$

$$\text{(a)} \qquad\qquad\qquad \text{(b)} \qquad\qquad\qquad \text{(c)}$$

Figure 2.2: inference rule notation

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 $\mathbf{\Pi}_1$, $\mathbf{\Pi}_2$ 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:

$$\text{If } \frac{\mathbf{\Pi}_1}{G\supset F} \text{ is a proof and } \frac{\mathbf{\Pi}_2}{G} \text{ is a proof then } {}_{-\supset\mathrm{E}}\frac{\dfrac{\mathbf{\Pi}_1}{G\supset F} \quad \dfrac{\mathbf{\Pi}_2}{G}}{F} \text{ is a proof.}$$

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 $\mathcal{C}_{\Pi}$ of figure 2.3, may be partitioned into four subsets:

Introduction Rules: The generic introduction rule, with premiss proofs $\mathbf{\Pi}_1 \ldots \mathbf{\Pi}_n$ and conclusion formula G, is expressed in the pattern:

$$_{-I}\frac{\mathbf{\Pi}_1 \;\ldots\; \mathbf{\Pi}_n}{G}$$

The premiss formulae are subformulae of the conclusion.

introduction rules

$${\scriptstyle \wedge\mathrm{I}}\ \dfrac{\dfrac{\Pi_1}{G} \quad \dfrac{\Pi_2}{H}}{G \wedge H}$$

$${\scriptstyle \vee\mathrm{I}}\ \dfrac{\dfrac{\Pi}{G}}{G \vee H} \qquad {\scriptstyle \vee\mathrm{I}}\ \dfrac{\dfrac{\Pi}{H}}{G \vee H}$$

$${\scriptstyle \supset\mathrm{I}}\ \dfrac{\dfrac{\overset{[F]}{\Pi}}{G}}{F \supset G}$$

$${\scriptstyle \forall\mathrm{I}}\ \dfrac{\dfrac{\Pi}{G(X)}}{\forall x G(x)}$$

$${\scriptstyle \exists\mathrm{I}}\ \dfrac{\dfrac{\Pi}{G(t)}}{\exists x G(x)}$$

$${\scriptstyle \sim\mathrm{I}}\ \dfrac{\dfrac{\overset{[G]}{\Pi}}{\#}}{\sim G}$$

elimination rules

$${\scriptstyle \wedge\mathrm{E}}\ \dfrac{\dfrac{\Pi}{E \wedge F}}{E} \qquad {\scriptstyle \wedge\mathrm{E}}\ \dfrac{\dfrac{\Pi}{E \wedge F}}{F}$$

$${\scriptstyle \vee\mathrm{E}}\ \dfrac{\dfrac{\Pi}{E \vee F} \quad \dfrac{\overset{[E]}{\Pi_1}}{G} \quad \dfrac{\overset{[F]}{\Pi_2}}{G}}{G}$$

$${\scriptstyle \supset\mathrm{E}}\ \dfrac{\dfrac{\Pi}{G \supset F} \quad \dfrac{\Pi_1}{G}}{F}$$

$${\scriptstyle \forall\mathrm{E}}\ \dfrac{\dfrac{\Pi}{\forall x F(x)}}{F(t)}$$

$${\scriptstyle \exists\mathrm{E}}\ \dfrac{\dfrac{\Pi}{\exists x F(x)} \quad \dfrac{\overset{[F(X)]}{\Pi_1}}{G}}{G}$$

$${\scriptstyle \sim\mathrm{E}}\ \dfrac{\dfrac{\Pi}{\sim F} \quad \dfrac{\Pi_1}{F}}{\#}$$

absurdity rule

$${\scriptstyle \#\mathrm{X}}\ \dfrac{\dfrac{\Pi}{\#}}{G}$$

excluded middle

$${\scriptstyle \mathrm{MX}}\ \dfrac{}{F \vee \sim F}$$

x	-	variable
X	-	parameter
t	-	term
E, F, G, H	-	formula
Π, Π_1, Π_2	-	proof

Figure 2.3: system $\mathcal{C}_\Pi$ — natural deduction proof for classical logic

Elimination Rules: The elimination rule with major premiss $\mathbf{\Pi}_0$, minor premisses $\mathbf{\Pi}_1 \ldots \mathbf{\Pi}_n$ and conclusion F is expressed in the pattern:

$${\scriptstyle -E}\frac{\dfrac{\mathbf{\Pi}_0}{E}\quad \mathbf{\Pi}_1\ \ldots\ \mathbf{\Pi}_n}{F}$$

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

$$\text{Given a proof } \frac{\mathbf{\Pi}_1}{F} \text{ and a proof } \begin{array}{c}(F)\\ \mathbf{\Pi}_2\end{array} \text{ then } {\scriptstyle =\text{CUT}=}\begin{array}{c}\underline{\mathbf{\Pi}_1}\\ F\\ \hline\hline (F)\\ \mathbf{\Pi}_2\end{array} \text{ is a proof.}$$

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

introduction rules

$$\scriptstyle\wedge\mathrm{I}\ \dfrac{\dfrac{\Pi\mathrm{N}_1}{G} \quad \dfrac{\Pi\mathrm{N}_2}{H}}{G\wedge H}$$

$$\scriptstyle\vee\mathrm{I}\ \dfrac{\dfrac{\Pi\mathrm{N}}{G}}{G\vee H} \qquad \scriptstyle\vee\mathrm{I}\ \dfrac{\dfrac{\Pi\mathrm{N}}{H}}{G\vee H}$$

$$\scriptstyle\supset\mathrm{I}\ \dfrac{\dfrac{\overset{[F]}{\Pi\mathrm{N}}}{G}}{F\supset G}$$

$$\scriptstyle\forall\mathrm{I}\ \dfrac{\dfrac{\Pi\mathrm{N}}{G(X)}}{\forall x G(x)}$$

$$\scriptstyle\exists\mathrm{I}\ \dfrac{\dfrac{\Pi\mathrm{N}}{G(t)}}{\exists x G(x)}$$

$$\scriptstyle\sim\mathrm{I}\ \dfrac{\dfrac{\overset{[G]}{\Pi\mathrm{N}}}{\#}}{\sim G}$$

elimination rules

$$\scriptstyle\wedge\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{E\wedge F}}{E} \qquad \scriptstyle\wedge\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{E\wedge F}}{F}$$

$$\scriptstyle\vee\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{E\vee F} \quad \dfrac{\overset{[E]}{\Pi\mathrm{N}_1}}{G} \quad \dfrac{\overset{[F]}{\Pi\mathrm{N}_2}}{G}}{G}$$

$$\scriptstyle\supset\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{G\supset F} \quad \dfrac{\Pi\mathrm{N}}{G}}{F}$$

$$\scriptstyle\forall\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{\forall x F(x)}}{F(t)}$$

$$\scriptstyle\exists\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{\exists x F(x)} \quad \dfrac{\overset{[F(X)]}{\Pi\mathrm{N}}}{G}}{G}$$

$$\scriptstyle\sim\mathrm{E}\ \dfrac{\dfrac{\Pi\mathrm{M}}{\sim F} \quad \dfrac{\Pi\mathrm{N}}{F}}{\#}$$

absurdity rule

$$\scriptstyle\#\mathrm{X}\ \dfrac{\dfrac{\Pi\mathrm{N}}{\#}}{G}$$

excluded middle

$$\scriptstyle\mathrm{MX}\ \dfrac{}{F\vee\sim F}$$

x	–	variable
X	–	parameter
t	–	term
E, F, G, H	–	formula
$\Pi\mathrm{M}$	–	major premiss proof
$\Pi\mathrm{N}, \Pi\mathrm{N}_1, \Pi\mathrm{N}_2$	–	normal form proof

Figure 2.4: system $\mathcal{C}_{\Pi N}$ — 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 $\mathcal{C}_{\Pi N}$, that incorporates the normal form constraint:

Normal Form Proof ($\mathbf{\Pi_N}$): A major premiss proof ($\mathbf{\Pi_M}$) is determined by the rules of inference on the right (elimination rules and excluded middle) of figure 2.4. A normal form proof ($\mathbf{\Pi_N}$) 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 $\mathbf{\Pi_M}$ is a $\mathbf{\Pi_N}$.

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 $\Delta \underset{\mathcal{C}_{\Pi}}{\vdash} G \quad iff \quad \Delta \underset{\mathcal{C}_{\Pi N}}{\vdash} G$

Proof:

$\Delta \underset{\mathcal{C}_{\Pi}}{\vdash} G \quad \Leftarrow \quad \Delta \underset{\mathcal{C}_{\Pi N}}{\vdash} G$

Every $\mathcal{C}_{\Pi N}$ proof is a $\mathcal{C}_{\Pi}$ proof.

$\Delta \underset{\mathcal{C}_{\Pi}}{\vdash} G \quad \Rightarrow \quad \Delta \underset{\mathcal{C}_{\Pi N}}{\vdash} G$

Every $\mathcal{C}_{\Pi}$ proof may be transformed into an $\mathcal{C}_{\Pi N}$ 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.

$\wedge$-reduction

$$\begin{array}{c} {\scriptstyle \wedge\mathrm{E}}\dfrac{{\scriptstyle \wedge\mathrm{I}}\dfrac{\dfrac{\Pi_1}{F_1} \quad \dfrac{\Pi_2}{F_2}}{F_1 \wedge F_2}}{(F_i)} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_i}{(F_i)} \\ \Pi \end{array}$$

$\vee$-reduction

$$\begin{array}{c} {\scriptstyle \vee\mathrm{E}}\dfrac{{\scriptstyle \vee\mathrm{I}}\dfrac{\dfrac{\Pi_0}{F_i}}{F_1 \vee F_2} \quad \begin{array}{c}[F_1]\\ \dfrac{\Pi_1}{G}\end{array} \quad \begin{array}{c}[F_2]\\ \dfrac{\Pi_2}{G}\end{array}}{(G)} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_0}{(F_i)} \\ \dfrac{\Pi_i}{(G)} \\ \Pi \end{array}$$

$\supset$-reduction

$$\begin{array}{c} {\scriptstyle \supset\mathrm{E}}\dfrac{{\scriptstyle \supset\mathrm{I}}\dfrac{\begin{array}{c}[F]\\ \dfrac{\Pi_0}{G}\end{array}}{F \supset G} \quad \dfrac{\Pi_1}{F}}{(G)} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_1}{(F)} \\ \dfrac{\Pi_0}{(G)} \\ \Pi \end{array}$$

$\forall$-reduction

$$\begin{array}{c} {\scriptstyle \forall\mathrm{E}}\dfrac{{\scriptstyle \forall\mathrm{I}}\dfrac{\dfrac{\Pi_0}{F(X)}}{\forall x F(x)}}{(F(t))} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_0(X = t)}{(F(t))} \\ \Pi \end{array}$$

$\exists$-reduction

$$\begin{array}{c} {\scriptstyle \exists\mathrm{E}}\dfrac{{\scriptstyle \exists\mathrm{I}}\dfrac{\dfrac{\Pi_0}{F(t)}}{\exists x F(x)} \quad \begin{array}{c}[F(X)]\\ \dfrac{\Pi_1}{G}\end{array}}{(G)} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_0}{(F(t))} \\ \dfrac{\Pi_1(X{=}t)}{(G)} \\ \Pi \end{array}$$

$\sim$-reduction

$$\begin{array}{c} {\scriptstyle \sim\mathrm{E}}\dfrac{{\scriptstyle \sim\mathrm{I}}\dfrac{\begin{array}{c}[F]\\ \dfrac{\Pi_0}{\#}\end{array}}{\sim F} \quad \dfrac{\Pi_1}{F}}{(\#)} \\ \Pi \end{array} \longrightarrow \begin{array}{c} \dfrac{\Pi_1}{(F)} \\ \dfrac{\Pi_0}{(\#)} \\ \Pi \end{array}$$

#-reduction

$$\begin{array}{c}{\scriptstyle -E}\dfrac{{\scriptstyle -\#X}\dfrac{\dfrac{\Pi_0}{\#}}{F_0} \quad \dfrac{\Pi_1}{F_1} \cdots \dfrac{\Pi_n}{F_n}}{(G)} \\ \Pi\end{array} \quad \longrightarrow \quad \begin{array}{c}{\scriptstyle -\#X}\dfrac{\dfrac{\Pi_0}{\#}}{(G)} \\ \Pi\end{array}$$

□

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 $\mathbf{\Pi}_{\mathrm{N}}$:

Subformula Property: For the intuitionistic system, every formula occurrence in $\mathbf{\Pi}_{\mathrm{N}}$ is a subformula of one of the premisses of $\mathbf{\Pi}_{\mathrm{N}}$ or of the conclusion of $\mathbf{\Pi}_{\mathrm{N}}$. For the classical system, negations of premisses and of the conclusion may also occur.

Minimal Formula Property: Every branch of a $\mathbf{\Pi}_{\mathrm{N}}$ 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[2] 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:

[2]The notions of *goal* and *assertion* are made precise in the next chapter.

introduction rules

$$\begin{array}{c} \dfrac{G \quad H}{(G\wedge H)}{\scriptstyle \wedge\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} \dfrac{G}{(G\vee H)}{\scriptstyle \vee\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array} \qquad \begin{array}{c} \dfrac{H}{(G\vee H)}{\scriptstyle \vee\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} [F] \\ \dfrac{G}{(F\supset G)}{\scriptstyle \supset\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} \dfrac{G(X)}{(\forall x G(x))}{\scriptstyle \forall\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} \dfrac{G(t)}{(\exists x G(x))}{\scriptstyle \exists\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} [G] \\ \dfrac{\#}{({\sim}G)}{\scriptstyle {\sim}\mathrm{I}} \\ \Pi_{\mathrm{I}} \end{array}$$

$$\begin{array}{c} \dfrac{\#}{(G)}{\scriptstyle \#\mathrm{X}} \\ \Pi_{\mathrm{I}} \end{array}$$

elimination elimination

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{E\wedge F}}{E}{\scriptstyle \wedge\mathrm{E}} \qquad \dfrac{\dfrac{\Pi_{\mathrm{E}}}{E\wedge F}}{F}{\scriptstyle \wedge\mathrm{E}}$$

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{E\vee F} \quad \dfrac{\begin{array}{c}[E]\\ \Pi_1\end{array}}{G} \quad \dfrac{\begin{array}{c}[F]\\ \Pi_2\end{array}}{G}}{G}{\scriptstyle \vee\mathrm{E}}$$

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{G\supset F} \quad \dfrac{\Pi_{\mathrm{I}}}{G}}{F}{\scriptstyle \supset\mathrm{E}}$$

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{\forall x F(x)}}{F(t)}{\scriptstyle \forall\mathrm{E}}$$

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{\exists x F(x)} \quad \dfrac{\begin{array}{c}[F(X)]\\ \Pi\end{array}}{G}}{G}{\scriptstyle \exists\mathrm{E}}$$

$$\dfrac{\dfrac{\Pi_{\mathrm{E}}}{{\sim}F} \quad \dfrac{\Pi_{\mathrm{I}}}{F}}{\#}{\scriptstyle {\sim}\mathrm{E}}$$

$$\dfrac{}{F \vee {\sim}F}{\scriptstyle \mathrm{MX}}$$

cut normal form proof

$$\begin{array}{c} \Pi_{\mathrm{E}} \\ \hline G \\ \hline\hline ({G}) \\ \hline \Pi_{\mathrm{C}} \end{array}\ {\scriptstyle \mathrm{CUT}}$$

atomic normal form proof

$$\begin{array}{c} \Pi_{\mathrm{E}} \\ \hline A \\ \hline\hline ({A}) \\ \hline \Pi_{\mathrm{A}} \end{array}\ {\scriptstyle \mathrm{CUT}}$$

x	–	variable	Π_{I}	–	introduction proof
X	–	parameter	Π_{E}	–	elimination proof
t	–	term	Π_{C}	–	cut normal form proof
A	–	atomic formula	Π_{A}	–	atomic normal form proof
E,F,G,H	–	formula	Π_1,Π_2	–	cut or atomic normal form proof

Figure 2.5: systems $\mathcal{C}_{\Pi C}$ and $\mathcal{C}_{\Pi A}$ — cut and atomic normal form proof

Introduction Proof Component ($\mathbf{\Pi}_{\mathrm{I}}$): We refer to a component generated by a goal formula as an introduction component ($\mathbf{\Pi}_{\mathrm{I}}$). A $\mathbf{\Pi}_{\mathrm{I}}$ is characterized by the inductive definition:

base A formula standing alone is a $\mathbf{\Pi}_{\mathrm{I}}$.

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 $\supset$I and $\sim$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 ($\mathbf{\Pi}_{\mathrm{E}}$): We refer to a component associated with an assertion as an elimination proof component ($\mathbf{\Pi}_{\mathrm{E}}$). A $\mathbf{\Pi}_{\mathrm{E}}$ is characterized by the inductive definition:

base A formula standing alone is a $\mathbf{\Pi}_{\mathrm{E}}$.

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[3] for the construction of cut normal form proofs.

Cut Normal Form Proof ($\mathbf{\Pi}_{\mathrm{C}}$): A cut normal form proof ($\mathbf{\Pi}_{\mathrm{C}}$) is determined by the inductive definition:

base: A $\mathbf{\Pi}_{\mathrm{I}}$ is a $\mathbf{\Pi}_{\mathrm{C}}$.

step: Given a $\mathbf{\Pi}_{\mathrm{C}}$ with premiss F, a $\mathbf{\Pi}_{\mathrm{E}}$ with conclusion F then we can compose the $\mathbf{\Pi}_{\mathrm{C}}$ and $\mathbf{\Pi}_{\mathrm{E}}$. 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.

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

Lemma 2 $\Delta \underset{c_{\Pi N}}{\vdash} G \quad iff \quad \Delta \underset{c_{\Pi C}}{\vdash} G$

Proof:

$\Delta \underset{c_{\Pi N}}{\vdash} G \quad \Leftarrow \quad \Delta \underset{c_{\Pi C}}{\vdash} G$

Every $\mathcal{C}_{\Pi C}$ proof may be transformed into a $\mathcal{C}_{\Pi N}$ proof by removing every cut rule occurrence by application of the cut-reduction transformation below.

cut-reduction

$$\begin{array}{c} \Pi_2 \\ {\scriptstyle -E}\overline{\quad F \quad} \\ {\scriptstyle =\text{CUT}}\overline{\overline{\quad (F) \quad}} \\ {\scriptstyle -I}\overline{\quad \Pi_1 \quad} \end{array} \quad \longrightarrow \quad \begin{array}{c} \Pi_2 \\ {\scriptstyle -E}\overline{\quad (F) \quad} \\ {\scriptstyle -I}\overline{\quad \Pi_1 \quad} \end{array}$$

$\Delta \underset{c_{\Pi N}}{\vdash} G \quad \Rightarrow \quad \Delta \underset{c_{\Pi C}}{\vdash} G$

Every $\mathcal{C}_{\Pi N}$ proof may be transformed into an $\mathcal{C}_{\Pi C}$ proof by adding the required instances of the cut rule by exhaustive application of the cut-expansion transformation below.

cut-expansion

$$\begin{array}{c} \Pi_2 \\ {\scriptstyle -E}\overline{\quad (F) \quad} \\ {\scriptstyle -I}\overline{\quad \Pi_1 \quad} \end{array} \quad \longrightarrow \quad \begin{array}{c} \Pi_2 \\ {\scriptstyle -E}\overline{\quad F \quad} \\ {\scriptstyle =\text{CUT}}\overline{\overline{\quad (F) \quad}} \\ {\scriptstyle -I}\overline{\quad \Pi_1 \quad} \end{array}$$

$\square$

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 (Π_A): 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 $\Delta \underset{c_{\Pi C}}{\vdash} G \quad iff \quad \Delta \underset{c_{\Pi A}}{\vdash} G$

Proof:

$\Delta \underset{c_{\Pi C}}{\vdash} G \quad \Leftarrow \quad \Delta \underset{c_{\Pi A}}{\vdash} G$

Everu $\mathcal{C}_{\Pi A}$ proof is a $\mathcal{C}_{\Pi C}$ proof.

$\Delta \underset{c_{\Pi C}}{\vdash} G \quad \Rightarrow \quad \Delta \underset{c_{\Pi A}}{\vdash} G$

The transformations below may be applied to atomize the premiss and conclusion of any CUT rule occurrence.

$\wedge$-expansion

$${\scriptstyle -I}\dfrac{{\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\wedge G}}{(F\wedge G)}}{\Pi_1} \quad\longrightarrow\quad {\scriptstyle -I}\dfrac{{\scriptstyle -\wedge I}\dfrac{{\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -\wedge E}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\wedge G}}{F}}{F} \quad {\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -\wedge E}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\wedge G}}{G}}{G}}{(F\wedge G)}}{\Pi_1}$$

$\vee$-expansion

$${\scriptstyle -I}\dfrac{{\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\vee G}}{(F\vee G)}}{\Pi_1} \quad\longrightarrow\quad {\scriptstyle -I}\dfrac{{\scriptstyle -\vee E}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\vee G} \quad \dfrac{{\scriptstyle =\text{CUT}=}\dfrac{\dfrac{\quad}{F}{\scriptstyle (i)}}{F}}{F\vee G} \quad \dfrac{{\scriptstyle =\text{CUT}=}\dfrac{\dfrac{\quad}{G}{\scriptstyle (i)}}{G}}{F\vee G}}{(F\vee G)}{\scriptstyle (i)}}{\Pi_1}$$

$\supset$-expansion

$${\scriptstyle -I}\dfrac{{\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\supset G}}{(F\supset G)}}{\Pi_1} \quad\longrightarrow\quad {\scriptstyle -I}\dfrac{{\scriptstyle -\supset I}\dfrac{{\scriptstyle =\text{CUT}=}\dfrac{{\scriptstyle -\supset E}\dfrac{{\scriptstyle -E}\dfrac{\Pi_2}{F\supset G} \quad \dfrac{\quad}{F}{\scriptstyle (i)}}{G}}{G}}{(F\supset G)}{\scriptstyle (i)}}{\Pi_1}$$

$\forall$-expansion

$$\dfrac{\dfrac{\dfrac{\Pi_2}{\forall x F(x)}{\scriptstyle -E}}{(\forall x F(x))}{\scriptstyle =\mathrm{CUT}}}{\Pi_1}{\scriptstyle -I} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_2}{\forall x F(x)}{\scriptstyle -E}}{F(X)}{\scriptstyle -\forall\mathrm{E}}}{F(X)}{\scriptstyle =\mathrm{CUT}}}{(\forall x F(x))}{\scriptstyle -\forall\mathrm{I}}}{\Pi_1}{\scriptstyle -I}$$

$\exists$-expansion

$$\dfrac{\dfrac{\dfrac{\Pi_2}{\exists x F(x)}{\scriptstyle -E}}{(\exists x F(x))}{\scriptstyle =\mathrm{CUT}}}{\Pi_1}{\scriptstyle -I} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\Pi_2}{\exists x F(x)}{\scriptstyle -E} \quad \dfrac{\dfrac{\overline{F(X)}^{(i)}}{F(X)}{\scriptstyle =\mathrm{CUT}}}{\exists x F(x)}}{(\exists x F(x))}{\scriptstyle -\exists\mathrm{E}\,(i)}}{\Pi_1}{\scriptstyle -I}$$

$\sim$-expansion

$$\dfrac{\dfrac{\dfrac{\Pi_2}{\sim F}{\scriptstyle -E}}{(\sim F)}{\scriptstyle =\mathrm{CUT}}}{\Pi_1}{\scriptstyle -I} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_2}{\sim F}{\scriptstyle -E} \quad \overline{F}^{(i)}}{\#}{\scriptstyle -\supset\mathrm{E}}}{\#}{\scriptstyle =\mathrm{CUT}}}{(\sim F)}{\scriptstyle -\supset\mathrm{I}\,(i)}}{\Pi_1}{\scriptstyle -I}$$

□

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 $\Delta \underset{c_\Pi}{\vdash} G \quad iff \quad \Delta \underset{c_{\Pi A}}{\vdash} G$

Proof: Immediately from lemmas 1, 2 and 3 and transitivity of iff. □

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 Δ and a query formula G, what, if any, proofs of G from Δ exist in S? We symbolise this problem as

$$\Delta \underset{\mathrm{S}}{?-} G$$

The members of Δ (*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 Γ, where Γ is a subset of Δ, is a *solution.* Such a solution is summarised by the deducibility assertion:

$$\Gamma \underset{\mathrm{S}}{\vdash} G$$

Consider the example deduction problem:

$$\{p\wedge q,\ r\} \underset{\mathcal{C}_{\Pi N}}{?-} p\vee(q\wedge r)$$

Notice that although this problem is posed for the system $\mathcal{C}_{\Pi N}$ (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.

$$\dfrac{\scriptstyle\text{AXIOM}}{p\wedge q} \qquad\qquad \dfrac{\scriptstyle\text{AXIOM}}{r} \qquad\qquad \dfrac{p\vee(q\wedge r)}{\scriptstyle\text{QUERY}}$$

(a) (b) (c)

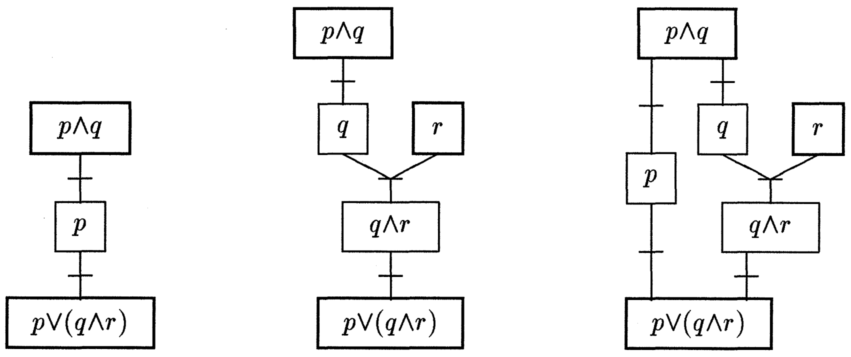
Figure 3.1: axioms and query

$$\dfrac{{\scriptstyle\vee\text{I}}\,\dfrac{{\scriptstyle\wedge\text{E}}\,\dfrac{\scriptstyle\text{AXIOM}}{p\wedge q}}{p}}{p\vee(q\wedge r)}\,{\scriptstyle\text{QUERY}} \qquad\qquad \dfrac{{\scriptstyle\vee\text{I}}\,\dfrac{{\scriptstyle\wedge\text{I}}\,\dfrac{{\scriptstyle\wedge\text{E}}\,\dfrac{\scriptstyle\text{AXIOM}}{p\wedge q} \qquad \dfrac{\scriptstyle\text{AXIOM}}{r}}{q \qquad\qquad r}}{q\wedge r}}{p\vee(q\wedge r)}\,{\scriptstyle\text{QUERY}}$$

(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.



(a) (b) (c)

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 $\mathcal{C}_{\Pi A}$ for ANF proofs, developed in the preceding chapter, so that it only admits proofs that are solutions. The aim of this new system $\mathcal{C}_{\Sigma}(\mathbf{\Phi})$ is to reflect, as clearly as possible, an efficient method for computing solutions for a given deduction problem $\mathbf{\Phi}$. 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 $\mathcal{C}_{\Pi A}$, 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 $\mathcal{C}_{\Sigma}(\mathbf{\Phi})$ 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 $\mathcal{C}_{\Pi}$ 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 premisses 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.

$$ {\scriptstyle \wedge\mathrm{I}}\dfrac{\dfrac{\Pi_1}{G} \quad \dfrac{\Pi_2}{H}}{G \wedge H} \qquad\qquad {\scriptstyle \wedge\mathrm{E}}\dfrac{\dfrac{\Pi}{E \wedge F}}{E} \qquad {\scriptstyle \wedge\mathrm{E}}\dfrac{\dfrac{\Pi}{E \wedge F}}{F} $$

Figure 3.5: example forward chaining deduction system

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

$$\{p\wedge q,\ r\} \ \underset{3.5}{?\!-}\ q\wedge r$$

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 $p \wedge q$ and r. The next two levels generated by a forward chaining, breadth first search strategy are displayed in figure 3.7.

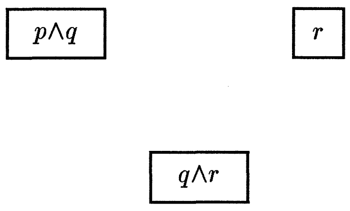


Figure 3.6: initial AND/OR graph

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.

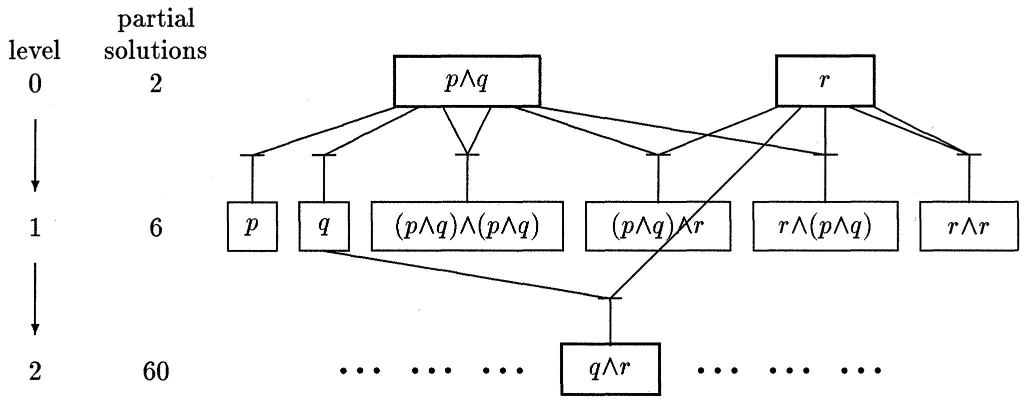


Figure 3.7: forward chaining search space

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

$$\begin{array}{c} -{\scriptstyle\wedge\mathrm{I}}\dfrac{G \qquad H}{(G\wedge H)} \\ \mathbf{\Pi} \end{array} \qquad\qquad \begin{array}{c} -{\scriptstyle\wedge\mathrm{E}}\dfrac{E\wedge F}{(E)} \\ \mathbf{\Pi} \end{array} \qquad \begin{array}{c} -{\scriptstyle\wedge\mathrm{E}}\dfrac{E\wedge F}{(F)} \\ \mathbf{\Pi} \end{array}$$

Figure 3.8: example backward chaining deduction system

of figure 3.8 for the problem:

$$\{p\wedge q,\ r\}\ \underset{3.8}{?-}\ q\wedge r$$

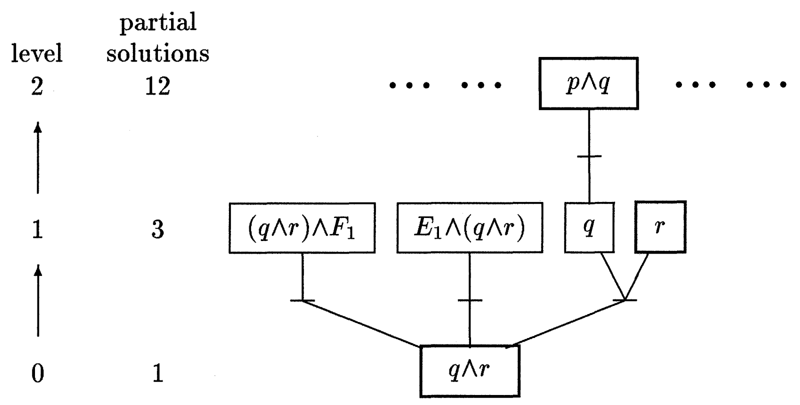


Figure 3.9: backward chaining search space

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:

$$\{p\wedge q,\ r\}\ \underset{3.10}{?-}\ q\wedge r$$

$$\begin{array}{c} \dfrac{G \quad H}{(G \wedge H)}{\scriptstyle \wedge I} \\ \Pi \end{array} \qquad\qquad \dfrac{\dfrac{\Pi}{E \wedge F}}{E}{\scriptstyle \wedge E} \qquad \dfrac{\dfrac{\Pi}{E \wedge F}}{F}{\scriptstyle \wedge E}$$

Figure 3.10: combination forward backward system

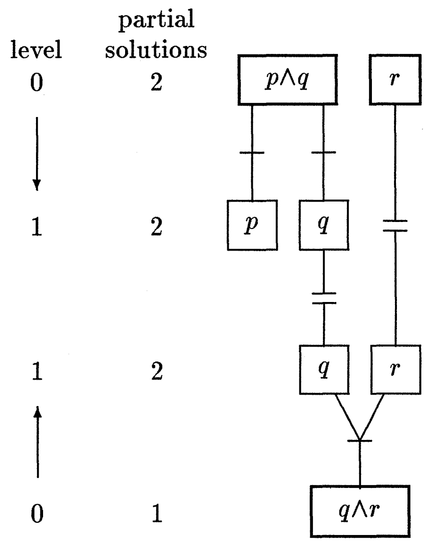


Figure 3.11: ANF (combined forward backward) search space

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 $\mathcal{C}_\Sigma(\Phi)$. 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 $\mathcal{C}_\Sigma(\mathbf{\Phi})$

For a given problem $\mathbf{\Phi}$ the deduction system $\mathcal{C}_\Sigma(\mathbf{\Phi})$ determines what is to count as an ANF solution of $\mathbf{\Phi}$. As already noted, the deduction system $\mathcal{C}_\Sigma(\mathbf{\Phi})$ 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 $\mathcal{C}_\Sigma(\mathbf{\Phi})$ 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, $\forall$E, $\exists$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 $\vee$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 $\mathcal{C}_\Sigma(\mathbf{\Phi})$ represent the particulars of the deduction problem $\mathbf{\Phi}$ at hand. A problem statement

$$\{F_1, F_2, \ldots, F_n\} \ ?\!-\ G$$

is represented as follows:

$$\begin{array}{cc} \text{introduction component} & \text{elimination component} \\[1em] \dfrac{G}{\quad} \,\textsc{query} & \dfrac{\quad}{F_i} \,\textsc{axiom} \\[1em] \text{(a)} & \text{(b)} \end{array}$$

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).

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 (Σ_{I}): A Σ_{I} is determined by the inductive definition:

base The query rule is a Σ_{I}. The minor premiss formula of any elimination rule is a Σ_{I}.

step The clauses of the definition are supplied by the introduction rules. Note that the assumptions thrown up by the $\supset$I and $\sim$I rules give rise to elimination solution components.

Σ_{I} 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 (Σ_{E}): A Σ_{E} is determined by the inductive definition:

base An axiom rule is a Σ_{E}. An assumption thrown up by either the implication or negation introduction rules is a Σ_{E}.

step The clauses of the definition are supplied by the elimination rules. Note that the minor premisses of the $\supset$E and $\sim$E rules give rise to introduction solution components.

Σ_{E} 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 (Σ_{A}): A Σ_{A} is characterized by the following backward chaining definition:

base A Σ_{I} is a Σ_{A}.

step Given a Σ_{A} with atomic premiss A, a Σ_{E} with atomic conclusion B and that $A\Theta = B\Theta$ then we may compose $\Sigma_{\text{A}}\Theta$ and $\Sigma_{\text{E}}\Theta$ to form a new solution. This computational form of the cut rule is shown at the bottom of figure 3.13.

Σ_{A} is *complete* if it has no premisses and no conclusion, otherwise it is *partial.*

We also require that a complete Σ_{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 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 fomulae. 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 ocurrences to partition the subformulae of a deduction problem statement into goal and assertion subformulae.

introduction component

$$-\wedge\text{I}\;\begin{array}{c} G \quad H \\ \hline (G\wedge H) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-\vee\text{I}\;\begin{array}{c} G \\ \hline (G\vee H) \\ \Sigma_{\mathbf{I}} \end{array} \qquad -\vee\text{I}\;\begin{array}{c} H \\ \hline (G\vee H) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-\supset\text{I}\;\begin{array}{c} [F] \\ G \\ \hline (F\supset G) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-\forall\text{I}\;\begin{array}{c} G(X^S) \\ \hline (\forall x G(x)) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-\exists\text{I}\;\begin{array}{c} G(X) \\ \hline (\exists x G(x)) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-{\sim}\text{I}\;\begin{array}{c} [G] \\ \# \\ \hline ({\sim}G) \\ \Sigma_{\mathbf{I}} \end{array}$$

$$-\#\text{X}\;\begin{array}{c} \# \\ \hline (G) \\ \Sigma_{\mathbf{I}} \end{array}$$

elimination component

$$-\wedge\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline E\wedge F \\ \hline E \end{array} \qquad -\wedge\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline E\wedge F \\ \hline F \end{array}$$

$$-\vee\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline E\vee F \\ \hline E \quad F \end{array}$$

$$-\supset\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \quad \Sigma_{\mathbf{I}} \\ \hline G\supset F \quad G \\ \hline F \end{array}$$

$$-\forall\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline \forall x F(x) \\ \hline F(X) \end{array}$$

$$-\exists\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline \exists x F(x) \\ \hline F(X^S) \end{array}$$

$$-{\sim}\text{E}\;\begin{array}{c} \Sigma_{\mathbf{E}} \quad \Sigma_{\mathbf{I}} \\ \hline {\sim}F \quad F \\ \hline \# \end{array}$$

$$-\text{MX}\;\begin{array}{c} \\ \hline F \vee {\sim}F \end{array}$$

structural rule

$$\left(=\text{CUT}\;\begin{array}{c} \Sigma_{\mathbf{E}} \\ \hline B \\ \hline\hline (A) \\ \hline \Sigma_{\mathbf{A}} \end{array} \right)\Theta \quad \text{where: } A\Theta = B\Theta$$

x	–	variable	A, B	–	atomic formula
X	–	parameter	E, F, G, H	–	formula
X^S	–	Skolem parameter	$\Sigma_{\mathbf{I}}$	–	introduction component
t	–	term	$\Sigma_{\mathbf{E}}$	–	elimination component
Θ	–	substitution	$\Sigma_{\mathbf{A}}$	–	atomic normal form solution

Figure 3.13: system $\mathcal{C}_\Sigma$— 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 $\wedge$, $\vee$, $\supset$, $\sim$, $\forall$, $\exists$ 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.

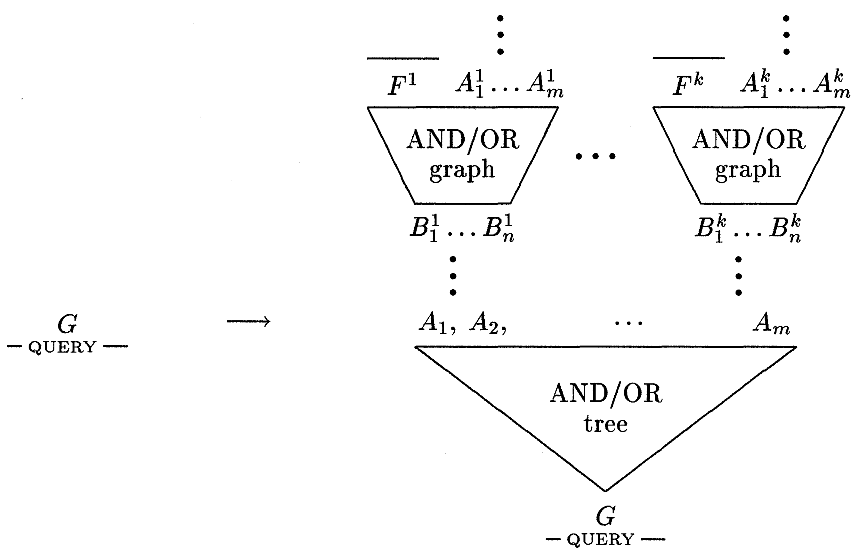


Figure 3.14: inferential extension of a query

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.

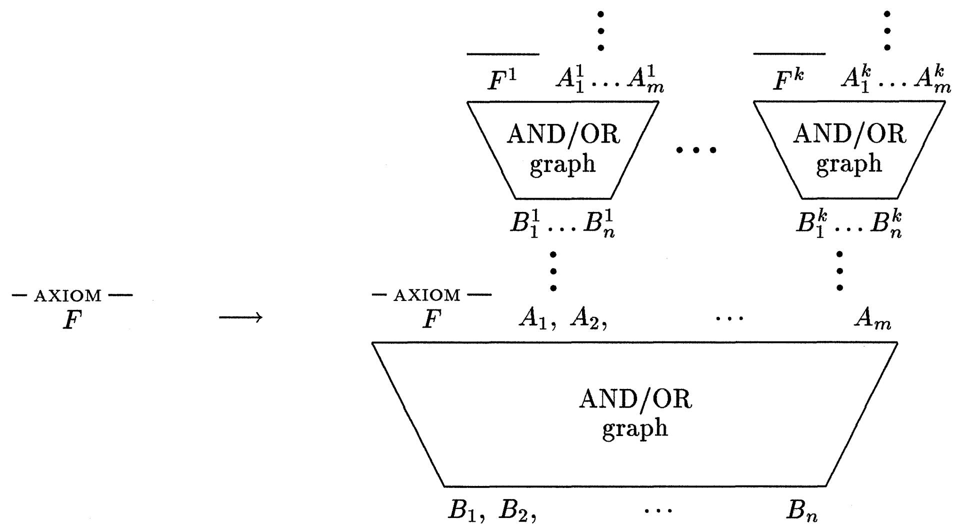


Figure 3.15: inferential extension of an axiom

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 Θ_i is a full renaming substitution. That is, Θ_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 $\{X_1, X_2, \ldots, X_m\}$.

and

Apart: Every substitution Θ_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.

By a CUT rule instance we mean the following:

CUT **Instance:** A CUT instance is a triple $\langle A, B, \Theta \rangle$ where A is an atomic goal occurrence, B an atomic assertion occurrence and Θ the most general unifier of A and B [Robinson 65], [Lassez et al. 88].

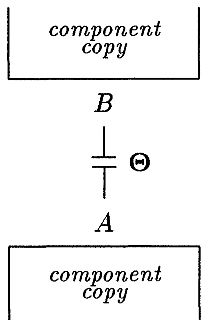


Figure 3.16: CUT rule instance

The notation of figure 3.16 is used for 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 CUT instances sharing a common conclusion formula into *choice points.*

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.

Figure 3.17 illustrates a choice point.

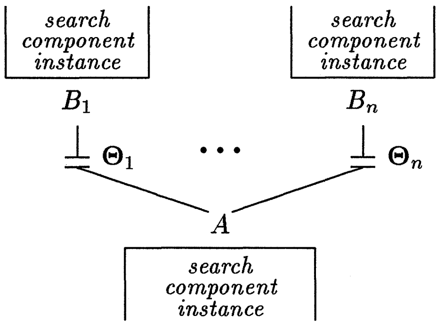

search component instance
search component instance
B_1
B_n
Θ_1
...
Θ_n
A
search component instance

Figure 3.17: choice point

In the interest of computational efficiency, it is important to minimize the number of 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 Bs that remain the *admissible assertions* for A.

Subgoal Context: If the path from A down to the query is known, only assumption search components arising from $\supset$I and $\sim$E rule applications on the path need be included.

Case Argument Context: Admissible assertions arising from $\vee$E 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 $\mathbf{\Phi}$.

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

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

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

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:

Loop Free Solution: Any proposed solution of the form shown in figure 3.18 (c), where $A_1 = A_2\Theta$ and Σ_1 does not discharge any assumptions in Σ_2, should be discarded in favour of the simpler form shown in (d).

A definition for an ANF solution for deduction problem $\mathbf{\Phi}$ follows.

$$\begin{array}{c} \mathbf{\Sigma} \\ \hline (A) \\ \mathbf{\Sigma} \\ \hline A \\ \\ \text{(a)} \end{array} \qquad \begin{array}{c} \\ \\ \\ \mathbf{\Sigma} \\ \hline A \\ \\ \text{(b)} \end{array} \qquad \begin{array}{c} \mathbf{\Sigma}_2 \\ \hline (A_2) \\ \mathbf{\Sigma}_1 \\ \hline A_1 \\ \\ \text{(c)} \end{array} \qquad \begin{array}{c} \\ \\ \\ \mathbf{\Sigma}_2\mathbf{\Theta} \\ \hline A_1 \\ \\ \text{(d)} \end{array}$$

Figure 3.18: transformation to loop free solution

ANF Solution Graph: An ANF solution graph for $\mathbf{\Phi}$ is a finite subgraph of an ANF search graph for $\mathbf{\Phi}$ 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 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 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.