Deduction Plans: a graphical proof procedure for the first-order predicate calculus

Philip T. Cox
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

Research Report CS-77-28

October 1977
DEDUCTION PLANS: A GRAPHICAL PROOF PROCEDURE
FOR THE FIRST-ORDER PREDICATE CALCULUS

Abstract
by
Philip Trevor Cox

A proof procedure is described that relies on the construction of certain directed graphs called "deduction plans". Plans represent the structure of proofs in such a way that problem-reduction may be used without imposing any ordering on the solution of subproblems, as required by other systems. The structure also allows access to all clauses deduced in the course of a proof, which may then be used as lemmas. Economy of representation is the maximum attainable, consistent with this unrestricted availability of lemmas.

Various restrictions of this deduction system are seen to correspond to existing linear deduction procedures, while overcoming many of their shortcomings. One of the rules for constructing plans, however, has no equivalent in existing systems.

A further economy is obtained by obviating the necessity for explicitly performing substitutions and for calculating most general unifiers.
An algorithm for determining the cause of unification failure is shown to exist. This allows the source of conflict to be located when a subproblem is found to be unsolvable, so that exact backtracking can be performed rather than the blind backtracking performed by existing systems. Therefore, a deduction system based on the construction of plans can avoid the wasteful search of irrelevant areas of the search space that results from the usual backtracking methods. Furthermore, because of the graphical structure, it is necessary to remove only the offending parts of the proof when a plan is pruned after backtracking, rather than the entire proof constructed after the cutting point.
Acknowledgements

First I want to thank Tom Pietrzykowski, who supervised this research with unfailing patience, constant encouragement and helpful criticism.

Thanks are also due to my external examiner, Ray Reiter, who has shown a continued interest in my work during its development, and to Maarten van Emden, Ed Ashcroft and Ronald Read for their careful reading of this thesis.

Finally, I wish to thank my friends for their encouragement when times were bad.
Table of Contents

Abstract.........................................................................................(iv)
Acknowledgements........................................................................... (vi)
CHAPTER 1: Introduction.................................................................1
CHAPTER 2: Preliminaries.................................................................8
  2.1: Graph Theory...........................................................................8
  2.2: Language...............................................................................11
  2.3: Substitution and unification..................................................14
  2.4: Predicate calculus...............................................................19
CHAPTER 3: Deduction Plans............................................................21
  3.1: Definition..............................................................................28
  3.2: Comments, notation and preliminary results.........................39
  3.3: Soundness and Completeness................................................68
CHAPTER 4: Constraint Processing..................................................89
  4.1: The Baxter unification algorithm...........................................94
  4.2: The modified unification algorithm.......................................99
    4.2.1: The algorithm CLASSIFY.............................................100
    4.2.2: The automaton for a constraint set..................................127
    4.2.3: The unification graph.....................................................141
  4.3: The failure location process.................................................146
CHAPTER 5: Illustrations and Conclusions......................................177
  5.1: Backtracking.........................................................................177
  5.2: Deduction plans and linear deduction....................................185
  5.3: Graph theory in theorem-proving.........................................193
REFERENCES....................................................................................196
CHAPTER 1

Introduction

Mechanical theorem-proving in essentially its modern form began in 1965 with the advent of Robinson's resolution principle [36]. This is a system of first-order predicate calculus which expresses sentences in a quantifier-free conjunctive normal form called "clause form", has no logical axioms, and has only one rule of inference. The resolution principle was expected to be particularly suited to theorem-proving by computer, since one application of the inference rule requires considerable amounts of computation, but is equivalent to several inferences in the traditional systems of predicate calculus.

The initial enthusiasm with which the resolution principle was received rapidly gave way to disappointment when early theorem-proving programs were found to be unable to prove any but the simplest theorems without exceeding their usually generous storage limits. This early failure led to a completely ill-founded condemnation, by some researchers, of resolution as an inference rule. The fault, of course, lay not with the deduction rule, but with the strategy employed in the search for a proof. The strategy
initially applied was "saturation": that is, all possible deductions are performed on the original set of clauses to obtain a new expanded set; then all possible deductions are performed on this new set, and so on. The process terminates when the clause is obtained whose theoremhood is being established, or when the empty clause (a contradiction) is obtained in a refutational system. Clearly, any theorem-prover operating in this way will be choked by exponential growth in the number of clauses it must handle.

The failure of theorem-provers using saturation search engendered in the late sixties a multitude of strategies for limiting the size of the search space generated by the resolution rule \([30,10]\). Unfortunately, little improvement was obtained using these strategies, and many researchers, disillusioned with predicate logic as a problem-solving tool, adopted a more pragmatic approach. As a result several programming languages were produced for solving problems of a general type \([15,31,37]\). The power of these languages lay in their use of "problem reduction", a well-known technique dating from the early game-playing and general problem-solving programs. Problem reduction entails substituting for a particular problem a set of simpler subproblems, the simultaneous solution of which implies the solution of the original problem.
Paralleling the development of these so-called "planning" languages, a new refinement of resolution appeared called "linear resolution", independently proposed by Loveland [26], Luckham [28], and Zanov and Sharonov [44]. Linear resolution also employed problem reduction, and appeared to be one of the more promising refinements of resolution. It has in fact led to the development of predicate logic as a programming language [13,14,20,21], implemented under the name of PROLOG [2,35]. Now that the optimism of the sixties has given way to more sensible aims, mechanical theorem-provers are not expected to provide complete automation of mathematics, and in this saner light PROLOG is doing much to redeem predicate calculus as a problem-solving tool. It has been used to program systems for understanding natural language [11,32]; for performing analytic integration and formula-manipulation [6,18]; and, most significantly, it has been used to implement a general problem-solving system of the planning variety [42], which compares favorably with the planning languages mentioned above. It is clear from the practical success of PROLOG that the study of linear theorem-proving systems is a worthwhile endeavour.

Existing linear deduction systems suffer from several drawbacks. The particular system on which PROLOG is based is complete only for sets of clauses of a certain restricted type (a system is complete if it is guaranteed to produce a refutation for an unsatisfiable set of clauses). This
restriction can lead to difficulties when one wishes to use the negation of a predicate. One solution to this problem is to use the original simple linear system \([26,28,44]\), in which any clause deduced in the course of a proof may be used as a "lemma"; that is, it may be used as if it were a member of the set of clauses whose unsatisfiability the system is trying to prove. This of course requires that copies are kept of all clauses deduced, and that this list is continually scanned for useful lemmas. The storage problem could be overcome by the use of a structure-sharing scheme such as that proposed by Boyer and Moore \([8]\) (Roberts' PROLOG implementation \([35]\) uses such a scheme), but this solves only the storage problem; the list of lemmas must still be scanned. Another solution to PROLOG's incompleteness is to use the linear system proposed by Loveland as Medel Elimination \([24,25,27]\), and Kowalski and Kuehner as SL-resolution \([23]\). In these systems, a rule is used which corresponds to the familiar proof technique "reductio ad absurdum". To ensure soundness, however, a strict ordering must be imposed on the solution of subproblems (a system is sound if it produces refutations for only unsatisfiable sets of clauses). Consequently, we lose one of the most attractive features of problem reduction, the parallel processing of subproblems. We quote Kowalski \([22]\):
"Although in many cases such a strategy is desirable, in other cases a more flexible rule is useful. The ability to attempt the achievement of several subgoals simultaneously is especially important in the general case when subgoals are not independent."

Linear deduction also suffers from two problems associated with "backtracking". If a subproblem is found to be unsolvable in the course of a proof, the system must return to an earlier state of the proof and attempt an alternative solution to a previously solved subproblem. The strategy normally adopted by a linear theorem-prover, PROLOG in particular, is to return to the last subproblem it solved for which there is an untried potential solution: this may not be the correct place to try an alternative, however, and although the correct point will be reached eventually, much effort may be expended in the meantime on a fruitless exploration of the search space. This inefficiency is compounded by the pruning that occurs whenever the system backtracks: when the linear prover returns to a subproblem A to attempt an alternative solution, it erases all the clauses produced since the last attempt at solving A. Some of these clauses may constitute a perfectly acceptable solution to some subproblem, and may eventually be re-generated. Furthermore, because of its blind backtracking behaviour, the linear system may backtrack over this
innocuous subproof several times, and regenerate it several
times.

An integral part of any mechanical theorem-prover is a
unification algorithm for determining whether a pair of
formulae have a common instance. The unification algorithm
provides the theorem-prover with its only link to the
internal structure of the literals in a set of clauses:
this is where the information lies concerning unification
failure. Because of this, we contend that mechanical
deduction systems should be designed to take advantage of
the particular features of their unification algorithms.
This has not been the custom in the past, and despite the
existence of several good unification algorithms
[3,4,5,33,41], mechanical theorem-provers usually use some
modification of Robinson's original inefficient algorithm
[36].

We present here a deduction system which overcomes the
above-mentioned deficiencies of linear resolution. It is
based on the linear systems, and is designed to use a
modification of a recent unification algorithm due to Baxter
[4,5]. In this system, a proof is represented as a directed
graph, the vertices of which are occurrences of literals
from the set of clauses under consideration. It is not a
new idea to use a graphical structure for predicate
calculus: Yates, Raphael and Bart [43], Sickel [39], Shostak
[38], and Kowalski [22] have all proposed systems based on
graphs. Ours, however, bears no resemblance to any of these. Our graphs are called "deduction plans": the word "plan" is intended to convey the notion that such a graph proves nothing until the rules used in its construction have been validated by the unification algorithm. To each rule of the various linear deduction systems, there corresponds a rule for plan construction; however, even though the rules for Model Elimination have equivalents in plan construction, no ordering need be imposed on the solution of the subproblems of a plan to obtain soundness.

Plans allow the use of lemmas as in simple linear deduction, but more lemmas are available. This is because each plan actually corresponds to a set of linear deductions, and any clause deduced in any one of these deductions is available as a lemma.

Plans represent proofs economically in that each literal used in a proof is represented only once.

A modification of Baxter's unification algorithm is used to verify the applicability of each rule as a plan is constructed, by performing an appropriate unification; however, no substitutions need be performed. If the unification algorithm detects nonunifiability at some stage, a tracing algorithm determines all the choices for backtracking, and the graphical structure of the proof ensures that no harmless subproofs are removed in the subsequent pruning.
CHAPTER 2

Preliminaries

Here we provide some notation, make preliminary definitions, and quote familiar results for later reference.

2.1: Graph Theory

With a few minor exceptions, our notation and definitions for the concepts of graph theory follows Bondy and Murty [7].

2.1.1: Definition: A directed graph \( G \), is an ordered triple \( <V(G), E(G), \psi_G> \), where \( V(G) \) is a nonempty set of vertices, \( E(G) \) is a set of arcs, disjoint from \( V(G) \), and \( \psi_G \) is an incidence function from \( E(G) \) to \( V(G) \times V(G) \). If \( e \) is an arc and \( u \) and \( v \) are vertices such that \( \psi_G(e) = (u,v) \), then \( e \) is said to join \( u \) to \( v \), \( u \) is called the tail of \( e \), and \( v \) is called the head of \( e \). We also say that \( e \) leaves \( u \) and enters \( v \). The indegree and outdegree of a vertex \( v \), are respectively the number of arcs which enter \( v \), and the number of arcs which leave \( v \).

We will henceforth abbreviate "directed graph" to "digraph".
2.1.2: **Definition:** A digraph \( D \) is a **subdigraph** of a digraph \( G \) if \( V(D) \subseteq V(G), \ E(D) \subseteq E(G) \) and \( \psi_D = \psi_G|E(D) \), where \(|\) denotes restriction.

If \( E' \subseteq E(G) \), then \( G' \) is a subdigraph of \( G \) where:

\[
V(G') = \{ v \mid \exists e \in E' \text{ such that } v \text{ is either the head or the tail of } e \} 
\]

\[
E(G') = E'
\]

\[
\psi_{G'} = \psi_G|E'
\]

\( G' \) is called the **subdigraph induced by** \( E' \).

2.1.3: **Definition:** If \( G \) is a digraph, a **directed walk** in \( G \) is a sequence \( v_1, e_1, v_2, e_2, \ldots, e_n, v_{n+1} (n \geq 1) \) whose elements are alternately vertices and arcs of \( G \), which begins with a vertex and ends with a vertex, and is such that for all \( i \) \((1 \leq i \leq n)\), \( v_i \) is the tail of \( e_i \), and \( v_{i+1} \) is the head of \( e_i \).

The **length** of the walk is \( n \); \( v_1 \) and \( v_{n+1} \) are respectively the **origin** and **terminus**, and \( v_2, \ldots, v_n \) are called the **internal vertices** of the walk.

We will frequently use such expressions as "\( v \) lies on the walk"; "a walk from \( u \) to \( v \)"; and "the walk passes through \( v \)." The meaning of such expressions is obvious.

Note that a walk can be unambiguously specified as a sequence of arcs, and in the case when \( \psi_G \) is injective, a walk can be unambiguously specified as a sequence of vertices. We will use both representations from time to time.
2.1.4: **Definition:** A directed walk \(v_1, e_1, \ldots, e_n, v_{n+1}\) is called a directed trail if for all \(i\) and \(j\) \((1 \leq i, j \leq n)\), \(e_i = e_j\) implies \(i = j\). A directed trail is called a directed path if for all \(i\) and \(j\) \((1 \leq i, j \leq n)\), \(v_i = v_j\) implies either \(i = j\) or \(i = 1\) and \(j = n+1\).

Note that a path can be regarded as the subdigraph induced by the set of arcs in the path; hence if \(W\) is a path, \(E(W)\) and \(V(W)\) are defined.

2.1.5: **Definition:** A closed directed walk is a directed walk in which the origin and terminus are identical. We usually refer to a closed directed path as a directed cycle.

2.1.6: **Definition:** A labelled digraph \(G\) is an ordered 4-tuple \(<V(G), E(G), I(G), \psi_G>\), where \(V(G)\) is a nonempty set of vertices; \(E(G)\) is a set of arcs disjoint from \(V(G)\); \(I(G)\) is a nonempty set of labels; and \(\psi_G\) is an incidence function from \(E(G)\) to \(V(G) \times I(G) \times V(G)\). If \(e\) is an arc, and \(\psi_G(e) = (u, l, v)\), \(l\) is called the label of \(e\). To avoid repetition, we note that all concepts defined in 2.1.1 to 2.1.5 in connection with digraphs, can be defined in exactly the same way for labelled digraphs.

Finally, in order to simplify our notation and descriptions, we make two observations. First, if the incidence function \(\psi_G\) of a digraph \(G\) is injective, we can
consider $E(G)$ as being a subset of $V(G) \times V(G)$ (or $V(G) \times I(G) \times V(G)$ for a labelled digraph): this is the case for all digraphs encountered in this thesis. Secondly, since we consider only directed graphs, we will usually omit the word "directed" and the prefix "di-", using "graph", "walk", "path", "subgraph", etc., instead of "directed graph", "directed walk", "directed path", "subdigraph", etc.

2.2: Language

Presentations of deduction systems usually do not include a thorough treatment of unification; consequently, the terminology surrounding concepts common to these two areas is not uniform. In this section we present a language suitable for discussing both theorem-proving and unification.

2.2.1: Definition: An alphabet is a 4-tuple $(V, M, P, \text{degree})$ where:

(i) $V$, $M$ and $P$ are mutually disjoint, nonempty countable sets.

(ii) degree is a function from $M \cup P \cup \{-\}$ to the nonnegative integers, where "-" is a special symbol not in $M$, $P$ or $V$, and $\text{degree}(-) = 1$.

Elements of $V$ are called variables; elements of $M$ are called mapping symbols and elements of $P$ are called predicate symbols. Elements of $M \cup P \cup \{-\}$ are called function
symbols. If \( s \) is any function symbol and \( \text{deg}(s) = m \), we say that \( s \) is of degree \( m \).

We juxtapose the elements of an alphabet, together with the punctuation symbols "("; ")", ",," according to certain rules, to obtain certain classes of strings. All the following definitions are with respect to some alphabet.

2.2.2: Definition: An expression is:

either (i) a variable

or (ii) a string of the form \( f() \) where \( f \) is a mapping symbol of degree 0.

or (iii) a string of the form \( f(p_1, \ldots, p_n) \), where \( f \) is a mapping symbol of degree \( n \geq 0 \) and \( p_1, \ldots, p_n \) are expressions.

An expression of the form \( f() \) is called a constant.

2.2.3: Definition: An atom is:

either (i) a string of the form \( P() \) where \( P \) is a predicate symbol of degree 0.

or (ii) a string of the form \( P(p_1, \ldots, p_n) \), where \( P \) is a predicate symbol of degree \( n \geq 0 \), and \( p_1, \ldots, p_n \) are expressions.

An atom of the form \( P() \) is called a proposition.
2.2.4: Definition: A literal is either an atom or a string of the form \(-(A)\), where A is an atom.

2.2.5: Definition: If L is a literal, the negation of L is the literal:
(i) \(-(A)\) if L is the literal A, where A is an atom
(ii) A if L is the literal \(-(A)\).
We denote the negation of a literal L by \(-L\). Note that \(-\) is not a symbol in the language, but is a bijection from the set of literals to the set of literals.

2.2.6: Definition: A formula is either an expression or a literal. A term is a formula which is not a variable.

2.2.7: Definition: ord is a function from the set of formulae to the nonnegative integers, defined as follows:

\[
ord(p) = \begin{cases} 
0 & \text{if } p \text{ is a variable, constant or proposition} \\
1 + \max \{ord(p_i) \mid 1 \leq i \leq n\} & \text{if } p = f(p_1, \ldots, p_n) 
\end{cases}
\]

If \(ord(p) = m\), we say that \(p\) is of order \(m\).

2.2.8: Definition: If \(p\) and \(q\) are formulae, then \(q\) is a subformula of \(p\) if:

either (i) \(q = p\)
or (ii) \(q\) is a subformula of \(p_i\) for some \(i\) \((1 \leq i \leq n)\), where \(p = f(p_1, \ldots, p_n)\).
Note that if \( q \) is a subformula of \( p \), then \( \text{ord}(p) \geq \text{ord}(q) \), and \( \text{ord}(p) = \text{ord}(q) \) iff \( p = q \). A formula \( p \) is said to be variable-free if no subformula of \( p \) is a variable.

2.2.10: Notational conventions

We will abbreviate a literal of the form \(- (A)\) as \(-A\). If \( F \) is a function symbol of degree 0, we will abbreviate the formula \( F(\ ) \) as \( F \).

Context will always allow us to decide whether a particular symbol is a mapping symbol or a predicate symbol if its degree is \( >0 \). However, having abbreviated each formula of order 0 as the function symbol with which it begins, we can now no longer decide whether a symbol not followed by \( "(\)" represents a variable or a constant. In such cases, we will always make the meaning explicit, and where possible, represent constants by lower case letters from early in the Roman alphabet, while representing variables by lower case letters from near the end of the Roman alphabet.

2.3: Substitution and unification

2.3.1: Definition: A substitution is a finite set of ordered pairs \( \{(v_1,p_1),\ldots,(v_n,p_n)\} \) where \( v_1,\ldots,v_n \) are distinct variables, \( p_1,\ldots,p_n \) are expressions, and for each
i (1 ≤ i ≤ n), \( v_i \neq p_{i-1} v_1, \ldots, v_n \) are called the replaced variables of the substitution, and each element of the substitution is called a component. We will denote substitutions by lower case Greek letters, except for the empty substitution, denoted by \( \emptyset \).

2.3.2: Definition: If \( p \) is a formula and \( \Theta \) is a substitution, the application of \( \Theta \) to \( p \), denoted \( p\Theta \), is the formula defined by:

\[
\begin{align*}
p\Theta &= \begin{cases} 
q & \text{if } \text{ord}(p) = 0, \text{ and } (p,q) \in \Theta \\
p & \text{if } \text{ord}(p) = 0 \\
& \text{and } p \text{ is not a replaced variable of } \Theta \\
f(p_1\Theta, \ldots, p_n\Theta) & \text{if } p = f(p_1, \ldots, p_n)
\end{cases}
\end{align*}
\]

If \( E \) is a set of formulae, \( \mathcal{E} \) is a set of sets of formulae, and \( \Theta \) is a substitution, we define:

\[
\begin{align*}
E\Theta &= \{ p\Theta \mid p \in E \} \\
\mathcal{E}\Theta &= \{ E\Theta \mid E \in \mathcal{E} \}
\end{align*}
\]

called the application of \( \Theta \) to \( E \), and the application of \( \Theta \) to \( \mathcal{E} \). If \( X \) is a formula, set of formulae or set of sets of formulae, and \( \Theta \) is a substitution, we call \( X\Theta \) an instance of \( X \), and call \( X \) a generalisation of \( X\Theta \). Note that since \( X\emptyset = X \), \( X \) is both an instance and a generalisation of itself.
2.3.3: Definition: If $\theta$ and $\gamma$ are substitutions, the composition of $\theta$ with $\gamma$, denoted $\theta \gamma$, is the substitution:

$$\{(v, p) \mid (v, p) \in \theta \text{ and } v \neq p\gamma\}$$

$$\cup \{(v, p) \mid (v, p) \in \gamma \text{ and } v \text{ is not a replaced variable of } \theta\}$$

Clearly $\theta \emptyset = \emptyset \theta = \emptyset$ for any substitution $\theta$. It is easy to show that $\cdot$ is associative, and that $p(\theta \gamma) = (p\theta)\gamma$ for all formulae $p$ and substitutions $\theta$ and $\gamma$.

2.3.4: Definition: A substitution $\theta$ unifies a set of formulae $E$ if and only if $E\theta$ contains one element. In this case, $E$ is said to be unifiable, and $\theta$ is a unifier for $E$. $\theta$ is called a most general unifier (mgu) for $E$ if and only if for every unifier $\gamma$ for $E$, there is a substitution $\beta$ such that $\gamma = \theta \beta$.

We extend the notion of unifiability to sets of sets of formulae.

2.3.5: Definition: If $\mathcal{S}$ is a set of sets of formulae and $\theta$ is a substitution, $\theta$ unifies $\mathcal{S}$ if and only if $\theta$ unifies $E$ for each $E \in \mathcal{S}$. We define "unifiable", "unifier", and "most general unifier" exactly as in 2.3.4.

The fact that every unifiable set has at least one mgu is clear from the existence of several unification algorithms.
If $E$ is a unifiable set of formulae, we denote by $\text{mgu}(E)$, some $\text{mgu}$ of $E$. We use this notation also for sets of sets of formulae. Note that if $E$ is a set of formulae, then $\text{mgu}(E) = \text{mgu}(E)$.

2.3.6: **Lemma:** If $X_1$ and $X_2$ are both sets of formulae, or both sets of sets of formulae, then:

(i) $X_1 \cup X_2$ is unifiable if and only if $X_1$ is unifiable and $X_2 \text{mgu} X_1$ is unifiable.

(ii) If $X_1 \cup X_2$ is unifiable:

$$\text{mgu}(X_1 \cup X_2) = \text{mgu}(X_1) \cdot \text{mgu}(X_2 \text{mgu} X_1)$$

**Proof:**

(i) (a) If $X_1 \cup X_2$ is unifiable, let $\theta$ be any unifier for $X_1 \cup X_2$. $\theta$ unifies $X_1$, so that by the definition of $\text{mgu}$, $\theta = \text{mgu} X_1 \cdot \beta$ for some substitution $\beta$. But $\theta$ unifies $X_2$, so that $\beta$ unifies $X_2 \text{mgu} X_1$. Therefore $X_1$ and $X_2 \text{mgu} X_1$ are unifiable.

(b) Suppose $X_1$ and $X_2 \text{mgu} X_1$ are unifiable, and let $\theta$ be a unifier for $X_2 \text{mgu} X_1$. Now $\text{mgu} X_1$ unifies $X_1$, so that $\text{mgu} X_1 \cdot \theta$ unifies $X_1$; also $\theta$ unifies $X_2 \text{mgu} X_1$ so that $\text{mgu} X_1 \cdot \theta$ unifies $X_2$. Hence $\text{mgu} X_1 \cdot \theta$ unifies $X_1 \cup X_2$ so that $X_1 \cup X_2$ is unifiable.

(ii) If $X_1 \cup X_2$ is unifiable, than by part (i), $\text{mgu}(X_2 \text{mgu} X_1)$ exists. Now $\text{mgu} X_1$ unifies $X_1$, so that $\text{mgu} X_1 \cdot (X_2 \text{mgu} X_1)$ unifies $X_2$. Also:

$$X_2 \text{mgu} X_1 \cdot \text{mgu}(X_2 \text{mgu} X_1) = \text{mgu}(X_2 \text{mgu} X_1)$$
So \( \text{mgux}_4 \cdot \text{mgux}(x_2 \cdot \text{mgux}_4) \) clearly unifies \( x_2 \). Therefore \( \text{mgux}_4 \cdot \text{mgux}(x_2 \cdot \text{mgux}_4) \) is a unifier for \( x_1 \cup x_2 \).

Suppose \( \Theta \) is a unifier for \( x_1 \cup x_2 \), then since \( \Theta \) unifies \( x_4 \):

\[
\Theta = \text{mgux}_4 \cdot \beta \text{ for some substitution } \beta
\]

But \( \Theta \) unifies \( x_2 \), so that \( \beta \) unifies \( x_2 \cdot \text{mgux}_4 \).

\[
\therefore \beta = \text{mgux}(x_2 \cdot \text{mgux}_4) \cdot \alpha \text{ for some substitution } \alpha
\]

\[
\therefore \Theta = \text{mgux}_4 \cdot \text{mgux}(x_2 \cdot \text{mgux}_4) \cdot \alpha
\]

Therefore \( \text{mgux}_4 \cdot \text{mgux}(x_2 \cdot \text{mgux}_4) \) is an \( \text{mgux} \) for \( x_1 \cup x_2 \).

\[\Box\]

2.3.7: Definition: A substitution \( \{(v_1,u_1),\ldots,(v_n,u_n)\} \) is called a renaming if \( u_1,\ldots,u_n \) are distinct variables, and:

\[
\{u_1,\ldots,u_n\} \cap \{v_1,\ldots,v_n\} = \emptyset
\]

If \( \gamma \) is a renaming, then \( \gamma^{-1} \) is the renaming \( \{(u,v)\mid (v,u) \in \gamma\} \). Clearly, if \( \gamma \) is a renaming, then \( \gamma \circ \gamma = \gamma \), \( (\gamma^{-1})^{-1} = \gamma \), and \( \gamma \circ \gamma^{-1} = \gamma^{-1} \).

2.3.8: Definition: If \( p \) is a formula and \( \gamma \) is a renaming, \( p \gamma \) is called a variant of \( p \) if and only if no variable is a subformula of both \( p \) and \( p \gamma \). If \( E \) is a set of formulae and \( \mathcal{E} \) is a set of sets of formulae, \( E \gamma \) is a variant of \( E \) if and only if \( p \gamma \) is a variant of \( p \) for all \( p \in E \); and \( \mathcal{E} \gamma \) is a variant of \( \mathcal{E} \) if and only if \( E \gamma \) is a variant of \( E \) for all \( E \in \mathcal{E} \). If \( p \in E \in \mathcal{E} \) it is clear that \( \mathcal{E} \gamma \) is a variant of \( \mathcal{E} \) implies that \( E \gamma \) is a variant of \( E \) implies that \( p \gamma \) is a
variant of \( p \). Also, if \( X \) is a formula, set of formulae, or set of sets of formulae, then \( X \gamma^{-1} = X \) so that \( (X \gamma) \gamma^{-1} = X \gamma \gamma^{-1} = X \gamma^{-1} = X \): \( X \) is therefore a variant of \( X \gamma \).

2.4: **Predicate calculus**

We describe here the quantifier-free conjunctive normal form of first-order predicate calculus usually used by mechanical theorem-provers.

2.4.1: **Definition:** A clause is a finite set of literals.

The empty clause is denoted by \( \emptyset \).

2.4.2: **Definition:** A valuation is a function \( \models \) from the set of all variable-free literals into the set \( \{T,F\} \) such that:

\[
\models(L) = T \text{ iff } \models(\neg L) = F
\]

We extend the domain of every valuation \( \models \) to the set of all clauses, as follows:

(i) If \( \mathcal{C} \) is a variable-free clause:

\[
\models(\mathcal{C}) = T \text{ iff } \models(L) = T \text{ for some } L \in \mathcal{L}
\]

(ii) If \( \mathcal{C} \) is a clause which is not variable-free:

\[
\models(\mathcal{C}) = T \text{ iff } \models(\mathcal{C}\theta) = T \text{ for all variable-free instances } \theta \text{ of } \mathcal{C}
\]
2.4.3: Definition: A valuation \( \Sigma \) is said to satisfy a clause \( \phi \) if and only if \( \Sigma(\phi) = T \). We also say that \( \Sigma \) is a model for \( \phi \). Similarly, a valuation \( \Sigma \) is said to satisfy, or to be a model for a set of clauses \( \mathcal{S} \) if and only if \( \Sigma(\phi) = T \) for all \( \phi \in \mathcal{S} \).
CHAPTER 3

Deduction Plans

In this chapter we present a deduction system which relies on the construction of certain directed graphs called "deduction plans". Before defining these graphs formally, we give an informal description of their structure.

The underlying structure of a deduction plan for a set $\mathcal{S}$ of clauses is a rooted tree the root of which is a special vertex called TOP. Every vertex other than TOP is a variant of some literal in some clause of $\mathcal{S}$. This underlying rooted tree corresponds exactly to a linear resolution deduction from $\mathcal{S}$ in which the only inference rule is binary resolution. The rule used in building such a tree is called "replacement". The following example illustrates such a rooted tree and the corresponding linear resolution deduction.
3.0: Example: Consider the set of clauses:

\[ \mathcal{G} = \{ [P(x,y), P(x,f(x)), -Q(x)] , \\
\quad [Q(x), R(x,a), -S(x)] , \\
\quad [S(x), Q(x)] , \\
\quad [-R(x,y), -S(y)] , \\
\quad [-P(a,f(a))] \} \]

where a is a constant.

Figure 3.1 is a linear binary resolution deduction from \( \mathcal{G} \), and figure 3.2 illustrates the corresponding plan. Note that the subgraph indicated by the dotted line corresponds to the top clause in the linear deduction, and the arcs labelled "SUB" connect the root vertex with the literals of the top clause. SUB stands for "subproblem" and indicates literals which must be removed by resolution. If a subproblem has no arcs leaving it, then that subproblem is said to be "open": that is, it has yet to be removed by resolution. Each arc labelled "REPL" indicates one application of the replacement rule, and shows that we have selected a variant \( \mathcal{G} \) of some clause from \( \mathcal{G} \), and have performed a binary resolution on the subproblem at the tail of the arc, using the literal of \( \mathcal{G} \) which appears as the head of the arc. The remaining literals of \( \mathcal{G} \) are then introduced as new subproblem vertices at the head of new SUB arcs: the tail of each of these new SUB arcs is the head of the new REPL arc. In figure 3.2 the REPL arcs are numbered, indicating the order in which the tree is constructed: this
A binary linear resolution deduction from the set of clauses of example 3.0. The substitution applied during each application of the resolution rule is noted beside the appropriate branch.

Figure 3.1
A plan for the set of clauses of example 3.0 corresponding to the linear resolution of figure 3.1.

Figure 3.2
order corresponds exactly to the order in which deductions are performed in the linear resolution deduction of figure 3.1. When a subproblem becomes the tail of an arc it is said to be "closed". Note that a vertex at the head of a REPL arc is not a subproblem.

In building this tree we have not applied the unifying substitutions as in the linear deduction. Instead, a record is kept of the formulae which must be unified in order to validate the construction. In chapter 4 we describe how this is done.

At each stage during the construction of this tree, the set of open subproblems corresponds to the clause deduced by the equivalent linear resolution. If at some stage there were no open subproblems left, then in the corresponding linear resolution, the empty clause would have been obtained, showing the set $\mathcal{F}$ to be unsatisfiable. The object, therefore, when using plans to prove unsatisfiability, is to construct a plan with no open subproblems; such a plan is said to be "closed".

Unfortunately, binary resolution is not complete; that is, for some unsatisfiable set of clauses it will not produce the empty clause. Similarly, the replacement rule in plan construction is not complete. The completeness of linear resolution can be assured by a variety of methods, and we have plan construction rules which simulate each of these. Two of these rules are "factoring" and "reduction".
As with replacement, factoring and reduction add new arcs to the plan; however, unlike replacement they introduce only one new arc, and add no new vertices. Factoring is analogous to the familiar factoring rule of resolution, and is applied by selecting an open subproblem of the plan, and directing an arc labelled "FACT" from it to another subproblem of the plan: eventually, of course, these subproblems must be shown to be unifiable. Factoring in plans differs from factoring in resolution in that when a FACT arc is added, the subproblem at its head need not be open. Reduction also has an analogy in linear resolution, namely, the reduction rule of model-elimination [24,25,27] and SL-resolution [23]. This corresponds to the familiar proof technique "reductio ad absurdum" in which a particular hypothesis is shown to imply its negation. To apply reduction, we select an open subproblem \( u \) and direct an arc labelled "RED" from it to some subproblem \( v \) which is "above" \( u \) in the underlying rooted tree: that is, there must be a walk from \( v \) to \( u \) consisting entirely of REPL and SUB arcs. For the reduction to apply, we must verify that \( u \) and the negation of \( v \) are unifiable. As for REPL arcs, subproblems at the tail of RED and FACT arcs are said to be closed.

In figure 3.3, we illustrate a closed plan obtained from the plan of figure 3.2 by applying these two rules.

Note that plans containing RED and FACT arcs are not trees.
A closed plan for the set of clauses of example 3.0.

Figure 3.3.
There is one further rule for constructing plans called "ancestor replacement", which is a variation of the simple replacement rule we have already described. This allows us to close a subproblem by replacement using a clause deduced earlier in the proof, rather than a clause from \( \mathcal{F} \). In order to apply this we must have some means of extracting such clauses from the plan. This leads us to the definition of a special kind of subgraph of a plan called a "subplan". The important feature of subplans is that, although they cannot necessarily be constructed from \( \mathcal{F} \) using the rules we have described, they have the same underlying rooted tree structure as plans. That is, each subplan contains the vertex TOP, the REPL and SUB arcs of the subplan form a rooted tree, and if a REPL arc is in the subplan, then so are all the SUB arcs associated with that REPL arc. To apply ancestor replacement, some subplan is extracted from the plan, and the set of open subproblems of this subplan is used as though it were a clause in \( \mathcal{F} \) for closing a subproblem of the plan by replacement.

We present now the formal definition of deduction plans, followed by the proofs of soundness and completeness of the various deduction systems based on them.

3.1: Definition: If \( \mathcal{F} \) is a set of clauses, a deduction plan for \( \mathcal{F} \) is a digraph \( G \), where:
(a) \( E(G) \) is divided into four mutually disjoint sets
\( \text{REPL}(G), \text{SUB}(G), \text{RED}(G) \) and \( \text{FACT}(G) \),

(b) \( \text{TOP} \in V(G) \), where TCP is a special symbol which does not occur in \( \mathfrak{S} \),

(c) \( G \) is constructed recursively, using a finite number of applications of the rules defined below (3.1.6)

Before defining the rules for constructing deduction plans, we must digress with the following remarks and definitions.

We will henceforth refer to deduction plans for \( \mathfrak{S} \) as "plans for \( \mathfrak{S} \)" , or when the context ensures that no ambiguity is likely, simply as "plans".

**3.1.1: Definition:** If \( G \) is a plan, and \( v_1, v_2 \in V(G) \), then \( v_1 \) is said to be a direct ancestor of \( v_2 \) if and only if there is a walk from \( v_1 \) to \( v_2 \) with no arcs in \( \text{FACT}(G) \cup \text{RED}(G) \). Also, \( v_2 \) is called a direct descendant of \( v_1 \).

**3.1.2:** If \( G \) is a plan, and \( H \) is any subgraph of \( G \), we can meaningfully refer to \( \text{SUB}(H), \text{FACT}(H), \text{RED}(H) \) and \( \text{REPL}(H) \). Henceforth, in any subgraph \( H \) of a plan \( G \), \( \text{SOL}(H) \) will be used as an abbreviation for \( \text{FACT}(H) \cup \text{RED}(H) \cup \text{REPL}(H) \).
3.1.3: Definition: If $G$ is a plan for $\mathcal{G}$ and $H$ is a subgraph of $G$, then $H$ is a **subplan of $G$ for $\mathcal{G}$** if and only if for every $x \in V(H)$:

(i) $(x,y) \in \text{SUB}(G) \Rightarrow (x,y) \in E(H)$

(ii) $(y,x) \in \text{FLP}(G) \Rightarrow (y,x) \in E(H)$

(iii) if $y$ is a direct ancestor of $x$ in $G$

then $y \in V(H)$.

We will say that $H$ is a **subplan for $\mathcal{G}$** if there exists a plan $G$ for $\mathcal{G}$ and $H$ is a subplan of $G$ for $\mathcal{G}$. When the context ensures that there will be no ambiguity, we will say $H$ is a **subplan of $G$**, or simply $H$ is a subplan.

Note that every plan is a subplan of itself. It is not the case though that every subplan is a plan: we will produce an example to illustrate this following the definition of the plan-construction rules. However, if $H$ is a subplan for $\mathcal{G}$ but not a plan for $\mathcal{G}$, it can be shown that there exists a set $\mathcal{S}_4$ of clauses for which $H$ is a plan, such that $\mathcal{S}_4$ is satisfiable if and only if $\mathcal{G}$ is satisfiable.

3.1.4: Definition: If $H$ is a subplan, the **top clause** of $H$ is the set $\{ v \mid (TCP,v) \in E(H) \}$.

3.1.5: Definition: If $H$ is a subplan, the set:

$\{ v \mid \exists x \in V(H) \text{ such that } (x,v) \in \text{SUB}(H) \}$
is called the set of subproblems of $H$, and is denoted $s(H)$. Also, if $v$ is a subproblem of $H$ and $\exists (v,y) \in E(H)$, then $v$ is said to be closed. A subproblem which is not closed is open. The sets of closed and open subproblems are denoted $cs(H)$ and $os(H)$ respectively.

3.1.6: Plan construction rules

We now define the rules for constructing plans.

3.1.6.1: Basis

If $\emptyset$ is a variant of any clause in $\mathcal{S}$, then $G$ is a plan, where $G$ is defined by:

$$V(G) = \{ TCP \} \cup \emptyset$$
$$SUB(G) = \{ (TCP,l) | l \in \emptyset \}$$
$$SCL(G) = \emptyset$$

$G$ is called a basic plan. A pictorial representation of a basic plan is shown in figure 3.4.

3.1.6.2: Induction

Rule (1): Replacement

(A) Simple replacement

If $G$ is a plan and:

(a) $v \in os(G)$,
(b) $l \in \emptyset$,
(c) $\emptyset$ is a variant of some clause in $\mathcal{S}$ and contains no variables which occur in any vertices of $G$;

3.1.6.1
A pictorial representation of a basic plan with top clause \( \{l_1, \ldots, l_k\} \).

**Figure 3.4**

then \( G' \) is a plan, where \( G' \) is defined by:

\[
\begin{align*}
V(G') &= V(G) \cup \varnothing \\
\text{REPL}(G') &= \text{REPL}(G) \cup \{(v,l)\} \\
\text{SUB}(G') &= \text{SUB}(G) \cup \{(l,m) \mid m \in \varnothing - \{l\}\} \\
\text{FACT}(G') &= \text{FACT}(G) \\
\text{RED}(G') &= \text{RED}(G)
\end{align*}
\]

(E) Ancestor replacement

If \( G \) is a plan, and:

(a) \( v \in \text{os}(G) \),
(b) \( l \in \mathcal{C} \),

(c) \( \mathcal{C} = os(H)\gamma \), where \( H \) is a subplan of \( G \), and \( \gamma \) is a renaming such that for every \( x \in V(H) \), \( x\gamma \) is a variant of \( x \);

then \( G' \) is a plan, where \( G' \) is defined as for simple replacement. We also define: In both types of replacement, we say that \( v \) is replaced through \( l \), and that \( v \) is replaced by \( \mathcal{C} - \{l\} \). Replacement may be represented pictorially as in figure 3.5.

Note that if some literal \( m \in \mathcal{C} \) contains no variables, then it is possible that the plan on which the replacement is performed may already contain a vertex corresponding to the literal \( m \). The set of vertices, however, should be regarded as a set of literal occurrences rather than as a set of literals: the replacement then introduces a new vertex corresponding to the new occurrence of \( m \).

**Rule (2): Reduction**

If \( G \) is a plan and:

(a) \( u \in os(G) \),

(b) \( v \in cs(G) \) is a direct ancestor of \( u \),

(c) if \( (x,y) \in \text{FACT}(G) \), and either \( y = u \) or there is a walk from \( y \) to \( u \) which does not pass through \( v \) and contains no arcs in \( \text{RED}(G) \), then \( v \) is a direct ancestor of \( x \);

then \( G' \) is a plan, where \( G' \) is defined by:
A representation of a plan $G'$ obtained from $G$ by replacement.

**Figure 3.5**

\[
V(G') = V(G)
\]

\[
RED(G') = RED(G) \cup \{(u,v)\}
\]

\[
FACT(G') = FACT(G)
\]

\[
REPL(G') = REPL(G)
\]

\[
SUB(G') = SUB(G)
\]
We say that \textit{u} is \textbf{reduced to} \textit{v}. Reduction may be represented pictorially as in figure 3.6.

\textbf{Rule (3): Factoring}

Factoring, like replacement, divides into two cases.

(A) Simple factoring

If \( G \) is a plan and:

(a) \( x \in \text{os}(G) \),
(b) \( y \in \text{os}(G) - \{x\} \);

then \( G' \) is a plan, where \( G' \) is defined by:

\[
\begin{align*}
V(G') &= V(G) \\
\text{FACT}(G') &= \text{FACT}(G) \cup \{(x,y)\} \\
\text{RED}(G') &= \text{RED}(G) \\
\text{REPL}(G') &= \text{REPL}(G) \\
\text{SUB}(G') &= \text{SUB}(G)
\end{align*}
\]

(E) Back factoring

If \( G \) is a plan and:

(a) \( x \in \text{os}(G) \),
(b) \( y \in \text{cs}(G) \),
The RED arc can be constructed, provided that the walk (1) exists, demonstrating that $v$ is a direct ancestor of $u$; and provided that, if the items marked (2) exist, then the walk (3) exists, demonstrating that $v$ is a direct ancestor of $x$.

Figure 3.6
(c) every walk from y to x contains at least one arc in RED(G);

and if \( u, v \in RED(G) \) and both of the following conditions hold:

(1) either \( y = u \) or there is a walk from \( y \) to \( u \) containing no arcs of RED(G) and not passing through \( v \)

(2) for some \( w \in V(G) \), either \( w = x \) or there is a walk from \( w \) to \( x \) containing no arcs of RED(G) and not passing through \( v \)

then \( v \) is a direct ancestor of \( w \)
then \( G' \) is a plan, where \( G' \) is defined as for simple factoring. In both types of factoring, we say that \( x \) is factored to \( y \). A pictorial representation of backfactoring is shown in figure 3.7

We now present an example to illustrate the construction of a plan.

3.1.7: Example: Let \( \$ \) be the set of clauses:

\[
\{ \{ P(x), P(y), -P(f(y)) \},
\{ P(w), Q(w, b), P(f(w)) \},
\{ -Q(f(z), z), P(z), P(f(f(z))) \} \}
\]

where \( b \) is a constant. Figure 3.8 illustrates a plan \( G \) for \( \$ \). Each arc of SCI(G) is labelled with the name of the
The FACT arc can be constructed provided that, if the walk (1) exists, it contains at least one arc of RED(G); and provided that, if the items (2), (3) and (4) exist, then v is a direct ancestor of w, as demonstrated by the walk (5).

Figure 3.7
subset of SOL(G) to which it belongs, and with an integer. The integer labels indicate the order of construction of G. Note that the EEFL arc numbered 3 is an ancestor replacement using the boxed clause. G has only one open subproblem, which is the vertex with a double outline.

3.2: Comments, notation and some preliminary results

Every vertex of a subplan, except TCP, has indegree at least 1; TCP has indegree 0. Open subproblems have outdegree 0 and closed subproblems have outdegree 1. In fact, closed subproblems are former open subproblems each of which has been closed by one application of a rule.

In every subplan, there are vertices which are not subproblems: TCF is one of these, and the others are those vertices through which subproblems are replaced.

3.2.1: Definition: If G is a plan, then there exists a sequence of plans \( D = (G_0, \ldots, G_n) \) such that \( G_0 \) is basic, \( G_n = G \), and for \( i \leq n \), \( G_i \) is derived from \( G_{i-1} \) by one application of a rule. Such a sequence is called a derivation of G of length n.

Since one application of a rule closes exactly one subproblem, and adds exactly one arc to SCL(G), we note that \( |cs(G)| = |SOL(G)| = n \), and that every derivation of G has the same length. We note also that every derivation of G
A plan $G$ for the set $\mathcal{G}$ of clauses of example 3.1.7.

Figure 3.8
begins with the same basic plan. Consequently, for any plan $G$ we can specify a derivation either as an ordering of $cs(G)$, or equivalently, as an ordering of $SCL(G)$. This leads to some notational devices which we will use frequently, despite their initially ambiguous appearance: namely, to define some new plan $G_2$ by applying a rule to $G_1$, we may write:

$$cs(G_2) = cs(G_1) \cup \{x\}$$

or $$SCL(G_2) = SCL(G_1) \cup \{(x,y)\}$$

or $$FACT(G_2) = FACT(G_1) \cup \{(x,y)\}$$

...etc.

when it is obvious from the context exactly how $x$ is to be closed, and exactly what vertices are to be added.

We now present some general results concerning the structure of plans, and the relationships between plans, subplans and the construction rules.

3.2.2: Lemma: If $G$ is a plan, there is no closed walk in $G$ with all its arcs in $REPL(G) \cup SUB(G)$.

Proof: Let $D = (G_0, \ldots, G_n)$ be a derivation of $G$. We use induction on this derivation.

Basis: $G_0$ is a basic plan, and therefore has no closed walks.

Induction: Suppose $G_i$ has no closed walks with arcs in $REPL(G_i) \cup SUB(G_i)$ only.
(i) Suppose $G_{i+1}$ is obtained from $G_i$ by factoring or reduction, then:

$$\text{REFL}(G_{i+1}) \cup \text{SUB}(G_{i+1}) = \text{REFL}(G_i) \cup \text{SUB}(G_i)$$

So if $G_{i+1}$ has a closed walk of the specified type, the same walk exists in $G_i$, contrary to hypothesis.

(ii) Suppose $G_{i+1}$ is obtained from $G_i$ by replacing $y \in \text{os}(G_i)$ through $x$ by $x_1, \ldots, x_m$. If $G_{i+1}$ has a closed walk of the specified type, this walk is either in $G_i$, contrary to hypothesis, or contains one of the new arcs $(y,x),(x,x_1), \ldots, (x,x_m)$. Consequently, the walk must pass through $x$ and therefore must pass through $x_j$ for some $j \in \{1, \ldots, m\}$. But for each $j \in \{1, \ldots, m\}$, $x_j$ has outdegree 0, so no walk through $x_j$ is closed.

\[\Box\]

3.2.3: \textbf{Lemma}: If \(w\) is any closed subproblem of a plan \(G\), then \(G'\) is a subplan of \(G\), where \(G'\) is defined by:

\[
V(G') = V(G) - \{u \mid u \text{ is a direct ancestor of } w\}
\]

\[
E(G') = E(G) - \{(u,v) \mid (u,v) \in E(G) \text{ and either } u = w \text{ or } u \text{ is a direct descendant of } w \text{ or } v \text{ is a direct descendant of } w \}
\]

\textbf{Proof}:

(1) We show first that \(G'\) is a subgraph of \(G\). Since \(V(G') \subseteq V(G)\) and \(E(G') \subseteq E(G)\), we need only show that \(G'\) is a graph; that is, that \(E(G') \subseteq V(G') \times V(G')\).
Suppose the contrary: then there exists \((x, y) \in E(G')\) such that either \(x \notin V(G')\) or \(y \notin V(G')\). Now if \(x \notin V(G')\), then \(x\) is a direct descendant of \(w\), so that \((x, y) \notin E(G')\), which is a contradiction. We obtain a similar contradiction by assuming \(y \notin V(G')\).

\[ \therefore G' \text{ is a subgraph of } G. \]

(2) Suppose now that \(G'\) is not a subplan of \(G\): we have three cases:

(i) Suppose for some \(x \in V(G')\) that:

\[(x, y) \in SUB(G) - SUB(G')\]

Since \((x, y) \notin SUB(G')\)

either \(x = w\), which is impossible since \(w\) is a subproblem and \(x\) is not.

or \(x\) is a direct descendant of \(w\), in which case \(x \notin V(G')\); a contradiction.

or \(y\) is a direct descendant of \(w\), so that \(x\) is also a direct descendant of \(w\), and again \(x \notin V(G')\); a contradiction.

(ii) Suppose for some \(x \in V(G')\) that:

\[(y, x) \in REPL(G) - REPL(G')\]

Since \((y, x) \notin REPL(G')\)

either \(y = w\), so that \(x\) is a direct descendant of \(w\)

or \(y\) is a direct descendant of \(w\), so that \(x\) is also a direct descendant of \(w\)

or \(x\) is a direct descendant of \(w\)

In each case \(x \notin V(G')\); a contradiction.
(iii) Finally, suppose for some $x \in V(G')$ that $y$ is a direct ancestor of $x$ and $y \notin V(G')$. In this case, $y$ must be a direct descendant of $w$, so that $x$ is a direct descendant of $w$, and therefore $x \notin V(G')$; a contradiction.

Therefore $G'$ is a subplan of $G$.

In the following lemma, we show how the structure of plans is affected by the conditions on the factoring and reduction rules.

3.2.4: Lemma: If $G$ is a plan, $(u,v) \in RED(G)$, and $(w,z) \in FACT(G)$, then:

(i) $v$ is a direct ancestor of $u$ in $G$,

(ii) every walk from $z$ to $w$ in $G$ contains an arc in $RED(G)$,

(iii) if either $z = u$ or there is a walk in $G$ from $z$ to $u$ which does not pass through $v$ and has no arcs in $RED(G)$, then $v$ is a direct ancestor of $w$ in $G$.

Proof: Let $D = (G_0, \ldots, G_n)$ be a derivation of $G$.

(i) For some $k$, where $0 < k \leq n$, we have:

$$cs(G_k) = cs(G_{k-1}) \cup \{u\}$$

so that $v$ is a direct ancestor of $u$ in $G_{k-1}$, and hence in $G$.

(ii) Suppose there is a walk from $z$ to $w$ in $G$ containing no arcs of $RED(G)$. Let $(x,y)$ be the last arc on this
walk to be constructed in the derivation \( D \). Then for some \( k \), where \( 0 < k \leq n \):

\[
SCL(G_k) = SCL(G_{k-1}) \cup \{(x,y)\}
\]

Suppose \( x \) is replaced through \( y \) by \( y_1, \ldots, y_m \); then \( y_i \) lies on the walk for some \( i \) such that \( 1 \leq i \leq m \), and \( y_i \) is an open subproblem of \( G_k \). This is a contradiction since \( x \) is the last subproblem on the walk to be closed. Therefore \( x \) must be closed by factoring, since there are no arcs of \( \text{RED}(G) \) on the walk. Since \( (x,y) \) is the last arc on the walk to be constructed, all the other arcs on the walk are in \( E(G_{k-1}) \), so there is a walk in \( G_{k-1} \) from \( y \) to \( x \) containing no arcs of \( \text{RED}(G) \). This contradicts the condition (c) on the factoring of \( x \) to \( y \) in \( G_{k-1} \).

Therefore every walk from \( z \) to \( w \) in \( G \) must contain an arc of \( \text{RED}(G) \).

(iii) Suppose either \( z = u \) or there is a walk from \( z \) to \( u \) in \( G \) containing no arcs of \( \text{RED}(G) \). Then in either case there is a walk from \( w \) to \( u \), containing no arcs of \( \text{RED}(G) \). We now have two cases to consider:

(a) Suppose \( u \) is closed in the derivation \( D \) after all other subproblems on the walk from \( w \) to \( u \) have been closed. Then for some \( k \), where \( 0 < k \leq n \):

\[
SCL(G_k) = SCL(G_{k-1}) \cup \{(u,v)\}
\]

and the walk from \( w \) to \( u \) containing no arcs of \( \text{RED}(G) \) exists in \( G_{k-1} \). So by the conditions on
the reduction of \( u \) to \( v \) in \( G_{k-1} \), \( u \) must be a direct ancestor of \( w \) in \( G_{k-1} \), and hence in \( G \).

(b) Suppose \( x \neq u \) is the last subproblem in the walk from \( w \) to \( u \) which is closed in the derivation \( D \). Let \((x, y)\) be the arc in \( SCL(G) \) constructed in this closure. Then for some \( k \), where \( 0 < k \leq n \):

\[
SCL(G_k) = SCL(G_{k-1}) \cup \{(x, y)\}
\]

Suppose \( x \) is replaced through \( y \) by \( y_1, \ldots, y_m \); then \( y_1 \) lies on the walk from \( w \) to \( u \), where \( 1 \leq i \leq m \), and \( y_1 \) is an open subproblem of \( G_k \). This contradicts the fact that \( x \) is the last subproblem on the walk to be closed. Therefore \( x \) must be closed by factoring, since there are no arcs of \( RED(G) \) on the walk, and \( x \neq u \). Now since \((x, y)\) is the last arc on the walk to be constructed, all the other arcs on the walk are in \( E(G_{k-1}) \), so in \( G_{k-1} \) both of the following hold:

1. Either \( x = w \) or there is a walk from \( x \) to \( w \) containing no arcs of \( RED(G_{k-1}) \) and not passing through \( v \)
2. Either \( y = u \) or there is a walk from \( y \) to \( u \) containing no arcs of \( RED(G_{k-1}) \) and not passing through \( v \)

So by the conditions on the factoring of \( x \) to \( y \) in \( G_{k-1} \), \( u \) must be a direct ancestor of \( w \) in \( G_{k-1} \), and hence in \( G \).
3.2.5: **Corollary:** Suppose \( G_k \) is a subplan of \( G \), \( G_k \) is a plan, \((x,y) \in \text{FACT}(G), (u,v) \in \text{RED}(G)\), and \( x, u \in \text{cs}(G_k) \).

(i) If \( G_{k+1} \) is the subplan of \( G \) defined by:

\[ \text{SCL}(G_{k+1}) = \text{SCL}(G_k) \cup \{(x,y)\} \]

then \( G_{k+1} \) is a plan provided that \( y \in s(G_k) \).

(ii) If \( G_{k+1}^\prime \) is the subplan of \( G \) defined by:

\[ \text{SOL}(G_{k+1}^\prime) = \text{SCL}(G_k) \cup \{(u,v)\} \]

then \( G_{k+1}^\prime \) is a plan.

**Proof:** Lemma 3.2.4 ensures that the conditions for closing either \( x \) or \( u \) are satisfied in \( G_k \).

\[ \square \]

3.2.6: **Lemma:** If \( G \) and \( G' \) are plans, and \( G' \) is a subplan of \( G \), then every derivation \( G_0, \ldots, G_m \) of \( G' \) may be extended to a derivation \( G_0, \ldots, G_n \) of \( G \), where \( n \geq m \).

**Proof:** Suppose \( G_m \neq G \). We show that there exists a plan \( G_{m+4} \) which is a subplan of \( G \), and is derived from \( G_m \) by one application of a rule.

Since \( G_m \neq G \), the set \( \text{cs}(G_m) \cap \text{cs}(G) \) is not empty. Let \( x_1, \ldots, x_k \) be the members of this set in the order of their closure in some derivation \( D \) of \( G \). We define \( G_{m+4} \) by:

\[ \text{cs}(G_{m+4}) = \text{cs}(G_m) \cup \{x_1\} \]

It remains to prove that \( G_{m+4} \) is a plan. We have four cases:

(a) Suppose \( x_1 \) is closed by simple replacement in \( G \). Then the conditions for closing \( x_1 \) are trivially satisfied in \( G_m \), so that \( G_{m+4} \) is a plan.
(b) Suppose \( x_i \) is closed by reduction, then corollary 3.2.5 guarantees that \( G_{m+1} \) is a plan.

(c) Suppose \( x_1 \) is closed by factoring to \( y \) in \( G \).

Suppose \( y \notin V(G_m) \), then there must exist some \( x_1 \in \text{os}(G_m) \) such that \( x_1 \) is a direct ancestor of \( y \), and \( x_1 \) is closed by replacement in \( G \). But \( x_1 \) must be closed before \( x_4 \) in every derivation of \( G \), in particular in \( D \), contrary to the ordering imposed on \( \text{os}(G_m) \cap \text{cs}(G) \). Therefore \( y \in V(G_m) \), so corollary 3.2.5 guarantees that \( G_{m+1} \) is a plan.

(d) Suppose \( x_4 \) is closed by ancestor replacement using some subplan \( H \) of \( G \). We must show that \( H \) is a subplan of \( G_m \).

Suppose the contrary, then we have two cases:

(i) \( \exists y \in V(H) - V(G_m) \)

In this case, \( \exists x_i \in \text{os}(G_m) \) such that \( x_i \) is a direct ancestor of \( y \) and is closed by replacement in \( G \). But \( x_i \) is closed in \( H \), and therefore is closed before \( x_4 \) in any derivation of \( G \). This contradicts the ordering imposed on \( \text{os}(G_m) \cap \text{cs}(G) \).

(ii) \( \exists e \in E(H) - E(G_m) \), say \( e = (y,z) \)

If \( e \notin \text{SUB}(G) \), then \( z \notin V(H) - V(G_m) \), which we have already shown to be impossible.

If \( e \in \text{SCL}(G) \), then since \( V(H) \subseteq V(G_m) \) by case (i), \( y \) is an open subproblem of \( G_m \), and so \( y = x_i \) for some \( i > 1 \). But \( y \) must be closed before \( x_4 \) in any derivation of \( G \); this contradicts the ordering imposed on \( \text{os}(G_m) \cap \text{cs}(G) \).
H is therefore a subplan of \( G_m \), so the conditions for closing \( x_k \) by ancestor replacement are satisfied in \( G_m \).

Therefore \( G_{m+1} \) is a plan.

Since \( cs(G) \) is finite and \( |cs(G_{m+1})| > |cs(G_m)| \), a finite number of such extensions must eventually result in a derivation for \( G \).

\[ \]  

3.2.7: **Corollary:** A subgraph \( H \) of a plan \( G \) is a plan if and only if there is a derivation \((G_0, \ldots, G_n)\) of \( G \) such that \( H = G_m \) for some \( m \leq n \).

3.2.8: **Lemma:** If \( G \) is a plan such that \( FACT(G) \neq \emptyset \) and for all \((x,y) \in FACT(G)\), \( y \) is neither open, nor closed by reduction; then \( \exists (x,y) \in FACT(G) \) such that \( y \) is closed by replacement and no direct descendant of \( y \) is closed by factoring.

**Proof:** Suppose the contrary; that is:

(A) for all \((x,y) \in FACT(G)\),

either \( y \) is closed by factoring,

or \( y \) is closed by replacement and a direct descendant of \( y \) is closed by factoring.

We first prove that for any integer \( n \geq 1 \), there is a walk of length \( n \) in \( G \) such that the last arc in the walk is in \( FACT(G) \) and no arc of the walk is in \( RED(G) \). The proof is by induction on \( n \).
**Basis:** \( n=1 \). Fact is not empty, so there exists \((x_1, x_2) \in \text{FACT}(G)\). The walk from \(x_1\) to \(x_2\) consisting of this single arc has the required properties.

**Induction:** Suppose there exists a walk \((x_1, x_2), \ldots, (x_{n-1}, x_n)\) with the required properties. Now \((x_{n-1}, x_n) \in \text{FACT}(G)\) by the induction hypothesis; so by the hypothesis \((A)\), we have two cases:

- **either** \(x_n\) is closed by factoring to \(z\), say. Then \((x_1, x_2), \ldots, (x_{n-1}, x_n), (x_n, z)\) is a walk of length \(>n\) with the required properties.

- **or** \(x_n\) is closed by replacement, and some direct descendant \(w\) of \(x_n\) is closed by factoring to some \(z\). Therefore, there is a walk from \(x_n\) to \(w\) containing only arcs of \(\text{REPL}(G) \cup \text{SUB}(G)\), and a walk from \(w\) to \(z\) consisting of the single arc \((w, z) \in \text{FACT}(G)\). Appending these two walks to the end of the walk \((x_1, x_2), \ldots, (x_{n-1}, x_n)\), we obtain a walk of length \(>n\) with the required properties.

Since this holds for any integer, and \(V(G)\) is finite, there exists a walk in \(G\) of length \(|V(G)| + 1\) with no arcs in \(\text{RED}(G)\). Such a walk must encounter some vertex more than once; hence there is a closed walk in \(G\) containing no arcs of \(\text{RED}(G)\). By lemma 3.2.2, some arc \((x, y)\) on this walk is in \(\text{FACT}(G)\). Consequently, there exists \((x, y) \in \text{FACT}(G)\), and a walk from \(y\) to \(x\) with no arcs in \(\text{RED}(G)\). This contradicts lemma 3.2.3, thereby disproving hypothesis \((A)\), and establishing the result.

\(\Box\)
As mentioned above, not every subplan is necessarily a plan; this is illustrated as follows.

3-2-9: Example: Consider the plan $G_2$ for the set of clauses

$\{ \{v_4,v_2\}, \{v_3,v_4\} \}$, where $G_2$ has the derivation $(G_0,G_1,G_2)$ defined as follows:

$V(G_0) = \{ TCP, v_1, v_2 \}$

$SUB(G_0) = \{(TCP,v_1), (TCP,v_2)\}$

$V(G_1) = V(G_0) \cup \{v_3,v_4\}$

$SUB(G_1) = SUB(G_0) \cup \{(v_3,v_4)\}$

$REPL(G_1) = REPL(G_0) \cup \{(v_4,v_3)\}$

$V(G_2) = V(G_1) \cup \{v_2,v_6\}$

$SUB(G_2) = SUB(G_1) \cup \{(v_6,v_2)\}$

$REPL(G_2) = REPL(G_1) \cup \{(v_2,v_6)\}$

where $v_2$ and $v_6$ are corresponding variants of $v_2$ and $v_4$ respectively. Now $H$ is a subplan of $G_2$, where:

$V(H) = \{ TCP, v_1, v_2, v_4, v_2' \}$

$E(H) = \{(TCP,v_4), (TCP,v_2), (v_4,v_2'), (v_2,v_4)\}$

However, $H$ is not a plan. This is illustrated in figure 3.9
B is a subplan of $G_2$ but is not a plan.

**Figure 3.9**

As the reader may have realized, situations such as that described in the above example can arise only in the presence of the ancestor replacement rule. This is proved in the following lemma.

**3.2.10: Lemma:** If $G$ is a plan constructed using rules (1A), (2) and (3) only, then every subplan of $G$ is a plan.
**Proof:** Suppose $G_m$ is a subplan of $G$, where $m = |SOL(G_m)|$.
We will show that there exists a subplan $G_{m-1}$ such that $|SOL(G_{m-1})| = m-1$, and $G_m$ is a plan if $G_{m-1}$ is a plan. We consider two cases.

**Case (a):** If $EED(G_m) \cup FACT(G_m)$ is not empty, define $G_{m-1}$ by:

$$V(G_{m-1}) = V(G_m)$$

$$E(G_{m-1}) = E(G_m) - \{(x, y)\}$$

where $(x, y) \in EED(G_m) \cup FACT(G_m)$. Now $|SCL(G_{m-1})| = m-1$, and by corollary 3.2.5, if $G_{m-1}$ is a plan then $G_m$ is a plan.

**Case (b):** Suppose $EED(G_m) \cup FACT(G_m)$ is empty. Let $x_1, \ldots, x_k$ be the closed subproblems of $G_m$ in order of their closure in some derivation $D$ of $G$. Suppose $x_k$ is replaced through $y$ by $\emptyset - \{y\}$, where $\emptyset$ is a variant of some clause in $G$. Now since $G_m$ is a subplan, $\emptyset \subseteq V(G_m)$. Also $\emptyset - \{y\} \subseteq vs(G_m)$, since otherwise $x_i \in \emptyset - \{y\}$ for some $i < k$ since $x_i$ is closed before $x_k$ in the derivation $D$. Hence we can define $G_{m-1}$ by:

$$V(G_{m-1}) = V(G_m) - \emptyset$$

$$E(G_{m-1}) = E(G_m) - \{(x_k, y)\} - \{(y, z) \mid z \in \emptyset - \{y\}\}$$

Now $|SOL(G_{m-1})| = m-1$ and if $G_{m-1}$ is a plan, $G_m$ is clearly a plan.

Since $SOL(G_m)$ is finite, a finite number of applications of this process must eventually yield a subplan having no arcs of $SCL(G)$. The only such subplan is $G_0$, the basic plan of $G$. Hence $G_m$ is a plan.

\[\]
The reduction and factoring rules are intimately related. Example 3.3.6 following the presentation of the soundness and completeness results demonstrates the need for the restrictive conditions on these rules. Meanwhile, we note that in the absence of factoring, condition (c) on the applicability of reduction can be removed. Similarly, in the absence of reduction, condition (c) on back factoring can be weakened to "y is not an ancestor of x". Also, in the absence of ancestor replacement, factoring is equivalent to simple factoring, according to the following lemma and its corollary.

3.2.11: Lemma: If G is a plan constructed using rules (1)A and (3), then G can be constructed using (1)A and (3)A.

Proof: We will show that there exists a derivation \( G_0, \ldots, G_n \) of G such that for some \( m \leq n \), \( G_m \) is obtained from \( G_{m-1} \) by simple factoring, and if \( m < n \), then for \( i > m \), \( G_i \) is derived from \( G_{i-1} \) by simple replacement. Since \( G_{m-1} \) is a plan constructed using (1)A and (3) only, and \( |\text{FACT}(G_{m-1})| = |\text{FACT}(G_m)|-1 \), a finite number of applications of this process will yield a derivation of G in which all factorings are simple.

If \( \text{FACT}(G) \) is empty, there is nothing to prove, so we suppose the contrary. We have two cases to consider.

Case (a): Suppose for some \((x, y) \in \text{FACT}(G)\), \( y \) is open.

Obviously this factoring is simple. Define \( G_{n-1} \) by:

\[ V(G_{n-1}) = V(G) \]
\[ E(G_{n-1}) = E(G) - \{(x,y)\} \]

Then \( G_{n-1} \) is a subplan of \( G \) by lemma 3.2.3, and hence a plan by lemma 3.2.10. If \((G_0, \ldots, G_{n-1})\) is a derivation of \( G_{n-1} \), then \((G_0, \ldots, G_n)\), where \( G_n = G \), is a derivation of \( G \) with the required properties.

**Case (b):** Suppose that for every \((x,y) \in \text{FACT}(G)\), \( y \) is closed.

(A) Suppose that for every \((x,y) \in \text{FACT}(G)\),

if \( y \) is closed by replacement,

then either some direct descendant of \( y \)

is closed by factoring,

or some subproblem \( z \) is factored to

a direct descendant of \( y \).

We show that hypothesis (A) leads to a contradiction.

To do this, we show that, for any integer \( k \), there exist two sequences of subproblems, \( x_1, x_2, \ldots, x_k \), and \( y_1, y_2, \ldots, y_k \) such that for all \( i \), where \( 1 \leq i < k \):

(i) \( (x_i, y_i) \in \text{FACT}(G) \)

(ii) \( y_i \) is closed by replacement.

(iii) \( y_{i+1} \) is a direct descendant of \( y_i \)

(iv) no direct descendant of \( y_i \) is closed by factoring.

These sequences are constructed inductively as follows:

**Basis:** Since for every \((x,y) \in \text{FACT}(G)\), \( y \) is closed, lemma 3.2.8 ensures that \( \exists (x, y) \in \text{FACT}(G) \) such \( y \) is closed by replacement, and no direct descendant of \( y \) is closed by factoring. Let \( x_1 = x \) and \( y_1 = y \). Then
$x_k$ and $y_k$ obviously satisfy the conditions (i) to (iv).

**Induction:** Suppose a suitable sequence of length $k-1$ has been constructed. Now $y_{k-1}$ is closed by replacement, and no direct descendant of $y_{k-1}$ is closed by factoring, by the induction hypothesis.

Therefore by the hypothesis (A), some subproblem $z$ is factored to a direct descendant $y$ of $y_{k-1}$. Let $x_k = z$ and $y_k = y$; then:

(i) $(x_k, y_k) \in \text{FACT}(G)$

(ii) $y_k$ is closed by replacement since it is a direct descendant of $y_{k-1}$, and by the induction hypothesis, no direct descendant of $y_{k-1}$ is closed by factoring.

(iii) $y_k$ is a direct descendant of $y_{k-1}$.

(iv) no direct descendant of $y_k$ is closed by factoring, since this would imply that $y_{k-1}$ has a direct descendant closed by factoring.

Hence sequences of length $k$ exist with the required properties.

Now since such sequences exist to any length, and $s(G)$ is finite, there exists a pair of sequences of length $|s(G)| + 1$: in this case, $y_i = y_j$ for some $i < j$, and $y_j$ is a direct descendant of $y_i$. This implies the existence of a closed walk in $G$, all the arcs of which
are in \( \text{WEPI}(G) \cup \text{SUB}(G) \), contrary to lemma 3.2.2. Thus hypothesis (A) is disproved, and there exists \((x, y) \in \text{FACT}(G)\) such that \(y\) is closed by replacement, no direct descendant of \(y\) is closed by factoring, and there is no subproblem \(z\) factored to a direct descendant of \(y\).

We now define:

\[
V(G_m) = V(G) - \{z \mid z \text{ is a direct descendant of } y\}
\]

\[
E(G_m) = E(G) - \{(w, z) \mid z \text{ is a direct descendant of } y\}
\]

It is easy to see that this definition is equivalent to the definition of the subgraph \(G'\) in lemma 3.2.3, so by that lemma, \(G_m\) is a subplan of \(G\), and therefore is a plan by lemma 3.2.10. Also \((x, y) \in \text{FACT}(G)\) and \(y\) is open, so by case(a) there is a derivation \((G_0, \ldots, G_m)\) such that \(G_m\) is obtained from \(G_{m-1}\) by simple factoring. Now by lemma 3.2.6, this derivation can be extended to a derivation for \(G\), and obviously \(G_i\) is obtained from \(G_{i-1}\) by simple replacement, for \(i > m\). Finally, from case(a) we have \(|\text{FACT}(G_m)\| = |\text{FACT}(G_m)| - 1 = |\text{FACT}(G)| - 1\).

This completes the proof.

3.2.12: Corollary: If \(G\) is a plan constructed using rules (1)A, (2) and (3) only, then \(G\) can be constructed using rules (1)A, (2) and (3)A only.

Proof: We define a subgraph \(H\) of \(G\) by:

\[
V(H) = V(G)
\]

\[
E(H) = E(G) - \text{RED}(G)
\]
One application of lemma 3.2.3 and one application of lemma 3.2.10 for each arc of RED(G) removed, proves that H is a plan. But H is constructed using rules (1)A and (3) only, so by lemma 3.2.11, there exists a derivation $(G_0, \ldots, G_m)$ of H, constructed using rules (1)A and (3)A only. By lemma 3.2.6, this derivation can be extended to a derivation $(G_0, \ldots, G_n)$ for G, where for $i \geq m$, $G_i$ is obtained from $G_{i-1}$ by reduction. $(G_0, \ldots, G_n)$ is a derivation of G constructed using rules (1)A, (2) and (3)A only.

3.2.13: Definition: A plan H is said to be closed if os(H) = 0.

3.2.14: Definition: Denote by $\mathcal{P}$ the set of all unordered pairs of variants and negations of variants of the literals which occur in the clauses of $\mathcal{S}$. That is:

$$\mathcal{P} = \{ \{l_1', l_2'\} \mid \exists e_1, e_2 \in \mathcal{S} \text{ such that}$$
$$l_1 \in e_1, l_2 \in e_2,$n
$$l_1' \text{ is a variant of } l_1,$n
$$l_2' \text{ is a variant of } l_2 \}$$

where $l$ denotes either $-l$ or $l$. If $H$ is any subgraph of a plan $G$ for $\mathcal{S}$, the constraint function from $\mathcal{P}(H)$ to $2^n$ is
defined by:

\[
C(e) = \begin{cases} \\
\emptyset & \text{if } e \not\in \text{SUB}(H) \\
\{(x,y)\} & \text{if } e = (x,y) \in \text{FACT}(H) \\
\{(x,-y)\} & \text{if } e = (x,y) \in \text{RED}(H) \text{ or } e = (x,y) \in \text{REPL}(H) \text{ where} \\
& (x,y) \text{ is a simple replacement} \\
\{(x,-y)\} \cup C(K)y & \text{if } e = (x,y) \in \text{REPL}(H) \text{ where} \\
& (x,y) \text{ is an ancestor replacement} \\
& \text{using the clause os}(K)y, \text{ and } K \\
& \text{is a subplan of } G \\
\end{cases}
\]

The set \( \bigcup_{e \in \text{GE}(H)} C(e) \) will be called the constraint set of \( H \), and will be denoted \( C(H) \).

The properties of plans discussed so far are purely structural, since they do not depend on the set of clauses under consideration. We now introduce the important concept "correctness", which is related to the unifiability of the constraint set of a plan, and hence determines the semantics of plans.

3.2.15: Definition: A subplan \( H \) is said to be correct if \( C(H) \) is unifiable, in which case we denote the most general unifier of \( C(H) \) by \( \theta(H) \), and call the clause \( \text{os}(H)\theta(H) \) the clause deduced by \( H \). Note that every subplan of a correct plan is correct, and that the clause deduced by a closed plan is the empty clause \( \Box \).
3.2.16: Example: If $\emptyset$ contains the empty clause $\emptyset$, we may use $\emptyset$ as the top clause for constructing the basic plan $G = \langle \text{TOP}, \emptyset \rangle$; then $G$ is a closed plan, and $os(G) = \emptyset$.

3.2.17: Example: Consider the plan $G$ of example 3.1.7 (figure 3.8). Figure 3.10 lists the constraints constituting $C(G)$: each constraint in this list is labelled with the integer corresponding to the arc in SOL(G) from which it originates (see figure 3.8). The constraint set $C(G)$ is unifiable, and its most general unifier is:

$$\Theta(G) = \{ (x,a), (y,a), (z,a), (w,f(a)), (y_4, f(a)) \}$$

---

1. $\{ P(x), P(y) \}$  
2. $\{ -P(f(y)), -P(w) \}$  
3. $\{ P(f(w)), P(f(y_1)) \}$  
4. $\{ P(y_4), P(f(y)) \}$  
5. $\{ Q(w,b), Q(f(z), z) \}$  
6. $\{ P(f(f(z)), P(f(w)) \}$  
7. $\{ P(z), P(y) \}$

The constraint set $C(G)$ for the plan $G$ of figure 3.8.

Figure 3.10

The clause deduced by $G$ is therefore $\{ P(y) \} \Theta(G) = \{ P(a) \}$.
3.2.18: Lemma: If $G$ is a correct plan for $\mathcal{E}$ generated by rules (1)A and (2) only, then there exists a correct plan $G'$ for $\mathcal{E}$ generated by rules (1) and (3)A only, such that:

$$os(G) \Theta(G) = os(G') \Theta(G') \cdot \alpha$$

for some substitution $\alpha$.

Proof: If $REI(G) = \emptyset$ there is nothing to prove, so we assume the contrary. Let $G_m$ be defined by:

$$V(G_m) = V(G)$$
$$E(G_m) = E(G) - RED(G)$$

One application of lemma 3.2.3 and one application of lemma 3.2.10 for each arc of $RED(G)$ removed, proves that $G_m$ is a plan; so by lemma 3.2.6, there is a derivation $(G_0, \ldots, G_n)$ of $G$ such that $m < n$ and for $i > m$, $G_i$ is derived by reduction from $G_{i-1}$.

We now construct a sequence of graphs $(G_m', G_{m+1}', \ldots, G_n')$ such that for $j \geq m$:

(i) $G_j'$ is a correct plan,

(ii) $os(G_j') = os(G_j)$

(iii) $G_{j+1}'$ is derived from $G_j'$ by ancestor replacement and simple factoring only.

(iv) there is a substitution $\alpha_j$ such that:

$$z \Theta(G_j) = z \Theta(G_j) \cdot \alpha_j$$

for all $z \in s(G_j)$

This sequence is constructed as follows:

(A) $G_m' = G_m$
(B) Suppose the construction is complete up to $G_j^\gamma$. $G_j^{k+1}$ is generated from $G_j$ by reducing some $x \in \text{os}(G_j)$ to some direct ancestor $y$. Let $H$ be defined by:

\[ V(H) = V(G_j^\gamma) - \{ z \mid z \text{ is a direct descendant of } y \} \]

\[ E(H) = E(G_j^\gamma) - \{(w,z) \mid w \text{ or } z \text{ is a direct descendant of } y \} \]

By lemma 3.2.3, $H$ is a subplan of $G_j^\gamma$. Also $\text{os}(H) = \{y,x_k,\ldots,x_k\}$, where $\{x_k,\ldots,x_k\} \subseteq \text{os}(G_j^\gamma)$. We now generate a sequence of graphs $(G_j^0,\ldots,G_j^k)$ as follows:

(a) Let $C(H)^\gamma$ be a variant of $C(H)$. $G_j^0$ is the plan obtained from $G_j^\gamma$ by replacing $x \in \text{os}(G_j^\gamma)$ through $y^\gamma$ by $\{x_k^\gamma,\ldots,x_k^\gamma\}$.

(b) For $i \in \{1,\ldots,k\}$, $G_j^i$ is the plan defined by:

\[ \text{FACT}(G_j^i) = \text{FACT}(G_j^{i-1}) \cup \{x_k^\gamma, x_k^\gamma\} \]

Let $G_j^{k+1} = G_j$; it remains to show that $G_j^{k+1}$ satisfies the required conditions.

(i) $G_j^{k+1}$ is clearly a plan, so we need only show that $C(G_j^{k+1})$ is unifiable.

First we show that $\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\}$ is unifiable. Since $C(G_j^{k+1}) = C(G_j) \cup \{x,y\}$ is unifiable, by lemma 2.3.6, $\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\}$ is unifiable, so by the induction hypothesis condition (iv), $\{x\theta(G_j^\gamma) \cdot a_j, \neg y\theta(G_j^\gamma) \cdot a_j\}$ is unifiable and therefore $\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\}$ has a unifier $a_j \cdot \text{mgu}\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\}$. Therefore there is a substitution $\beta$ such that:

\[
(1) \quad \alpha_j \cdot \text{mgu}\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\} = \text{mgu}\{x\theta(G_j^\gamma), \neg y\theta(G_j^\gamma)\} \cdot \beta
\]
We now proceed to show that $C(G_{j+1}^\gamma)$ is unifiable.

(2) $\cdots C(G_{j+1}^\gamma) = C(G_j^\gamma) \cup C(H) \gamma \cup \{(x, -y \gamma)\}$

$\cup \{[x_i \gamma, x_i] \mid i \in \{1, \ldots, k\}\}$

Let $\delta = \gamma^{-1} \Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\}$

We show that $\delta$ unifies each set in the above expression (2) for $C(G_{j+1}^\gamma)$.

$C(G_j^\gamma) \delta = (C(G_j^\gamma) \gamma^{-1}) (\Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

$= (C(G_j^\gamma) \Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

since none of the replaced variables of $\gamma^{-1}$ occur in $C(G_j^\gamma)$

So since $\Theta(G_j^\gamma)$ unifies $C(G_j^\gamma)$, $\delta$ unifies $C(G_j^\gamma)$.

$(C(H) \gamma) \delta = (C(H) (\gamma \gamma^{-1})) (\Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

$= (C(H) \gamma^{-1}) (\Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

by 2.3.7

$= (C(H) \Theta(G_j^\gamma)) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\}$

Since none of the replaced variables of $\gamma^{-1}$ occur in $C(H)$

So since $C(H) \subseteq C(G_j^\gamma)$, $\Theta(G_j^\gamma)$ unifies $C(H)$, so that $\delta$ unifies $C(H)$.

$\{(x, -y \gamma)\} \delta = \{(x \gamma^{-1}, -y \gamma^{-1})\} (\Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

by 2.3.7

$\{(x, -y)\} (\Theta(G_j^\gamma) \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\})$

since none of the replaced variables of $\gamma^{-1}$ occur in $x$ or $y$

$= \{(x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma))\} \text{mgu} \{x \Theta(G_j^\gamma), -y \Theta(G_j^\gamma)\}$

Therefore $\delta$ unifies $\{(x, -y \gamma)\}$.
Finally, for each \( i \in \{1, \ldots, k\} \):

\[
\{x_i \gamma, x_i \} = \{x_i \gamma^{-1}, x_i \gamma^{-1}\}(\Theta(G_i^\gamma) \text{e} \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\})
\]

by 2.3.7

Therefore \( \delta \) unifies \( \{x_i \gamma, x_i \} \).

Hence \( C(G_i^\gamma +_4) \) is unifiable so that \( G_i^\gamma +_4 \) is correct.

(ii) \( \text{os}(G_i^\gamma +_4) = \text{os}(G_i^\gamma) - \{x\} \)

\[= \text{os}(G_i) - \{x\} \]

\[= \text{os}(G_i +_4) \]

(iii) Since \( \{x_4, \ldots, x_k\} \subseteq \text{os}(G_i^\gamma) \), all the factorings performed in deriving \( G_i^\gamma +_4 \) from \( G_i^\gamma \) are simple.

(iv) In the proof of (i) above, we have shown that

\[
\gamma^{-1} \Theta(G_i^\gamma) \text{e} \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\}
\]

unifies \( C(G_i^\gamma +_4) \), so for some substitution \( \tau \):

\[
(2) \cdots \Theta(G_i^\gamma +_4) \cdot \tau = \gamma^{-1} \Theta(G_i^\gamma) \text{e} \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\}
\]

Let \( \beta \) be the substitution defined in equation (1) in part (i) above, and let \( \alpha_i +_4 = \tau \cdot \beta \), then if \( z \in \text{os}(G_i^\gamma +_4) \):

\[
z \Theta(G_i^\gamma +_4) \cdot \alpha_i +_4 = z \Theta(G_i^\gamma +_4) \cdot \tau \cdot \beta
\]

\[= z \gamma^{-1} \Theta(G_i^\gamma) \text{e} \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\} \cdot \beta
\]

from (2) above

\[= z \Theta(G_i^\gamma) \text{e} \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\} \cdot \beta
\]

since none of the replaced variables of \( \gamma^{-1} \) occur in \( z \)

\[= z \Theta(G_i^\gamma) \cdot \alpha_i \cdot \text{mgu}\{x\Theta(G_i^\gamma), \neg y \Theta(G_i^\gamma)\}\]

3.2.18
by (1) in (i) above

\[ z \in (G_j) \cup \{ y \in (G_j) \} \cup \{ x \in (G_j) \} \]

by induction hypothesis, condition (iv), and since
\[ z \in s(G_j + 1) = s(G_j) \]

But \( C(G_j + 1) = C(G_j) \cup \{ x, \neg y \} \)

So by lemma 2.3.6:

\[ \epsilon(G_j + 1) = \Theta(G_j) \cup \{ x \Theta(G_j), \neg y \Theta(G_j) \} \]

\[ \therefore z \in (G_j + 1) = z \Theta(G_j + 1) \cup \{ \alpha_j + 1 \} \]

The sequence having been constructed, let \( G' = \bar{G}_n \) and \( \alpha = \alpha_n \), then:

\[ o_s(G) = o_s(G') \]

and \( z \in (G) = z \Theta(G') \cup \{ \alpha \} \) for all \( z \in s(G) \)

\[ \therefore o_s(G) \Theta(G) = o_s(G') \Theta(G') \cup \{ \alpha \} \]

3.2.19: Lemma: If \( G \) is a closed, correct plan for \( \$ \)
generated by rules (1)A and (2) only, then there exists a
closed, correct plan \( G' \) for \( \$ \) generated by rules (1) and
(3)B only.

Proof: If \( RED(G) = \emptyset \) there is nothing to prove, so we
assume the contrary. Let \( G_m \) be defined by:

\[ V(G_m) = V(G) \]

\[ E(G_m) = E(G) - RED(G) \]

One application of lemma 3.2.3, and one application of lemma
3.2.10 for each arc of RED(G) removed, proves that \( G_m \) is a
plan; so by lemma 3.2.6, there is a derivation \((G_0, \cdots, G_n)\) of \(G\) and for \(i > n\), \(G_i\) is derived by reduction from \(G_i - i\). Let \(H\) be defined by:

\[
V(H) = V(G_m) - \{z \mid \exists (x, y) \in \text{RED}(G) \text{ such that } z \text{ is a direct descendant of } y \}
\]

\[
E(H) = E(G_m) - \{(w, z) \mid \exists (x, y) \in \text{RED}(G) \text{ such that either } w \text{ or } z \text{ is a direct descendant of } y \}
\]

Again, lemmas 3.2.3 and 3.2.10 ensure that \(H\) is a subplan of \(G\).

Now suppose \(x \in \text{os}(G_m)\), then \((x, y) \in \text{RED}(G)\) for some direct ancestor \(y\) of \(x\); but \(x\) is then a direct descendant of \(y\), so that \(x \notin V(H)\). Hence \(\text{os}(H) \subseteq \text{cs}(G_m)\).

We now construct a sequence of graphs \((G'_m, G'_{m+1}, \cdots, G'_n)\)

such that for \(j \geq m\):

(i) \(G'_j\) is a correct plan,

(ii) \(\text{os}(G'_j) = \text{os}(G_j)\)

(iii) \(G'_{j+1}\) is derived from \(G'_j\) by ancestor replacement and back factoring only.

(iv) there is a substitution \(a_j\) such that:

\[
z \text{os}(G_j) = z \text{os}(G'_j) \ast a_j \text{ for all } z \in s(G_j)
\]

(v) \(G_m\) is a subplan of \(G'_j\)

This sequence is constructed as follows:

(A) \(G'_m = G_m\)

(B) Suppose the construction is complete up to \(G'_j\). \(G'_{j+1}\) is generated from \(G_j\) by reducing some \(x \in \text{os}(G_j)\) to some direct ancestor \(y\). Now \(y \notin \text{os}(H)\), since \(y\) is at the
head of an arc of RED(G), and all such vertices lost their direct descendants in the construction of H.
Suppose \( cs(H) = \{y, x_1, \ldots, x_k\} \). We now generate a sequence of graphs \( (G_0, \ldots, G_k) \) as follows:
(a) Let \( C(H) \gamma \) be a variant of \( C(H) \). \( G_0 \) is the plan obtained from \( G_1^\gamma \) by replacing \( x \in os(G_1^\gamma) \) through \( y \gamma \) by \( \{x_1 \gamma, \ldots, x_k \gamma\} \).
(b) For \( i \in \{1, \ldots, k\} \), \( G_i^\gamma \) is the plan defined by:
\[
FACT(G_i^\gamma) = FACT(G_i^{\gamma^{-1}}) \cup \{(x_i \gamma, x_i)\}
\]
Let \( G_{i+1}^\gamma = G_i^\gamma \); it remains to show that \( G_{i+1}^\gamma \) satisfies the required conditions.
(i), (ii) and (iv) are proved analogously to the corresponding parts of lemma 3.2.18.
(iii) Now since \( os(H) \subseteq cs(G_m) \), then by condition (iv), \( os(H) \subseteq cs(G_i^\gamma) \). Hence all the factorings performed in deriving \( G_{i+1}^\gamma \) from \( G_i^\gamma \) are back factorings.
(v) \( G_m \) is a subplan of \( G_i^\gamma \) and hence of \( G_{i+1}^\gamma \).

The sequence having been constructed, let \( G' = G_n^\gamma \), then \( cs(G') = os(G) = \emptyset \), and \( G' \) is correct.
3.3: **Soundness and Completeness**

Our aim in this section is to show that a set of clauses $\mathcal{G}$ is unsatisfiable if and only if there is a closed, correct plan for $\mathcal{G}$.

**2.3.1: Definition:** If $G$ is a subplan, and $\gamma$ is any substitution, we define:

$$E(G)\gamma = \{(x\gamma, y\gamma) \mid (x, y) \in E(G)\}$$

Also, we denote the graph $<V(G)\gamma, E(G)\gamma>$ by $G\gamma$, and call $G\gamma$ an instance of $G$. $G\gamma$ is a variable-free instance of $G$ if for every $x \in V(G)$, $x\gamma$ is variable-free. $G\gamma$ is a variant of $G$ if $x\gamma$ is a variant of $x$ for every $x \in V(G)$. If $G$ is a plan for a set $\mathcal{G}$ of clauses, then clearly so is every variant of $G$.

**3.3.2: Lemma:** Let $G$ be a correct plan for a set $\mathcal{G}$ of clauses, $H$ a subplan of $G$, $\Xi$ a model for $\mathcal{G}$, and $H\theta(H)\gamma$ a variable-free instance of $H\theta(H)$: then there exists a walk $(x_0, x_1, \ldots, x_{n-1}, x_n)$ in $H$ containing no arcs in $RED(H)$, such that $x_0 = \text{TOP}$, $x_n \in os(H)$, and $\Xi(x_i \theta(H)\gamma) = 1$ for every $x_i \in s(H)$, $0 < i < n$.

**Proof:** We prove this by induction on the number of ancestor replacements performed in the construction of $G$.

**Basis:** Suppose $G$ is constructed without ancestor replacement. We construct the walk recursively as follows:
(i) The top clause \( \{ x \mid (\text{TOP}, x) \in \text{SUB}(H) \} \) of \( H \) is a variant of some clause in \( \mathcal{G} \), and therefore contains some literal \( x_a \) such that \( \mathcal{I}(x_a \Theta(H) \land y) = T \).

\((\text{TOP}, x_a)\) is the first arc in the walk.

(ii) Suppose the walk has been constructed up to the arc \((x_{i-1}, x_i)\), and that \( x_i \) is a subproblem.

Suppose \((x_i, y) \in \text{RED}(H)\).

either the walk from TOP to \( x_i \) contains no arcs of \( \text{FACT}(H) \), in which case all direct ancestors of \( x_i \) must lie on the walk, so \( y \) in particular must lie on the walk.

or the walk from TOP to \( x_i \) contains arcs in \( \text{FACT}(H) \). In this case, let \((x_j, x_{j+1})\) be the first arc on the walk in \( \text{FACT}(H) \), where \( j < i \), then:

either that part of the walk from \( x_{j+1} \) to \( x_i \) passes through \( y \),
or by lemma 3.2.4, \( y \) is a direct ancestor of \( x_j \), so that part of the walk from TOP to \( x_j \) passes through \( y \).

In any event, \( y \) must lie on the walk.

\[ \mathcal{I}(y \Theta(H) \land y) = T \]

But \( \mathcal{I}(y \Theta(H) \land y) = \mathcal{I}(\neg x_i \Theta(H) \land y) \)

\[ = F \]

which is a contradiction. Therefore \( x_i \) is not closed by reduction in \( H \). Hence we have three cases to consider:
(a) \( x_i \in \text{os}(B) \), in which case \( n = 1 \), and we have the required walk.

(b) \( x_i \) is closed by factoring in \( H \), say \((x_i, z) \in \text{FACT}(B)\), in which case:

\[
z \in (B) = x_i \in (B)
\]

\[\therefore \Sigma(z \in (B) \cdot \gamma) = \Sigma(x_i \in (B) \cdot \gamma) = T\]

Let \( x_i + z = z; (x_i, x_i + z) \) is then the next arc in the walk.

(c) \( x_i \) is closed by simple replacement through \( y \) in \( B \); then we define \( x_i + z = y \), so that \((x_i, x_i + z) \in \text{REPL}(B)\). In this case \( \{x_i + z\} \cup \{x \mid (x_i + z, x) \in \text{SUB}(H)\} \) is a variant of a clause in \( \mathcal{E} \), and therefore:

\[
\Sigma([\{x_i + z\} \cup \{x \mid (x_i + z, x) \in \text{SUB}(H)\}] \Theta(B) \cdot \gamma) = T
\]

But \( x_i + z \in (B) = \neg x_i \in (B) \)

\[\therefore \Sigma(x_i + z \in (B) \cdot \gamma) = \Sigma(\neg x_i \in (B) \cdot \gamma) = F\]

Hence \( \exists x \in \{x \mid (x_i + z, x) \in \text{SUB}(H)\} \) such that:

\[\Sigma(x \in (B) \cdot \gamma) = T\.

We define \( x_i + 2 = z \). Then \((x_i, x_i + z)(x_i + z, x_i + 2)\) are the next two arcs in the walk, and satisfy the required conditions.

Suppose this construction does not terminate as in case (a), then since \( V(H) \) is finite, the process must generate a walk which passes through some
vertex of \( H \) twice. In the latter case, there exists a closed walk in \( H \) containing no arcs in \( \text{RED}(H) \). By lemma 3.3.2, however, no closed walk can consist entirely of arcs in \( \text{REPL}(H) \cup \text{SUB}(H) \). Therefore some arc \((x_j, x_{j+1})\) on this closed walk must belong to \( \text{FACT}(H) \): then there is a walk from \( x_{j+1} \) to \( x_j \) containing no arcs in \( \text{RED}(H) \), contrary to lemma 3.2.4. Hence the construction must terminate as in case (a).

**Induction:** Assume the result for plans constructed with fewer ancestor replacements than \( G \). We then construct the required walk in \( H \) exactly as in the basis of the proof, except that we have one more case to consider when extending the walk, as follows:

(d) \( x_i \) is closed by ancestor replacement, using a variant \( Ka \) of some subplan \( K \) of \( G \). Suppose \( x_i \) is replaced through \( ya \). Let \( x_{i+1} = ya \), then:

\[
\text{cs}(K) = \{y\} \cup \{w \mid (x_{i+1}, wa) \in \text{SUB}(H)\}
\]

Let \((G_0, \ldots, G_m)\) be a derivation of \( G \), then for some \( k \leq m \):

\[
\text{cs}(G_k) = \text{cs}(G_{k-1}) \cup \{x_i\}
\]

\( G_{k-1} \) is a correct plan for \( \bar{F} \), constructed using fewer ancestor replacements than are used in the construction of \( G \), and \( K \) is a subplan of \( G_{k-1} \). So by the induction hypothesis:
(1) If $K\theta(K) \circ \tau$ is a variable-free instance of $K\theta(K)$, then $\exists z \in \text{os}(K)$ such that:

$$\mathcal{I}(z\theta(K) \circ \tau) = T$$

Also we have:

$$C(K) \alpha \subseteq C(H)$$

- $\theta(H)$ unifies $C(K) \alpha$
- $\alpha \circ \theta(H)$ unifies $C(K)$

Therefore:

(2) $\alpha \circ \theta(H) = \theta(K) \circ \beta$

for some substitution $\beta$

$K\theta(H) \circ \gamma$ is a variable-free instance of $K\theta(H)$.

Suppose $(K\alpha)\theta(H) \circ \gamma$ is not variable-free; this is possible since $K$ is not necessarily a subplan of $H$. In this case, let $\gamma_1$ be some substitution such that $(K\alpha)\theta(H) \circ \gamma \circ \gamma_1$ is variable-free, then:

(3) $x \theta(H) \circ \gamma \circ \gamma_1 = x \theta \gamma$ if $x \in \text{V}(H)$

Now since $(K\alpha)\theta(H) \circ \gamma \circ \gamma_1$ is variable-free, by applying (2), we see that $K\theta(K) \circ (\beta \circ \gamma \circ \gamma_1)$ is a variable-free instance of $K\theta(K)$, so by (1):

$$\mathcal{I}(z\theta(K) \circ (\beta \circ \gamma \circ \gamma_1)) = T$$

for some $z \in \text{os}(K)$

Now $\mathcal{I}(y\theta(K) \circ (\beta \circ \gamma \circ \gamma_1)) = \mathcal{I}(x \circ \theta(K) \circ (\gamma \circ \gamma_1))$

by applying (2)

$$= \mathcal{I}(-x \theta(H) \circ (\gamma \circ \gamma_1))$$

$$= \mathcal{I}(-x \theta(H) \circ \gamma)$$

by (3)
= F
  ⋅⋅⋅ z ≠ y
  ⋅⋅⋅ z ∈ \{w | (x_{i+1}, wα) ∈ \text{SUB}(H)\}

Let \( x_{i+2} = zα \)

Then \((x_{i+1}, x_{i+2}) ∈ \text{SUB}(H)\)

and \(\Xi(x_{i+2}θ(H)•γ) = \Xi((zαθ(H)•γ)γ_1)\)

= \(\Xi(ze(K)•(β•γ)γ_4)\) = 1

\((x_i, x_{i+1}), (x_{i+1}, x_{i+2})\) are then the next two arcs on the walk and satisfy the required conditions.

3.3.3: Theorem: The Soundness of Plans

If there exists a closed, correct plan for a set \( S \) of clauses, then \( S \) is unsatisfiable.

Proof: Let \( G \) be a closed, correct plan for \( S \), and suppose \( S \) has a model. Then by lemma 3.3.2, there is a walk from \( \text{TOP} \) to \( y \) in \( G \) such that \( y \in \text{os}(G) \), contradicting the fact that \( G \) is closed. Therefore \( S \) has no model.

It now remains to show that plans are complete. To this end, we now present a description of Loveland's model elimination deduction system [24,25,26], which is equivalent to the SL-resolution system of Kowalski and Kuehner [23].
3.3.4: Model Elimination

3.3.4.1: Definition: A literal occurrence is an ordered pair \((l, i)\) where \(l\) is a literal and \(i\) is an integer. We will refer to literal occurrences simply as occurrences. \((l, i)\) is said to be an occurrence of \(l\).

3.3.4.2: Definition: A chain of length \(n\) is a set \(K\) of occurrences, such that:

\[ K = \{(l_1, 1), \ldots, (l_n, n)\} \]

Each chain \(K\) is divided into two disjoint subsets \(A(K)\) and \(B(K)\), the elements of which are called \(A\)-occurrences and \(B\)-occurrences respectively.

3.3.4.3: Definition: If \(A(K) = \emptyset\), \(K\) is said to be elementary.

Those familiar with Loveland's description of his system will have noticed that our definitions differ from his in that we have made the difference between "literals" and "literal occurrences" explicit. We will later define functions which map elements of a chain into the vertices of a plan, so confusion is likely unless we can distinguish between different occurrences of the same literal in a chain. The above representation is cumbersome, however, so we will streamline it as follows.
3.3.4.4

(i) The chain \([(l_1,1), \ldots, (l_n,n)]\) will henceforth be represented as \((l_1, \ldots, l_n)\).

(ii) If \(l = (m,i)\) is an occurrence, we use \(\text{lit}(l)\) and \(\text{pos}(l)\) to denote \(m\) and \(i\) respectively.

(iii) If \(L\) is a set of occurrences, we will denote the set \[\{\text{lit}(l) \mid l \in L\}\] by \(\text{lit}(L)\).

(iv) For any substitution \(\gamma\), we define \((m,i)\gamma\) to be the occurrence \((m\gamma, i)\).

(v) If \(L = (l_1, \ldots, l_n)\) is a chain, we define \(L\gamma = (l_1\gamma, \ldots, l_n\gamma)\).

This representation for chains allows us to refer to the relative positions of occurrences in chains as follows:

3.3.4.5: Definition: If \(K\) is a chain, and \(l, k \in K\), then we say that \(l\) is to the left of \(k\) in \(K\) if \(\text{pos}(l) < \text{pos}(k)\), and denote this by \(l < (K) k\). Similarly, we can use such phrases as "\(k\) is the last occurrence in \(K\); "\(k\) and \(l\) are separated by \(p\); and so on.

3.3.4.6: Definition: A chain \(K\) is said to be preadmissible if and only if:

(i) any two \(E\)-occurrences of complementary literals are separated by some \(A\)-occurrence.
(ii) no B-occurrence appears to the right of an A-occurrence of the same literal,
(iii) there are no two A-occurrences of identical or complementary literals.

3.3.4.7: Definition: A chain is admissible if it is preadmissible and its last occurrence is a B-occurrence. The empty chain is defined to be admissible.

3.3.4.8: Definition: If \( \mathcal{W} \) is a set of elementary chains, a finite sequence \( (k_0, \ldots, k_n) \) of chains is called an \( \mathcal{WF} \)-deduction of \( k \) from \( \mathcal{W} \) if \( k_0 \) is a variant of some chain in \( \mathcal{W} \), and for each \( i \in \{1, \ldots, n\} \), \( k_i \) is derived from \( k_{i-1} = (k_1, \ldots, k_m) \) by one of the following rules.

(i) Extension

If \( k_{i-1} \) is admissible, and \( k = (l_1, \ldots, l_r) \) is a variant of some chain in \( \mathcal{W} \), then:

\[
K_i = (k_1, \ldots, k_m, l_2, \ldots, l_r) \gamma
\]

and \( A(K_i) = (A(k_{i-1}) \cup \{(k_m, m)\}) \gamma \)

where \( \gamma = mgu(k_m, \neg l_1) \)

(ii) Reduction

If \( k_{i-1} \) is admissible:

\[
k_i = (k_1, \ldots, k_m, l) \gamma
\]

and \( A(k_i) = A(k_{i-1}) \gamma \)

where \( \gamma = mgu(k_m, \neg k_1) \)

and \( (k_j, j) \in A(k_{i-1}) \)

for some \( k_j \) to the left of \( k_m \).
(iii) Contraction

If $K_1 \rightarrow k$ is preadmissible, and the last occurrence of $K_1 \rightarrow k$ is an $A$-occurrence, then:

$$X_1 = (k_1, \ldots, k_{m-1})$$

and $A(X_1) = A(K_1) - \{(k_m, m)\}$

3.3.4.9: Definition: If $C$ is a clause and $k \in C$, then a **matrix chain for $C$** is a chain obtained by imposing some ordering on $C$. Note that there is only one occurrence of any literal in a matrix chain.

If $C$ is a clause, a **matrix set for $C$** is a set $M$ of matrix chains for $C$ such that:

(a) If $k \in C$

then $M$ contains a chain in which the first occurrence is an occurrence of $k$.

(b) If $L_1, L_2 \in M$ and the first occurrences of $L_1$ and $L_2$ are occurrences of the same literal of $C$

then $L_1 = L_2$

If $\mathcal{S}$ is a set of clauses, let $\mathcal{F}$ be a family of matrix sets consisting of one and only one matrix set for each $C \in \mathcal{S}$, then $\bigcup \mathcal{F}$ is called a **matrix set for $\mathcal{S}$**.

3.3.4.10: Definition: A clause $C$ is said to be **ME-deducible from $\mathcal{S}$**, a set of clauses, if there exists an ME-deduction $(X_0, \ldots, X_n)$ from some matrix set $M$ for $\mathcal{S}$, such that $C = \text{lit}(B(X_n))$. 

3.3.4.10
3.3.4.11: Theorem: (Loveland [25])

$\mathcal{F}$ is unsatisfiable if and only if the empty clause $\varnothing$ is NE-deducible from $\mathcal{F}$.

3.3.4.12: Lemma: If a clause $\mathcal{C}$ is NE-deducible from a set $\mathcal{F}$ of clauses, then there exists a correct plan $G$ for $\mathcal{F}$, such that $G$ is constructed using rules (1A) and (2) only, and $os(G)\Theta(G) = \mathcal{C}$.

Proof: Let $(X_0, \ldots, X_n)$ be an NE-deduction from some matrix set $M$ for $\mathcal{F}$ such that $\mathcal{C} = lit(B(X_n))$. We show now that for each $i \in \{0, \ldots, n\}$, there is a correct plan $G_i$ and a function $g_i : X_i \rightarrow s(G_i)$ such that:

(i) $G_i$ is constructed using rules (1A) and (2) only, and is correct,

(ii) $g_i$ is injective,

(iii) $g_i(B(X_i)) = os(G_i)$,

where for any set of occurrences $L$, we define $g_i(L) = \{g_i(l) | l \in L\}$

(iv) $lit(l) = g_i(l) \Theta(G_i)$ for all $l \in X_i$,

(v) If $l \in A(X_i)$ and $l < (X_i)_p$, then $g_i(l)$ is a direct ancestor of $g_i(p)$.

The proof is by induction on $i$.

Basis: $X_0$ is a variant of a chain in $M$, and so $\{lit(l) | l \in X_0\}$ is a variant of a clause in $\mathcal{F}$. Let $G_0$ be the basic plan with this top clause, and define:
\( g_0(l) = \text{lit}(l) \) for all \( l \in X_0 \)

Then \( G_0 \) and \( g_0 \) have the required properties.

**Induction:** Assume \( G_{i-1} \) and \( g_{i-1} \) have been constructed. We have three cases to consider.

**Case(a):** \( X_i \) is derived by contraction from \( X_{i-1} \). We define:

\[
G_i = G_{i-1}
\]

and \( g_i = g_{i-1}|_{X_i} \)

Again \( G_i \) and \( g_i \) have the required properties.

**Case(b):** \( X_i \) is derived by reduction from \( X_{i-1} \).

Let \( X_{i-1} = (k_1, \ldots, k_m) \)

then \( X_i = (k_1, \ldots, k_{m-1})\gamma \)

and \( B(X_i) = (B(X_{i-1}) - \{(k_m, m)\})\gamma \)

where \( \gamma = \text{mg}(k_m, -k_i) \)

and \((k_i, j) \in A(X_{i-1})\)

Now \((k_i, j) \in A(X_{i-1})\) and \((k_i, j) < (X_{i-1})(k_m, m)\), so by condition (v) on \( G_{i-1} \), \( g_{i-1}((k_i, j)) \) is a direct ancestor of \( g_{i-1}((k_m, m)) \). Also, \((k_m, m) \in B(X_{i-1})\), so that \( g_{i-1}((k_m, m)) \in \text{os}(G_{i-1}) \) by condition (ii). The conditions for reducing \( g_{i-1}((k_m, m)) \) to \( g_{i-1}((k_i, j)) \) are thereby satisfied in \( G_{i-1} \), and we define \( G_i \) accordingly.

We define \( g_i \) by:

\[
g_i(k\gamma) = g_{i-1}(k) \quad \text{for all} \ k\gamma \in X_i
\]
(i) \( G_i \) is constructed using (1)A and (2) only, since it is derived by reduction from \( G_{i-1} \), which satisfies this condition by the induction hypothesis. We must now show that \( G_i \) is correct.

Now \( C(G_i) = C(G_{i-1}) \cup \{\{\varphi_{i-1}((k_m,m)), \neg \varphi_{i-1}((k_j,j))\}\} \)

But \( \text{mguc}(G_{i-1}) = \Theta(G_{i-1}) \), so by lemma 2.3.6 \( C(G_i) \) is unifiable if

\[ \{\{\varphi_{i-1}((k_m,m))\Theta(G_{i-1}) \neg \varphi_{i-1}((k_j,j))\Theta(G_{i-1})\}\} \]

is unifiable. But by the induction hypothesis, from condition (iv) we have:

\[ \varphi_{i-1}((k_m,m))\Theta(G_{i-1}) = k_m \]

and \( \varphi_{i-1}((k_j,j))\Theta(G_{i-1}) = k_j \)

So \( C(G_i) \) is unifiable if \( \{k_m, k_j\} \) is unifiable; therefore, since \( \text{mguc}\{k_m, k_j\} = \gamma \), \( C(G_i) \) is unifiable.

Therefore \( G_i \) is correct and:

\[ \Theta(G_i) = \Theta(G_{i-1})^\gamma \] by lemma 2.3.6

(ii) \( \varphi_i \) is clearly injective, since \( \varphi_{i-1} \) is injective, by the induction hypothesis.

(iii) \( \varphi_i(B(X_i)) = \varphi_i((B(X_{i-1}) - (\{k_m,m\})) \gamma) \)

\[ = \varphi_i(B(X_{i-1}) \gamma - (\{k_m,m\}) \gamma) \]

\[ = \varphi_i(B(X_{i-1}) \gamma) - \{\varphi_i((k_m,m)) \gamma) \}

since \( \varphi_i \) is injective

\[ = \varphi_{i-1}(B(X_{i-1})) - \{\varphi_{i-1}((k_m,m)) \}

\[ = \Theta(G_{i-1}) - \{\varphi_{i-1}((k_m,m)) \}

\[ = \Theta(G_i) \]

(iv) Suppose \( k \gamma \in X_i \), then \( k \in X_{i-1} \) and:

\[ \text{lit}(k \gamma) = \text{lit}(k) \gamma \] 3.3.4.12
= (g_{i-1}(k)\Theta(G_{i-4}))\gamma

by condition (iv)

= g_{i-1}(k)(\Theta(G_{i-1})\Theta(\gamma))

= g_{i-1}(k)\Theta(G_{i})

as shown in (i) above.

(v) If q\gamma \in A(K_{i}) and q\gamma \preceq (K_{i}) p\gamma, then q \in A(K_{i-1}) and q \preceq (K_{i-4}) p, so that g_{i-1}(q) is a direct ancestor of g_{i-1}(p) by condition (v). Consequently, g_{i}(q\gamma) is a direct ancestor of g_{i}(p\gamma).

Case (c): K_{i} is derived by extension from K_{i-4}.

Let K_{i-4} = (k_{1}, \cdots, k_{m})

then K_{i} = (k_{1}, \cdots, k_{m}, l_{2}, \cdots, l_{s})\gamma

and A(K_{i}) = (A(K_{i-4}) \cup \{(k_{m}, m)\})\gamma

where \gamma = mgu\{k_{m}, l_{s}\}

and K = (l_{2}, \cdots, l_{s}) is a variant of some chain in \mathcal{M}.

(k_{m}, m) \in B(K_{i-4}), so by condition (iii) on G_{i-4},

\varphi_{i-4}((k_{m}, m)) \in os(G_{i-4}). Now \{l_{2}, \cdots, l_{s}\} is a variant of a clause in \varphi, so we can replace \varphi_{i-4}((k_{m}, m)) through \varphi_{i-4} to obtain G_{i}.

We define \varphi_{i} as follows:

\varphi_{i}(k\gamma) = \begin{cases} \varphi_{i-4}(k) & \text{for all } k \in K_{i-4} \\
\text{lit}(k) \in V(G_{i}) - V(G_{i-4}) & \text{otherwise} \end{cases}
Note that since there may be more than one vertex in a plan corresponding to a particular literal, we must specify in our definition of $g_i$ that for $k \in \{(l_2, m+1), \ldots, (l_r, m+r-1)\}$, $g_i(k)$ is a new vertex.

(i) $G_i$ is constructed by simple replacement from $G_{i-1}$, which is constructed using rules (1)A and (2) only, by the induction hypothesis. Therefore $G_i$ is constructed using (1)A and (2). We must now show that $G_i$ is correct.

$$C(G_i) = C(G_{i-1}) \cup \{[g_{i-1}((k_m, m)), \lnot l_4]\}$$

Now $C(G_{i-1})$ is unifiable, with mgu $\Theta(G_{i-1})$; so by lemma 2.3.6, $C(G_i)$ is unifiable provided that $[g_{i-1}((k_m, m))\Theta(G_{i-1}), \lnot l_4 \Theta(G_{i-1})]$ is unifiable. But by the induction hypothesis, from condition (iv) we have:

$$g_{i-1}((k_m, m))\Theta(G_{i-1}) = k_m$$

Also, since $l_4$ contains only variables which do not occur in $G_{i-1}$:

$$\lnot l_4 \Theta(G_{i-1}) = \lnot l_4$$

Hence $C(G_i)$ is unifiable if $[k_m, \lnot l_4]$ is unifiable; so since mgu$k_m, \lnot l_4] = \gamma$, $C(G_i)$ is unifiable. Therefore $G_i$ is correct and:

$$\Theta(G_i) = \Theta(G_{i-1}) \gamma$$

(ii) We establish that $g_i$ is injective by making the following observations:
(1) $\varphi |_{k_1 - q}$ is injective since, by the induction hypothesis, $\varphi |_{k_1}$ is injective.

(2) $\varphi |_{\{l_2, m+1\}, \ldots, (l_r, m+r-1)}$ is obviously injective.

(3) If $k_4 \in K_{l_4} - 1$ and $k_2 \in \{l_2, m+1\}, \ldots, (l_r, m+r-1)$
then $\varphi |_{k_4} \in V(G_{l_4})$
and $\varphi |_{k_2} \in V(G_i) - V(G_{l_4})$
so $\varphi |_{k_4} \neq \varphi |_{k_2}$

(iii) $B(K_i) = B(K_{l_4}) - \{(k_m, m)\}$

$\cup \{(l_2, m+1), \ldots, (l_r, m+r-1)\}$

$\cdot \cdot \cdot \varphi |_{B(K_i)} = \varphi |_{B(K_{l_4})} - \varphi |_{\{(k_m, m)\}}$

$\cup \{\text{lit}(k) \mid k \in \{(l_2, m), \ldots, (l_r, m+r-1)\}\}$

since $\varphi |_{k_4}$ is injective

$= \varphi |_{k_4} - \varphi |_{\{(k_m, m)\}}$

$= \text{os}(G_{l_4}) - \varphi |_{\{(k_m, m)\}}$

$= \varphi |_{k_4} - \varphi |_{\{(k_m, m)\}}$

by the induction hypothesis

(iv) Suppose $l \gamma \in K_{l_4}$, then either $l \in K_{l_4} - 1$ or $l \in K_i$.

In the first case:

$\text{lit}(l \gamma) = \text{lit}(l) \gamma$

$= \varphi |_{l \gamma} - \varphi |_{\{(k_m, m)\}}$

$= \varphi |_{l \gamma} - \varphi |_{\{(k_m, m)\}}$ as shown above

$= \varphi |_{l \gamma} \Theta(G_i)$

In the second case:

$\text{lit}(l \gamma) = \text{lit}(l) \gamma$

$= (\text{lit}(l) \Theta(G_{l_4})) \gamma$
since lit(l) contains only variables not in \( G_{i-\xi} \)

\[ = \text{lit}(l) \Theta(G_i) \]
\[ = g_i(l \gamma) \Theta(G_i) \]

(v) Suppose \( q \gamma \in A(k_i) \) and \( q \gamma \prec (k_i) \ p \gamma \). There are two cases to consider.

(a) Suppose \( p \in X_{i-\xi} \), then \( q \neq (k_m, m) \), so that \( q \in A(X_{i-\xi}) \). Also, \( q \prec (X_{i-\xi}) \ p \) so \( g_{i-\xi}(q) \) is a direct ancestor of \( g_{i-\xi}(p) \). Consequently, \( g_i(q \gamma) \) is a direct ancestor of \( g_i(p \gamma) \).

(b) Suppose \( p \notin X_{i-\xi} \), then for some \( s \), where \( 2 \leq s \leq r \),

\[ g_i(p \gamma) = l_s \in \text{os}(G_i) \cap \text{os}(G_{i-\xi}) \]; so by the construction of \( G_i \), \( g_i((k_m, m) \gamma) \) is a direct ancestor of \( g_i(p \gamma) \).

But \( q \gamma \in A(k_i) \), so either \( q \gamma = (k_m, m) \gamma \) and the result is proved; or \( q \in A(X_{i-\xi}) \) in which case \( g_i(q \gamma) \) is a direct ancestor of \( g_i((k_m, m) \gamma) \), by case (a), and hence of \( g_i(p \gamma) \).

Hence the required sequence of plans exists.

Now \( \mathcal{G} = \text{lit}(B(X_n)) \)

\[ = \{ \text{lit}(l) \mid l \in B(X_n) \} \]
\[ = \{ g_n(l) \Theta(G_n) \mid l \in B(X_n) \} \]

by condition (iv)
Therefore $G = G_n$ is the required plan.

3.3.4.13: Corollary: If $\mathcal{F}$ is unsatisfiable, there exists a closed, correct plan for $\mathcal{F}$, constructed using rules (1)A and (2) only.

Proof: If $\mathcal{F}$ is unsatisfiable, then there is an ME-deduction of the empty chain from $\mathcal{F}$, so, by lemma 3.3.4.12, there exists a correct plan $G$ for $\mathcal{F}$ constructed using rules (1)A and (2), such that $os(G) \in G = n$.

3.3.4.14: Corollary: If $\mathcal{F}$ is unsatisfiable, there exist closed, correct plans $G$ and $H$ for $\mathcal{F}$, where $G$ is constructed using (1) and (3)A, and $H$ is constructed using (1) and (3)B.

Proof: This follows from corollary 3.3.4.13 and lemmas 3.2.18 and 3.2.19.

Either of the above corollaries implies the completeness of general plans as follows.
3.3.5: Theorem: Completeness of plans

If $\mathcal{S}$ is unsatisfiable, there exists a closed, correct plan for $\mathcal{S}$.

We are now in a position to demonstrate that the restrictive conditions on reduction and factoring are necessary to ensure soundness, by presenting the following example as promised in section 3.2.

3.3.6: Example: Let $\mathcal{S}$ be the set of clauses:

\[
\{ [P(x), F(a), H(x)], \\
[\neg P(y), Q(y)], \\
[\neg R(z), Q(z), M(z)], \\
[\neg Q(w), \neg R(w)], \\
[\neg M(r), \neg R(r), \neg E(a)] \}
\]

where a is a constant. $\mathcal{S}$ is obviously satisfiable. Consider the graph of figure 3.11: this graph can be constructed using the rules for plan construction with condition (c) on reduction removed, and condition (c) on backfactoring weakened to "y is not an ancestor of x". This graph is closed and is clearly correct, despite the satisfiability of $\mathcal{S}$.
A plan for the set of clauses of example 3.3.6 demonstrating the unsoundness that results when the restrictive conditions on reduction and factoring are removed.

Figure 3.11
3.3.7: Sound and complete subsets of rules

3.3.7.1: Definition: If $R$ is any subset of the rules for constructing plans, we say that $R$ is sound (complete) if, for every set of clauses $\mathcal{G}$, $\mathcal{G}$ is unsatisfiable if (only if) there exists a closed, correct plan for $\mathcal{G}$ constructed using the set $R$ of rules.

By theorem 3.3.3 the set of all rules is sound, so obviously any subset is also sound. By corollary 3.3.4.13 and corollary 3.3.4.14, the sets $\{1A, (2)\}, \{(1), (3)A\}$ and $\{(1), (3)B\}$ are complete, so a superset of any of these sets is also complete. Furthermore, these three sets are minimal in the sense that no subsets of them are complete. This is clear if we observe that sets which do not contain $\{(1)A\}$ are not complete; and that neither $\{(1)A, (3)\}$ nor $\{(1)\}$ are complete since, for example, neither can generate a closed plan for the unsatisfiable set of clauses $\{\{P(x), P(y)\}, \{-P(x), -P(y)\}\}$. We also note that the subsets $\{(1)A, (2), (3)\}$ and $\{(1)A, (2), (3)A\}$ are equivalent in the sense that both generate exactly the same plans for a given set of clauses (corollary 3.2.12). Furthermore, both are equivalent to $\{(1)A, (2), (3)B\}$ in that they generate the same set of closed plans for a given set of clauses.
CHAPTER 4

Constraint Processing

In chapter 3, we described the construction of plans and proved the soundness and completeness of various deduction systems based on them. We have not, however, suggested any methods for unifying the set of constraints produced during the construction of a plan.

In a practical theorem-proving system, it would obviously be unwise to attempt to construct a closed, correct plan in the way suggested by the presentation of chapter 3 (that is, by constructing a closed plan, then verifying its correctness) since constraints introduced early in the derivation may be nonunifiable, so that continuing the derivation past the point where these constraints are produced is pointless. Instead, as each open subproblem is closed, the new constraints this closure introduces should be unified with the constraints already produced, to determine whether the new plan is correct. Consequently, to process the constraint set, we require an algorithm which can efficiently unify the constraints on-line as they are produced. This requirement indicates
which of the existing unification algorithms we should choose as the basis of our constraint processing system, according to the following argument.

Two formulae may be nonunifiable for two reasons. For example, the formulae $F(G(x))$ and $F(a)$ cannot be unified because of the disagreement between the function symbol $G$ and the constant $a$. The second type of nonunifiability is exemplified by the two formulae $F(G(x))$ and $F(x)$, which cannot be unified because $x$ occurs in $G(x)$.

A recent unification algorithm of Baxter [4,5], is based on detecting these two types of nonunifiability separately, and accordingly, operates in two stages: first the transformational stage detects nonunifiability due to incompatible function symbols, then the sorting stage checks that no variable is unified with a formula in which it occurs. This requires time proportional to $nG(n)$, where $n$ is the length of the input formulae, and $G(n)$ is an extremely slow-growing function of $n$. The transformational stage operates in a serial manner on the constraints, and so is particularly suited to our on-line application: the sorting stage is a topological sort of a digraph, and unfortunately, no efficient on-line algorithm is known for this task. However, when a new constraint is added to a previously unified set, only the sorting stage of the
algorithm must be completely repeated. By contrast, other unification algorithms combine the transformational and sorting stages, so that complete reprocessing must be done following the addition of a new constraint. A recent algorithm of Paterson and Wegman [33], although of linear time complexity is of this latter type, and hence is not suited to our purposes. In fact, because of its two-stage structure, Baxter's algorithm appears to be the only one which satisfies our requirement for efficient on-line operation.

Another important consideration when deciding how constraints are to be processed, involves backtracking: a problem which to date has received little or no attention from researchers in the field of mechanical deduction.

At each point in the search for a proof, there is usually a variety of possible actions which can be performed by a theorem-proving system: it must choose the subproblem to work on next, then choose which of several solutions to that subproblem it should try. If the system should fail to solve a subproblem, it must return to an earlier point in the search, and attempt an alternative solution to an earlier subproblem. This action is termed "backtracking". The usual strategy employed in backtracking, is to return to the last point in the search at which there exists an
untried alternative solution. The wastefulness of this exhaustive approach is illustrated by the following example.

4.0: Example: Let $\mathcal{S}$ be the set of clauses:

1. $P(x), R(b), R(x)$
2. $\neg P(x), Q(x)$
3. $\neg P(x), B(x)$
4. $\neg Q(x), K(x)$
5. $\neg Q(x), N(x)$
6. $\neg R(x), K(x)$
7. $\neg R(x), N(x)$
8. $\neg K(x), M(x)$
9. $\neg K(x), S(x)$
10. $\neg N(x), M(x)$
11. $\neg N(x), S(x)$
12. $\neg M(x), \neg B(x)$
13. $\neg S(x), \neg B(x)$
14. $\neg R(x)$
15. $B(a)$

where $a$ and $b$ are constants. Suppose a proof of the unsatisfiability of this set is attempted using model elimination with factoring [27]. To determine the order of alternative solutions, suppose that the rules are tried in the order: contraction, reduction, factoring, extension; and that input clauses for extension are taken from $\mathcal{S}$ in the above order. Selection of subproblems is right to left,
necessary to maintain soundness of model-elimination. The following search is performed, in which A-literals are framed:

(1) \( P(x), R(z), R(x) \)

(16) \( P(b), R(b) \) \hspace{1cm} \text{Factoring}

(17) \( P(b), R(b) \) \hspace{1cm} \text{Extension with (14)}

(18) \( P(b) \) \hspace{1cm} \text{Contraction}

(19) \( P(b), Q(b) \) \hspace{1cm} \text{Extension with (2)}

(20) \( P(b), Q(b), K(b) \) \hspace{1cm} \text{Extension with (4)}

(21) \( P(b), Q(b), X(b), M(b) \) \hspace{1cm} \text{Extension with (8)}

(22) \( P(b), Q(b), X(b), M(b), \neg B(b) \) \hspace{1cm} \text{Extension with (12)}

\hspace{1cm} \text{Backtrack to (20)}

(23) \( P(b), Q(b), X(b), S(b) \) \hspace{1cm} \text{Extension with (9)}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{three backtracks occur here}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

\hspace{1cm} \text{.}

(39) \( P(b), R(b), N(b), S(b), \neg B(b) \) \hspace{1cm} \text{Extension with (13)}

\hspace{1cm} \text{Backtrack to (1)}

(40) \( P(x), R(z), R(z) \) \hspace{1cm} \text{Extension with (14)}

(41) \( P(x), R(b) \) \hspace{1cm} \text{Contraction}

(42) \( P(x), R(b) \) \hspace{1cm} \text{Extension with (14)}

(43) \( P(x) \) \hspace{1cm} \text{Contraction}

(44) \( P(x), Q(x) \) \hspace{1cm} \text{Extension with (2)}

(45) \( P(x), Q(z), X(z) \) \hspace{1cm} \text{Extension with (4)}

(46) \( P(x), Q(z), X(z), M(x) \) \hspace{1cm} \text{Extension with (8)}

(47) \( P(x), Q(z), X(z), M(z), \neg B(z) \) \hspace{1cm} \text{Extension with (12)}
The solution of subproblem \(-B(b)\) is attempted five times before the search eventually backtracks to (1) to try solving \(R(x)\) by extension rather than factoring. This could be avoided if the system was able to observe that the nonunifiability which makes \(-B(b)\) unsolveable, is caused by the factoring of \(R(x)\) to \(R(b)\) at the beginning of the search.

A system for processing the constraints should therefore be able to locate the source of conflict when nonunifiability occurs, in order that the deduction system may backtrack to the correct point in the search.

4.1: The Baxter Unification Algorithm

4.1.1: Definition: A constraint is an unordered pair of formulae. If \(C\) is a set of constraints, we will say that a formula \(p\) is a subformula of \(C\) if \(p\) is a subformula of some formula \(q\), such that \(\{q,r\} \in C\), for some \(r\).
4.1.2: Transformational Stage

The input to this stage is $C$, the set of constraints to be unified. When the algorithm stops, either the original set $C$ is nonunifiable, or the algorithm returns a partition $F_{\text{out}}$ of all subformulae occurring in $C$. $F_{\text{out}}$ has the property that for all substitutions $\theta$, $\theta$ unifies $C$ if and only if $\theta$ unifies $F_{\text{out}}$.

During this stage two sets are manipulated: a set $S$ of constraints, with initial value $C$, and a set $F$ of classes of formulae. $C$ and $S$ may contain repetitions. $F$ is initially $F_{\text{in}}$, the partition of the set of subformulae of $C$ in which each class contains one and only one formula. The final value of $F$ on successful termination of the algorithm is $F_{\text{out}}$, and $S$ is finally empty.

In his description of the algorithm in [5], Baxter allows $F_{\text{in}}$ to contain several identical classes containing the same term, although variables can appear only once. This is because his main concern is the complexity of the algorithm, and since he assumes the constraints to be unified are input as strings of characters, he must allow the repetition of terms in order to avoid the task of identifying multiple occurrences of a term.
We, however, are not considering the unification algorithm in isolation, but as a component of a theorem-prover in which structure is shared and every subformula is represented only once. Consequently, we can restrict every class in $F_{in}$ to be unique, and as a result $F$ is always strictly a partition of a set according to the usual definition. This restriction allows us to make the following definition.

**4.1.2.1: Definition:** If $F$ is a partition of the set of subformulae of $C$, and $p$ and $q$ are subformulae of $C$, then we denote by $[p]_F$ the class in $F$ which contains $p$, and define $p \equiv q \text{ mod } F$ if and only if $[p]_F = [q]_F$. When $F$ is understood from the context, we will write $[p]$ for $[p]_F$.

The algorithm which performs the transformational stage is as follows:
algorithm TRANSFORM(C);
S <-- C;
F <-- F.in;
while S ≠ ∅
  do [ Delete a constraint \{p₁,p₂\} from S;
    | if [p₁] ≠ [p₂]
    |    such that
    |    \[p₁\] contains a term \(f₁(q₁₁,\ldots,q_{₁m})\)
    |    and \[p₂\] contains a term \(f₂(q₂₁,\ldots,q_{₂n})\)
    |    then [ if \(f₁ \neq f₂\)
    |    | then unification fails;
    |    | stop
    |    | else add to S the pairs:
    |    | \{q₁₁,q₂₁\},\ldots,\{q₁ₙ,q₂ₙ\}
    |    | Replace [p₁] and [p₂] by [p₁] ∪ [p₂] in F
  stop

Two important results concerning this algorithm are:

4.1.2.2: Lemma: C is unifiable with mgu θ if and only if
TRANSFORM(C) succeeds, producing partition F.out which has
mgu θ.

4.1.2.3: Lemma: If \(t₁ \equiv t₂ \mod F.out\), where
\(t₁ = f(q₁₁,\ldots,q_{₁n})\) and \(t₂ = f(q₂₁,\ldots,q_{₂n})\) then for each
\(i \in \{1,\ldots,n\}, q₁ᵢ \equiv q₂ᵢ \mod F.out\).

Both these results are proved in [5].
4.1.2: **Sorting stage**

To determine whether or not $F_{\text{out}}$ is unifiable, it is necessary to construct a certain directed graph whose vertex set is $F_{\text{out}}$. This digraph is then topologically sorted: that is, an attempt is made to place the vertices of the digraph in a linear order which preserves the relation defined by the arcs of the digraph. A standard algorithm to perform this task is given in [19]. If the digraph can in fact be sorted, then $F_{\text{out}}$ is unifiable, and the mgu can be determined from the resulting linear order.

4.1.3.1: **Definition:** If $C$ is a set of constraints for which $\text{TRANSFORM}(C)$ succeeds, returning $F_{\text{out}}$, then $D(C)$ is a digraph, where $V(D(C)) = F_{\text{out}}$, and $E(D(C))$ is defined as follows. Suppose there are $m$ classes of $F_{\text{out}}$ containing terms, and let $t_1, \ldots, t_m$ be $m$ terms such that $[t_i] = [t_j]$ if and only if $i = j$. Suppose also that $t_i = f_i(p_{i1}, \ldots, p_{in_i})$ for all $i \in \{1, \ldots, m\}$, then:

$$E(D(C)) = \{([t_i], [p_{ij}]) \mid i \in \{1, \ldots, m\}, j \in \{1, \ldots, n_i\}\}$$

We note that given a particular partition $F_{\text{out}}$, $D(C)$ is unique. This follows from the fact that if a class of $F_{\text{out}}$ contains terms, then those terms all begin with the same function symbol by lemma 4.1.2.2; and by lemma 4.1.2.3, the set of arcs leaving a particular vertex is independent of the term we choose to represent that vertex.

4.1.3.1
Because of the nondeterministic nature of TRANSFORM, we cannot assume that the output partition or digraph are unique. It happens, however, that this is the case; this fact will be proved later in the chapter.

4.1.3.2: Lemma: If C is a set of constraints, then C is unifiable if and only if the digraph D(C) can be topologically sorted (i.e. iff D(C) has no cycles).

This is proved in [5].

We combine lemmas 4.1.2.2 and 4.1.3.2 into the following theorem, which is the basis of the results presented in the rest of this chapter.

4.1.4: Theorem: A set of constraints is unifiable if and only if TRANSFORM(C) succeeds returning partition F.out and the digraph D(C) constructed from F.out has no cycles.

4.2: The modified unification algorithm

All unification algorithms detect unification failure; however, as we illustrated in example 4.0, if a deduction system is to backtrack intelligently, it must be able to go further than this, and locate the source of unification failure. The unification algorithm as described in section
4.1, halts at the first sign of nonunifiability. This does not quite suit our purposes since a set of constraints may be nonunifiable for more than one reason, and we wish to remove all sources of nonunifiability. Consequently, we modify the algorithm so that the transformational stage continues to merge sets even though they may contain terms beginning with different function symbols. The modified algorithm classifies the subformulae of C into sets of formulae: if any of these sets contain incompatible terms, we must then discover how to remove constraints in order to subdivide the sets so that the resulting partition contains no such function symbol clashes.

The sorting stage of the algorithm must be similarly modified. The aim of performing a topological sort on the digraph \( D(C) \), is to determine if a cycle exists. We must, however, enumerate the cycles in order to eliminate them all.

4.2.1: The algorithm CLASSIFY

As with TRANSFORM, the input to CLASSIFY is \( C \), the set of constraints to be unified. When the algorithm halts, it returns a partition \( F \).out of the subformulae of \( C \), and a partition \( F \).out of the subformulae of \( C \) which are terms. Recall that a term is a formula that is not a variable (2.2.6). CLASSIFY manipulates three sets: a set \( S \) of constraints with initial value \( C \), a set \( F \), which is a
partition of the set of subformulae of C, and a set P, which is a partition of the set of all terms which are subformulae of C. F is initially F.in, the partition in which each class contains only one member. Similarly P is initially P.in in which each class contains only one member. As before, C is unifiable with mgu θ, if and only if F.out is unifiable with mgu θ. Each class in F always contains either no classes of F or entire classes of F throughout the execution of CLASSIFY, and if two terms in the same class of F begin with the same function symbol, then they also belong to the same class of F.

4.2.1.1: Definition: If t is a term which is a subformula of C, we denote by \(<t>_P\), the class in P containing t, and write \(t_1 \equiv t_2 \mod P\) if and only if \(<t_1>_P = <t_2>_P\). When no ambiguity is likely we will write \(<t>_P\) for \(<t>_P\). This parallels definition 4.1.2.1 concerning classes of F.

The modified transformational algorithm is:
algorithm CLASSIFY(C);
S <-- C;
F <-- F.in;
P <-- P.in;
while S ≠ ∅
  do [ Delete a constraint \( [p_1, p_2] \) from S;
    |    if \( [p_1] ≠ [p_2] \)
    |    then [ T <-- \( [p_1] \);
    |    | while T contains a term \( t_1 = f(q_{11}, \ldots, q_{1n}) \)
    |    |    do [ Delete from T all terms in \( <t_1> \);
    |    |    |    if \( [p_2] \) contains a term
    |    |    |    | \( t_2 = f(q_{21}, \ldots, q_{2n}) \)
    |    |    |    then [ Add to S the pairs:
    |    |    |    | \( \{q_{11}, q_{21}\}, \ldots, \{q_{1n}, q_{2n}\} \);
    |    |    |    | Replace \( <t_1> \) and \( <t_2> \) by
    |    |    |    | \( <t_1> \cup <t_2> \) in P
    |    |    ] Replace \( [p_1] \) and \( [p_2] \) by \( [p_1] \cup [p_2] \) in F
  ] stop

In the remainder of section 4.2.1, we prove several properties of the algorithm CLASSIFY, and in particular, we show precisely how TRANSFORM and CLASSIFY are related. These proofs refer to the flowcharts of the algorithms, shown in figures 4.1 and 4.2. In the flowchart for CLASSIFY, each decision box is labelled, and each branch out of a decision box is labelled. Hence we can specify an execution (or part of an execution) of CLASSIFY by an alternating sequence of decision boxes and branches. Suppose a path is specified in this way, then we will use the notation \( S(C3, n) \) (for example), to denote the value of the variable \( S \) at the \( n \)-th encounter with the decision box labelled C3 on that path.
The flowchart for the algorithm TRANSFORM.

Figure 4.1
The flowchart for the algorithm CLASSIFY.

Figure 4.2
All our proofs of properties of CLASSIFY are of the same general form. We define some assertion \( H \) about the state of the variables, and attempt to show that it always holds at the point \( C_1 \) in the flowchart. This involves investigating the two paths \( PC_3 \) and \( PC_5 \) from \( C_1 \) to \( C_1 \), showing that if the assertion holds at \( C_1 \) at the beginning of such a loop, it again holds on returning to \( C_1 \). To complete the induction, \( H \) must be shown to hold at the first encounter with \( C_1 \) in the execution of CLASSIFY. The path \( PC_5 \) includes a loop from \( C_3 \) to \( C_3 \), so in general, we must establish the invariance of some other assertion at \( C_3 \) in order to prove the main result. An overview of techniques for proving properties of programs is given in [29].

**4.2.1.2: Example** Figure 4.3 illustrates an execution of \( \text{CLASSIFY}(C) \), where \( C \) is the set of constraints:

\[
\{ \{ F(x,x), v \}, \\
    \{ v, F(f(a), h(y)) \}, \\
    \{ F(u, f(y)), F(h(g(b)), u) \} \\
    \{ h(u), h(f(a)) \} \}
\]

and \( a \) and \( b \) are constants. The set of subformulae of \( C \) is partitioned into six classes:
\[ F_{out} = \{ \begin{array}{l} \{F(x,x), F(f(a),h(y)), v\}, \\ \{F(u,f(y)), F(h(g(b)),u)\}, \\ \{h(u), h(f(a))\}, \\ \{x, u, f(a), f(y), h(y), h(g(b))\}, \\ \{a, y, g(b)\}, \\ \{b\} \end{array} \] \]

The set of subformulae of C which are terms, is partitioned into eight classes:

\[ F_{out} = \{ \begin{array}{l} \{F(x,x), F(f(a),h(y))\}, \\ \{F(u,f(y)), F(h(g(b)),u)\}, \\ \{h(u), h(f(a))\}, \\ \{f(a), f(y)\}, \\ \{h(y), h(g(b))\}, \\ \{a\}, \\ \{g(b)\}, \\ \{b\} \end{array} \] \]
This table illustrates an execution of CLASSIFY(C), where C is the set of constraints of example 4.2.1.2. A number in the column labelled "Introduced", indicates the encounter with CI in the flowchart at which the corresponding constraint first appears; similarly, the "Removed" column indicates the encounter with CI at which the constraint is first absent. (Continued on page 105.)
<table>
<thead>
<tr>
<th>$F(C_1, 6)$</th>
<th>$F(C_1, 7)$</th>
<th>$F(C_1, 8)$</th>
<th>$F(C_1, 9)$</th>
<th>$F(C_1, 10)$</th>
<th>$F(C_1, 11)$</th>
<th>$F_{out}=F(C_1, 12)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(C_1, 6)$</td>
<td>$P(C_1, 7)$</td>
<td>$P(C_1, 8)$</td>
<td>$P(C_1, 9)$</td>
<td>$P(C_1, 10)$</td>
<td>$P(C_1, 11)$</td>
<td>$P_{out}=P(C_1, 12)$</td>
</tr>
<tr>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
<td>$F(x, x)$</td>
</tr>
<tr>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
<td>$F(f(a), h(y))$</td>
</tr>
<tr>
<td>$v$</td>
<td>$v$</td>
<td>$v$</td>
<td>$v$</td>
<td>$v$</td>
<td>$v$</td>
<td>$v$</td>
</tr>
<tr>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
<td>$F(u, f(y))$</td>
</tr>
<tr>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
<td>$F(h(g(b)), u)$</td>
</tr>
<tr>
<td>$h(u)$</td>
<td>$h(u)$</td>
<td>$h(u)$</td>
<td>$h(u)$</td>
<td>$h(u)$</td>
<td>$h(u)$</td>
<td>$h(u)$</td>
</tr>
<tr>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
<td>$h(f(a))$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>$f(a)$</td>
<td>$f(a)$</td>
<td>$f(a)$</td>
<td>$f(a)$</td>
<td>$f(a)$</td>
<td>$f(a)$</td>
<td>$f(a)$</td>
</tr>
<tr>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
</tr>
<tr>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
<td>$h(y)$</td>
</tr>
<tr>
<td>$u$</td>
<td>$u$</td>
<td>$u$</td>
<td>$u$</td>
<td>$h(g(b))$</td>
<td>$h(g(b))$</td>
<td>$h(g(b))$</td>
</tr>
<tr>
<td>$f(y)$</td>
<td>$f(y)$</td>
<td>$h(g(b))$</td>
<td>$h(g(b))$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td>$h(g(b))$</td>
<td>$h(g(b))$</td>
<td>$f(y)$</td>
<td>$f(y)$</td>
<td>$y$</td>
<td>$y$</td>
<td>$y$</td>
</tr>
<tr>
<td>$g(b)$</td>
<td>$g(b)$</td>
<td>$g(b)$</td>
<td>$g(b)$</td>
<td>$g(b)$</td>
<td>$g(b)$</td>
<td>$g(b)$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
</tbody>
</table>

The narrow columns represent the partitions $P$ and $P_{out}$ at successive encounters with $C_1$. In each column, solid horizontal lines divide classes of the partition $F$. Solid lines and dotted lines divide classes of $P$, except of course that the variables are not included. Terms marked with a * are those selected to represent their classes in $P$.

*Figure 4-3 (cont.)*
4.2.1.3: **Definition:** If $F_1$ and $F_2$ are two partitions of the same set, we write $F_1 \leq F_2$ if and only if for every $A_1 \in F_1$ there exists $A_2 \in F_2$ such that $A_1 \subseteq A_2$. If $F_1 \leq F_2$ and $x$ and $y$ are in the same class of $F_1$, then clearly $x$ and $y$ are in the same class of $F_2$.

4.2.1.4: **Lemma:** CLASSIFY(C) halts for every set of constraints C.

**Proof:** We refer to the flowchart for CLASSIFY in figure 4.2.

First we show that $S$ is always finite at $C_1$, and that every path from $C_1$ to $C_1$ is executed in finite time. Since $S$ is initially finite at $C_1$, having been initialised to C, we need only investigate each path from $C_1$ to $C_1$, demonstrating that the finiteness of $S$ is preserved. Note that this investigation will also show that each path is executed in finite time.

1. Consider the path returning to $C_1$ via $PC_3$.

   $$S(C_1,2) = S(C_1,1) - \{p_1,p_2\},$$

   so $S(C_1,2)$ is finite if $S(C_1,1)$ is finite.

2. Now consider the path returning to $C_1$ via $PC_5$. We observe that for each $i$, if $S(C_3,i)$ is finite then $S(C_3,i+1)$ is also finite, and that $T(C_3,i+1)$ contains less terms than $T(C_3,i)$. But $T(C_3,1) = [p_1]$, which contains a finite number of terms, and $S(C_3,1)$ is finite. Therefore, for some integer $n$, $S(C_3,n)$ is
finite and $T(C3,n)$ contains no terms. In view of the latter fact, $C3$ is not encountered again; instead $PC5$ is followed to $C1$ after the $n$-th encounter with $C3$, so that $S(C1,2) = S(C3,n)$, which is finite.

We now note that each loop that returns to $C1$ via $PC5$ reduces the number of classes in $F$. But $F$ is initially finite, so this loop can be traversed only a finite number of times. Also, each traversal of the loop returning to $C1$ via $PC3$ reduces the size of $S$, so this loop can be traversed only a finite number of times in succession. Hence $C1$ can be encountered only a finite number of times in an execution of $CLASSIFY(C)$, and since each loop from $C1$ to $C1$ is traversed in finite time, $CLASSIFY(C)$ must halt.

\[ \]

4.2.1.5: **Lemma:** During the execution of $CLASSIFY(C)$, it is always the case at $C1$, that for two terms $s_1$ and $s_2$: 
\[
<s_1> = <s_2> \text{ if and only if } s_1 \text{ and } s_2 \text{ begin with the same function symbol and } [s_1] = [s_2].
\]

**Proof:** Let $H(F,F)$ be the assertion:

"$<s_1> = <s_2>$ iff $s_1$ and $s_2$ begin with the same function symbol, and $[s_1] = [s_2]$"

Since the classes of $F$.in and $F$.in contain only one formula each, $H(F,F)$ holds initially at $C1$; so to establish the result, we need only show that $H(F,F)$ is invariant at $C1$, by investigating each path from $C1$ to $C1$. 

4.2.1.5
(1) Consider the path returning to C1 via PC3.

\[ F(C1, 2) = P(C1, 1) \]
\[ F(C1, 2) = P(C1, 1) \]

so the result is trivial in this case.

(2) Now consider the path returning to C1 via PC5. We define the following three assertions.

A(P) asserts:

"if \([s_4] = [s_2]\) and \(s_1\) and \(s_2\) begin with the same function symbol,

then \(<s_1> = <s_2>\)"

B(P) asserts:

"if \([s_4] = [s_2]\) and \(s_1\) and \(s_2\) begin with the same function symbol,

then either \(s_4 \in T\) or \(<s_4> = <s_2>\)"

D(P) asserts:

"if \(<s_4> = <s_2>\)

then \(s_4\) and \(s_2\) begin with the same function symbol

and either \([s_4] = [s_2]\)

or \([s_4] = [r_4]\) and \([s_2] = [p_2]\)

or \([s_1] = [r_2]\) and \([s_2] = [p_1]\)"

We show that if A(P), B(P) and D(P) hold at C3, they again hold at C3 after the loop PC6 is traversed.

(a) Suppose \([s_4] = [s_2]\) and that \(s_1\) and \(s_2\) begin with the same function symbol, then:

\(s_1 \equiv s_2 \mod P(C3, 1)\) by A(P(C3, 1))

But \(P(C3, 1) \leq P(C3, 2)\)

\(\therefore s_1 \equiv s_2 \mod P(C3, 2)\)
A(P(C3,2) holds

(b) Suppose \([s_1] = [p_1], [s_2] = [p_2]\) and that \(s_1\) and \(s_2\) begin with the same function symbol. Then by B(P(C3,1)):

either \(s_1 \in T(C3,1)\)

or \(s_1 \equiv s_2 \mod P(C3,1)\).

Suppose \(s_1 \notin T(C3,2)\), then we have two cases.

**case (i)** If \(s_1 \in T(C3,1)\), then \(s_1 \equiv t_4 \mod P(C3,1)\), so by D(P(C3,1)), \(s_1\) and \(t_4\) begin with the same function symbol. Also \([s_2] = [p_2]\), and \(s_2\) begins with the same function symbol as \(s_4\) (and hence \(t_4\)), so that path PC8 is followed. Let \(t_2 \in [p_2]\) be the term selected; then \(t_2\) and \(s_2\) begin with the same function symbol, and \([s_2] = [t_2]\) so by A(P(C3,1)), we have \(s_2 \equiv t_2 \mod P(C3,1)\), and therefore, \(s_1 \equiv s_2 \mod P(C3,2)\).

**case (ii)** If \(s_1 \notin T(C3,1)\) then \(s_1 \equiv s_2 \mod P(C3,1)\), and since \(P(C3,2) \geq P(C3,1)\), we have \(s_1 \equiv s_2 \mod P(C3,2)\).

So in both cases, B(P(C3,2)) holds.

(c) Suppose \(s_1 \equiv s_2 \mod P(C3,2)\); then we have two cases.

**case (i)** \(s_1 \equiv s_2 \mod P(C3,1)\), so by D(P(C3,1)):

\(s_1\) and \(s_2\) begin with the same function symbol and:

either \([s_1] = [s_2]\)
or \(([s_1] = [p_1] \text{ and } [s_2] = [p_2])\)

or \(([s_1] = [p_1] \text{ and } [s_2] = [p_4])\)

**case (ii)\(^1\)**  \(s_1 \equiv t_1 \mod \text{P(C3,1)}\)

and \(s_2 \equiv t_2 \mod \text{P(C3,1)}\)

So by \(\text{D(P(C3,1))}\), \(s_4\) and \(t_4\) begin with the same function symbol. But \(t_4\) and \(t_2\) begin with the same function symbol, so \(s_1\) and \(s_2\) begin with the same function symbol. Also by \(\text{D(P(C3,1))}\):

\[
\begin{bmatrix}
[s_1] = [t_1] \\
\text{or } ([s_1] = [p_1] \text{ and } [t_1] = [p_2]) \\
\text{or } ([s_1] = [p_2] \text{ and } [t_1] = [p_4])
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
[s_2] = [t_2] \\
\text{or } ([s_2] = [p_4] \text{ and } [t_2] = [p_2]) \\
\text{or } ([s_2] = [p_2] \text{ and } [t_2] = [p_4])
\end{bmatrix}
\]

Now \([t_1] = [p_1]\), \([t_2] = [p_2]\) and \([p_4] \neq [p_2]\), so the above reduces to:

\[
\begin{bmatrix}
[s_1] = [p_1] \\
\text{or } ([s_1] = [p_2] \text{ and } [t_1] = [p_2])
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
[s_2] = [p_2] \\
\text{or } ([s_2] = [p_4] \text{ and } [t_2] = [p_2])
\end{bmatrix}
\]

From this we can deduce:

\([s_1] = [s_2]\)
or ([s₁] = [p₁] and [s₂] = [p₂])

or ([s₁] = [p₂] and [s₂] = [p₄])

So in both cases, D(P(C3, 2)) holds.

Now consider the path from C₁ to C₁ via PC₅, and suppose that C₃ is encountered n times on this path. We assume B(P(C1, 1), F(C1, 1)) holds, so that s₁ ≡ s₂ mod P(C1, 1) if and only if both s₁ ≡ s₂ mod F(C1, 1) and s₁ and s₂ begin with the same function symbol.

But F(C₃, 1) = F(C₁, 1)

P(C₃, 1) = P(C₁, 1)

and T(C₃, 1) = [p₁]

Therefore A(P(C₃, 1)), B(P(C₃, 1)) and D(P(C₃, 1)) hold, so by the invariance of A, B and D at C₃, we have

A(P(C₃, m)), B(P(C₃, m)) and D(P(C₃, m)).

Now F(C₁, 2) = F(C₁, 1) - {([p₁], [p₂]) U ([p₁] U [p₂])}

and P(C₁, 2) = P(C₃, m)

(A) Suppose that s₁ ≡ s₂ mod F(C₁, 2) and that s₄ and s₂ begin with the same function symbol, then:

either s₁ ≡ s₂ mod F(C₁, 1)

so by A(P(C₃, m)) we have:

  s₁ ≡ s₂ mod F(C₃, m)

  • • s₁ ≡ s₂ mod F(C₁, 2)

or s₁ ≡ p₄ mod F(C₁, 1)

and s₂ ≡ p₂ mod F(C₁, 1)

but by B(P(C₃, m)) we have:

  s₁ ∈ T(C₃, m)
or \( s_4 \equiv p_2 \mod F(C_3, m) \)

However, since \( C_3 \) is encountered only \( m \) times, \( T(C_3, m) \) contains no terms.

• \( s_4 \equiv s_2 \mod F(C_3, m) \)

• \( s_4 \equiv s_2 \mod F(C_1, 2) \)

(B) Suppose that \( s_4 \equiv s_2 \mod F(C_1, 2) \)

then \( s_4 \equiv s_2 \mod F(C_3, m) \)

so by \( D(F(C_3, m)) \), \( s_4 \) and \( s_2 \) begin with the same function symbol and:

either \( s_4 \equiv s_2 \mod F(C_1, 1) \)

so \( s_4 \equiv s_2 \mod F(C_1, 2) \) since \( F(C_1, 2) \geq F(C_1, 1) \)

or \( s_4 \equiv p_4 \mod F(c_1, 1) \)

and \( s_2 \equiv p_2 \mod F(C_1, 1) \)

• \( s_4 \equiv p_4 \mod F(C_1, 2) \)

and \( s_2 \equiv p_2 \mod F(C_1, 2) \) since \( F(C_1, 2) \geq F(C_1, 1) \)

but \( p_4 \equiv p_2 \mod F(C_1, 2) \)

• \( s_4 \equiv s_2 \mod F(C_1, 2) \)

or \( s_4 \equiv p_2 \mod F(C_1, 1) \)

and \( s_2 \equiv p_4 \mod F(C_1, 1) \)

so by the same reasoning as in the second alternative:

\( s_4 \equiv s_2 \mod F(C_1, 2) \)

So for the path from \( C_1 \) to \( C_1 \) via \( PC5 \),

\( H(F(C_1, 2), F(C_1, 2)) \) holds.
Hence \( H(P,F) \) is invariant at \( C1 \), and since \( H(P,F) \) holds initially at \( C1 \), the result is proved.

We now show how the algorithms CLASSIFY and TRANSFORM are related.

4.2.1.6: Lemma:
(a) If TRANSFORM\((C)\) succeeds returning partition \( F\text{-out} \), then there is an execution of CLASSIFY\((C)\) returning partition \( F\text{-out} \), where in each class of \( F\text{-out} \), all terms begin with the same function symbol.

(b) If TRANSFORM\((C)\) fails, then there is an execution of CLASSIFY\((C)\) returning partition \( F\text{-out} \), where some class of \( F\text{-out} \) contains terms beginning with different function symbols.

Proof:
(a) Let \( H(F',S') \) be the assertion:

"in each class of \( F' \), all terms begin with the same function symbol

and there exists an execution of CLASSIFY\((C)\) during which \( S \) and \( F \) have values \( S' \) and \( F' \) at some encounter with \( C1 \)"

Consider an execution of TRANSFORM\((C)\). Clearly \( H(F,S) \) holds initially at \( T1 \). We now show, by investigating
(1) Suppose \([p_4]\) contains no terms, then path PC5 is followed to C1, where:

\[
F(C1, 2) = F(C3, 1) - \{(p_4), (p_2)\} \cup \{(p_4) \cup (p_2)\}
\]

\[
= F(C1, 1) - \{(p_4), (p_2)\} \cup \{(p_4) \cup (p_2)\}
\]

\[
S(C1, 2) = S(C3, 1)
\]

\[
= S(C1, 1) - \{(p_4, p_2)\}
\]

(2) Suppose \([p_4]\) contains a term \(t_4\), so that by (iii), \([p_2]\) contains no terms. Path PC6 is followed to C4, where:

\[
T(C4, 1) = T(C3, 1) - \langle t_4 \rangle
\]

\[
= \{p_4\} - \langle t_4 \rangle
\]

But by \(F(T1, 1), S(T1, 1)\), all terms in \([p_4]\) begin with the same function symbol, so by lemma 4.2.1.5, all terms in \([p_4]\) are in \(\langle t_4 \rangle\), so that \(T(C4, 1)\) contains no terms.

Now \([p_2]\) contains no terms, so path PC7 is followed to C3 where:

\[
F(C3, 2) = F(C3, 1)
\]

\[
S(C3, 2) = S(C3, 1),
\]

and since \(T(C3, 2) = T(C4, 1)\) which contains no terms, path PC5 is followed to C1, where:

\[
F(C1, 2) = F(C3, 2) - \{(p_4), (p_2)\} \cup \{(p_4) \cup (p_2)\}
\]

\[
= F(C1, 1) - \{(p_4), (p_2)\} \cup \{(p_4) \cup (p_2)\}
\]

\[
S(C1, 2) = S(C3, 2) = S(C3, 1)
\]

\[
= S(C1, 1) - \{(p_4, p_2)\}
\]

But \(F(T1, 1), S(T1, 1)\) holds so that:

\[
F(C1, 1) = F(T1, 1)
\]
and \( S(C_1,1) = S(T_1,1) \)

So in both cases:

\[
F(C_1,2) = F(T_1,2)
\]

and \( S(C_1,2) = S(T_1,2) \)

Hence \( H(F(T_1,2), S(T_1,2)) \) holds for this path.

**Path PT3:** On returning to \( T_1 \), we have:

\[
F(T_1,2) = F(T_1,1) - \{[p_1], [p_2]\} \cup \{[p_1] \cup [p_2]\}
\]

and \( S(T_1,2) = S(T_1,1) - \{[p_1], [p_2]\} \)

\[
\cup \{[q_{11}, q_{21}], ..., [q_{1n}, q_{2n}]\}
\]

Also, for this path to be traversed the following conditions must hold:

(i) \( S(T_1,1) \neq \emptyset \)

(ii) \([f_1] \neq [p_2] \)

(iii) both \([p_1] \) and \([p_2] \) contain terms, respectively

\[
t_1 = f_1(q_{11}, ..., q_{1m}), \text{ and } t_2 = f_2(q_{21}, ..., q_{2n})
\]

(iv) \( f_1 = f_2 \)

Every class of \( F(T_1,2) \) is either a class of \( F(T_1,1) \), in which case all terms in it begin with the same function symbol, by \( H(F(T_1,1), S(T_1,1)) \); or it is the union of classes \([p_1] \) and \([p_2] \) of \( F(T_1,1) \). In the latter case, since \( t_1 \in [p_1] \) and \( t_1 \) begins with function symbol \( f_1 \), all terms in \([p_1] \) begin with \( f_1 \); by \( H(F(T_1,1), S(T_1,1)) \); similarly, all terms in \([p_2] \) begin with \( f_2 \). But \( f_1 = f_2 \) by (iv), so that all terms in \([p_1] \cup [p_2] \) begin with the same function symbol.
Now by (1) and (ii), CLASSIFY can follow PC2 and PC4 to C3, where:

$$F(C3,1) = F(C1,1)$$

$$S(C3,1) = S(C1,1) = S(C2,1) - \{[p_1,p_2]\}$$

By (iii), path PC6 is followed to C4 where, by the same reasoning as that employed in case (2) for path PT2 above, T(C4,1) contains no terms. Again, by (iii), path PC8 is followed to C3, where:

$$S(C3,2) = S(C3,1) \cup \{[q_{11},q_{21}],\ldots,[q_{1n},q_{2n}]\}$$

$$F(C3,2) = F(C3,1)$$

Since T(C3,2) = T(C4,1) which contains no terms, path PC5 is followed to C1 where:

$$S(C1,2) = S(C3,2)$$

$$= S(C1,1) - \{[p_1,p_2]\}$$

$$\cup \{[q_{11},q_{21}],\ldots,[q_{1n},q_{2n}]\}$$

$$F(C1,2) = F(C3,1) - \{[p_1],[p_2]\} \cup \{[p_1],([p_2])\}$$

$$= F(C1,1) - \{[p_1],[p_2]\} \cup \{[p_1],([p_2])\}$$

But H(F(T1,1),S(T1,1)) so that:

$$F(C1,1) = F(T1,1)$$

and $$S(C1,1) = S(T1,1)$$

Therefore:

$$F(C1,2) = F(T1,2)$$

and $$S(C1,2) = S(T1,2)$$

Hence H(F(T1,2),S(T1,2)) holds for this path.
This proves the invariance of \( H(F,S) \) at \( T_1 \), so since \( H(F,S) \) is initially true at \( T_1 \), \( H(F,S) \) always holds at \( T_1 \). Consequently if \( \text{TRANSFORM}(C) \) succeeds, returning \( F.out \), then there exists an execution of \( \text{CLASSIFY}(C) \) which encounters \( C_1 \) with values \( F.out \) and \( \emptyset \) for \( F \) and \( S \) respectively, and the result is proved.

(b) Now suppose \( \text{TRANSFORM}(C) \) fails. By part (a), there exists an execution of \( \text{CLASSIFY}(C) \) which encounters \( C_1 \) with the same values of \( S \) and \( F \) that obtain at the last encounter with \( T_1 \) in the execution of \( \text{TRANSFORM}(C) \). Since \( \text{TRANSFORM}(C) \) fails, the path \( \text{EXIT} \) must be followed to the point \( \text{FAIL} \), so that \([p_1]\) and \([p_2]\) contain terms \( t_1 \) and \( t_2 \) respectively, which begin with different function symbols. Since \([p_1] \neq [p_2]\), \( \text{CLASSIFY} \) can follow the loop returning to \( C_1 \) via \( FC5 \), and as a result:

\[
F(C_1,2) = F(C_1,1) - ([p_1],[p_2]) \cup ([p_1] \cup [p_2])
\]

Hence \( F(C_1,2) \) contains a class in which two terms, namely \( t_1 \) and \( t_2 \), begin with different function symbols, and since \( F.out \in F(C_1,2) \), the same can be said for \( F.out \).

We now prove another property of \( \text{CLASSIFY} \) that will be important later on: namely, if two terms beginning with the
same function symbol are identified by CLASSIFY in that they are placed in the same class, then corresponding subformulae of those terms will be similarly identified. More precisely:

4.2.1.7: Lemma: Suppose the output partitions of CLASSIFY are \( F_{\text{out}} \) and \( P_{\text{out}} \). If \( s_1 = g(p_{11}, \ldots, p_{1m}), \)
\( s_2 = g(p_{21}, \ldots, p_{2m}), \) and \( s_1 \equiv s_2 \mod P_{\text{out}} \), then for each \( i \in \{1, \ldots, m\} \), \( F_{a_i} \equiv P_{2i} \mod F_{\text{out}} \).

Proof: Let \( B(F,P,S) \) assert that:

"if \( s_1 \equiv s_2 \mod P \), then for each \( i \in \{1, \ldots, m\} \), there exists an integer \( k \geq 1 \), and formulae \( r_1, \ldots, r_k, w_1, \ldots, w_k \) such that:

\[
\begin{align*}
P_{a_i} &= r_1 \\
P_{2i} &= w_k \\
r_j &\equiv w_j \mod F \text{ for all } j \in \{1, \ldots, k\} \\
\{w_j, r_{j+1}\} &\in S \text{ for all } j \in \{1, \ldots, k-1\} 
\end{align*}
\]

We will ultimately show that \( B(F,P,S) \) always holds at \( C_1 \); first, however, we show that if \( B(F,P,S \cup \{p_{4}, p_{2}\}) \) holds at \( C_3 \), then it again holds at \( C_3 \) after the loop \( PC_6 \) has been traversed.

Path PC7: On this path, values of \( F \), \( P \), and \( S \) are unchanged so the result holds.

Path PC8: In this case:

\[
\begin{align*}
F(C_3, 2) &= F(C_3, 1) \\
P(C_3, 2) &= P(C_3, 1) - \{<t_1>, <t_2>\} \cup \{<t_4> \cup <t_2>\} \\
S(C_3, 2) &= S(C_3, 1) \cup \{\{a_{11}, a_{21}\}, \ldots, \{a_{1n}, a_{2n}\}\}
\end{align*}
\]

Now if \( s_1 \equiv s_2 \mod P(C_3, 2) \) then:

either \( s_1 \equiv s_2 \mod P(C_3, 2), \)

4.2.1.7
so the result holds in view of the fact that
\[ F(C3, 2) \]  \(=\)  \(F(C3, 1)\) and \(S(C3, 1) \subseteq S(C3, 2)\)

or

\[ s_1 \equiv t_1 \mod F(C3, 1) \]

\[ s_2 \equiv t_2 \mod F(C3, 1), \]

so \(f=g\) and \(n=m\), and by hypothesis, for each \(i \in \{1, \ldots, m\}\):

(a) there exists \(k_i \geq 1\), \(r_i, \ldots, r_{k_i}, w_i, \ldots, w_{k_i}\) such that:

\[ p_{i1} = r_i \]

\[ q_{i1} = w_{k_i} \]

\[ r_i \equiv w_j \mod F(C3, 1) \] for all \(j \in \{1, \ldots, k_i\}\)

\(\{w_i, r_{i+1}\} \in S(C3, 1) \cup \{\{p_i, p_{i+1}\}\} \)

for all \(j \in \{1, \ldots, k_i - 1\}\)

(b) there exists \(k_2 \geq 1\), \(r'_i, \ldots, r'_{k_2}, w'_i, \ldots, w'_{k_2}\) such that:

\[ p_{21} = r'_i \]

\[ q_{21} = w'_{k_2} \]

\[ r'_i \equiv w'_j \mod F(C3, 1) \] for all \(j \in \{1, \ldots, k_2\}\)

\(\{w'_i, r'_{i+1}\} \in S(C3, 1) \cup \{\{p_i, p_{i+1}\}\} \)

for all \(j \in \{1, \ldots, k_2 - 1\}\)

But \(\{q_{k_1}, q_{k_2}\} \in S(C3, 2)\)

\(S(C3, 1) \subseteq S(C3, 2)\)

and \(F(C3, 2) = F(C3, 1)\)

Now if we let \(k = k_1 + k_2\), rename \(r_i, \ldots, r_{k_2}\) as \(w_k, \ldots, w_{k_1} +1\)

and rename \(w'_i, \ldots, w'_{k_2}\) as \(r_k, \ldots, r_{k_1} +1\), then:

\(\exists k \geq 1, r_k, \ldots, r_k, w_k, \ldots, w_k\) such that:
\[ P_{a_1} = r_4 \]
\[ P_{2i} = w_k \]
\[ r_j \equiv w_j \mod F(C3, 2) \quad \text{for all } j \in \{1, \ldots, k\} \]
\[ \{w_j, r_j+4\} \in S(C3, 2) \cup \{\{p_1, p_2\}\} \]
\[ \text{for all } j \in \{1, \ldots, k-1\} \]

This is the required result.

Having shown that \( H(F, P, S \cup \{\{p_1, p_2\}\}) \) is invariant at \( C3 \), we are now in a position to show that \( H(F, P, S) \) always holds at \( C1 \). Initially, \( F = P \cdot in \) and \( s_1 \equiv s_2 \mod P \cdot in \) implies \( s_4 = s_2 \), since each class in \( P \cdot in \) contains only one formula. Therefore, for each \( i \), \( p_{a_1} = p_{2i} \), so that \( p_{4i} \equiv p_{2i} \mod F \cdot in \), and \( H(F, P, S) \) holds initially at \( C1 \). We must now investigate each path from \( C1 \) to \( C1 \), showing that \( H(F, P, S) \) is invariant at \( C1 \).

(1) For the path returning to \( C1 \) via \( PC3 \), we have:

\[ F(C1, 2) = F(C1, 1) \]
\[ P(C1, 2) = P(C1, 1) \]

and \( S(C1, 2) = S(C1, 1) - \{\{p_1, p_2\}\} \)

where \( r_4 \equiv p_2 \mod F(C1, 1) \)

Suppose \( s_4 \equiv s_2 \mod P(C1, 2) \)

then \( s_4 \equiv s_2 \mod P(C1, 1) \)

so by \( H(F(C1, 1), P(C1, 1), S(C1, 1)) \), there exists an integer \( k \geq 1 \), and formulae \( r_4, \ldots, r_k, w_4, \ldots, w_k \) such that:
\[ P_{a1} = r_a \]
\[ F_{21} = w_k \]
\[ r_j \equiv w_j \mod F(C1,1) \text{ for all } j \in \{1, \ldots , k\} \]
\[ \{w_j, r_j + a\} \in S(C1,1) \text{ for all } j \in \{1, \ldots , k-1\} \]

which is equivalent to:
\[ P_{a1} = r_a \]
\[ F_{21} = w_k \]
\[ r_j \equiv w_j \mod F(C1,2) \text{ for all } j \in \{1, \ldots , k\} \]
\[ \{w_j, r_j + a\} \in S(C1,2) \cup \{\{p_4, p_2\}\} \]

for all \( j \in \{1, \ldots , k\} \)

where \( p_4 \equiv p_2 \mod F(C1,2) \)

Next we derive the same result for the other path, then show that it is equivalent to the result we require.

(2) For the path returning to C1 via PC5, we have:
\[ F(C1,2) = F(C3,1) - \{\{p_1\}, \{p_2\}\} \cup \{\{p_1\} \cup \{p_2\}\} \]
\[ P(C1,2) = P(C3,1) \]
\[ S(C1,2) = S(C3,1) \]

where C3 is encountered l times on the path.

Now \( H(F(C1,1), P(C1,1), S(C1,1)) \) holds, by hypothesis, and
\[ F(C3,1) = F(C1,1) \]
\[ P(C3,1) = P(C3,1) \]
\[ S(C3,1) = S(C1,1) - \{\{p_4, p_2\}\} \]

so that \( H(F(C3,1), P(C3,1), S(C3,1) \cup \{\{p_4, p_2\}\} ) \) holds.

But \( H(F, P, S \cup \{\{p_4, p_2\}\} ) \) has been proved invariant at C3, so that \( H(F(C3,1), P(C3,1), S(C3,1) \cup \{\{p_4, p_2\}\} ) \) holds.
Suppose $s_k \equiv s_2 \mod F(C_1, 2)$

then $s_k \equiv s_2 \mod F(C_3, 1)$

and therefore, for each $i$, there exists $k \geq 1$, $r_1, \ldots, r_k, w_1, \ldots, w_k$ such that:

$p_{a_1} = r_1$

$p_{2^1} = w_k$

$r_j \equiv w_j \mod F(C_3, 1)$ for all $j \in \{1, \ldots, k\}$

$\{w_j, r_{j+1}\} \in S(C_3, 1) \cup \{\{p_1, p_2\}\}$

for all $j \in \{1, \ldots, k-1\}$

Since $F(C_1, 2) \geq F(C_3, 1)$, this is equivalent to:

$p_{a_1} = r_1$

$p_{2^1} = w_k$

$r_j \equiv w_j \mod F(C_1, 2)$ for all $j \in \{1, \ldots, k\}$

$\{w_j, r_{j+1}\} \in S(C_1, 2) \cup \{\{p_1, p_2\}\}$

for all $j \in \{1, \ldots, k-1\}$

where $p_1 \equiv p_2 \mod F(C_1, 2)$

This is exactly the result obtained in case (1).

If for all $j \in \{1, \ldots, k-1\}$, $\{w_j, r_{j+1}\} \neq \{p_1, p_2\}$, then the result is proved, so we assume the contrary. Now let $j_1$ and $j_2$ be the least and greatest integers respectively, such that:

$\{w_{j_1}, r_{j_2+1}\} = \{p_1, p_2\} = \{w_{j_2}, e_{j_2+1}\}$

Then $r_{j_1} \equiv w_{j_1} \equiv r_{j_1+1} \equiv w_{j_2} \equiv r_{j_2+1} \equiv w_{j_2+1} \equiv F(C_1, 2)$. So if we let $k_1 = k - j_2 + j_4 - 1$; rename $w_1, \ldots, w_k$ as $w_1, \ldots, w_k$; and rename $r_1, \ldots, r_k$ as $r_{j_1+1}, \ldots, r_k$, we then have $k_1 \geq 1$ since $1 \leq j_4 \leq j_2 < k$, so that:

4.2.1.7
\[ \exists k_4 \geq 1, r_4, \ldots, r_{k_4}, w_4, \ldots, w_{k_4} \text{ such that:} \]
\[ p_{i_1} = r_{i_1} \]
\[ F_{2i} = w_{i_1} \]
\[ r_i \equiv w_i \mod F(C_i, 2) \text{ for all } j \in \{1, \ldots, k_4\} \]
\[ \{w_j, r_{j+1}\} \in S(C_i, 2) \text{ for all } j \in \{1, \ldots, k_4-1\} \]

which is the required result.

Therefore \( H(F, P, S) \) holds at \( C_1 \), and in particular \( H(F.out, P.out, \emptyset) \), so:

If \( s_4 \equiv s_2 \mod F.out \), then for each \( i \in \{1, \ldots, n\} \) there exists \( k \geq 1 \), and \( r_4, \ldots, r_k, w_4, \ldots, w_k \) such that:
\[ p_{i_1} = r_{i_1} \]
\[ F_{2i} = w_k \]
\[ r_i \equiv w_i \mod F.out \text{ for all } j \in \{1, \ldots, k\} \]
\[ \{w_j, r_{j+1}\} \in \emptyset \text{ for all } j \in \{1, \ldots, k-1\} \]

In view of this last result, \( k=1 \), and therefore \( p_4 \equiv p_2 \mod F.out \).

\[ \square \]

4.2.2: The Automaton for a Constraint Set

\( \text{CLASSIFY}(C) \) divides the set of subformulae of \( C \) into classes of formulae which must be unifiable for \( C \) to be unifiable: therefore, if two terms \( t_1 \) and \( t_2 \) begin with different function symbols and occur in the same class, then \( C \) is not unifiable. By inspecting each class, we can
discover every such pair of incompatible terms. We now introduce a mechanism for determining why two incompatible terms are in the same class: that is, for finding all the constraints responsible for this situation. Example 4.2.2.15 at the end of this section, illustrates the concepts introduced here.

4.2.2.1: **Definition**: If $C$ is a set of constraints, denote by $N(C)$ the set of all function symbols occurring in $C$. We then define:

$$\text{degree}(C) = \max_{f \in N(C)} \text{degree}(f)$$

and $N(C) = \{i \mid 1 \leq i \leq \text{degree}(C)\}$

4.2.2.2: **Definition**: If $C$ is a set of constraints, then $A(C)$ is a labelled, directed graph, where:

- $V(A(C)) = \{p \mid p$ is a subformula of $C\}$
- $I(A(C)) = C \cup (N(C) \times N(C))$
- $E(A(C)) = \text{TRANS}(A(C)) \cup \text{PUSH}(A(C)) \cup \text{POP}(A(C))$

where $\text{TRANS}(A(C))$, $\text{PUSH}(A(C))$ and $\text{POP}(A(C))$ are mutually disjoint sets of arcs defined by:

$$\text{TRANS}(A(C)) = \{(p_1, f, p_2) \mid \{p_1, p_2\} = \varepsilon \in C\}$$

$$\text{PUSH}(A(C)) = \{(p, (f, i), t) \mid p$ and $t$ are subformulae of $C,$ $t = f(p_{i+1}, \ldots, p_n),$ and $p = p_i$ for some $i \in \{1, \ldots, n\}\}$$

$$\text{POP}(A(C)) = \{(t, (f, i), p) \mid p$ and $t$ are subformulae of $C,$ $t = f(p_{i-1}, \ldots, p_n),$ and $p = p_i$ for some $i \in \{1, \ldots, n\}\}$$
If $e = (p_1, \text{label}, p_2) \in E(A(C))$, we denote by $e^{-1}$ the ordered triple $(p_2, \text{label}, p_1)$. Then $(e^{-1})^{-1} = e$ if and only if $e^{-1} \in \text{TRANS}(A(C))$; and $e \in \text{PUSH}(A(C))$ if and only if $e^{-1} \in \text{FCP}(A(C))$.

Note that $A(C)$ can be regarded as a nondeterministic, finite, pushdown automaton [1], where $V(A(C))$ is the set of states, (which we will also refer to as vertices of $A(C)$, and subformulæ of $C$), and the transition function is defined in the obvious way by the arcs. The npda $A(C)$ has unspecified initial and final states; input alphabet $C$; and pushdown alphabet $M(C) \times N(C)$, which we will henceforth refer to as $Z$. Accordingly, we call $A(C)$ the automaton for $C$, and make the following definitions.

4.2.2.3: Definition: If $X$ is any finite set, a word of length $n$ over $X$, is any sequence of elements of $X$ of length $n$. The word of length 0 is denoted by $\emptyset$; the set of all words of positive length over $X$ by $X^*$; and the set of all words over $X$ by $X^|=$. We will denote the length of a word $x$ by $|x|$, and denote concatenation of words by juxtaposition.

4.2.2.4: Definition: If $a \in C^*$, $\gamma \in Z^*$, and $p \in V(A(C))$, then $(p,a,\gamma)$ is called a configuration of $A(C)$, and $p,a$ and $\gamma$ are called the state, input and stack of the configuration, respectively.
4.2.2.5: **Definition:** If \( e \in E(A(C)) \), we define a relation \( \{ e \} \) on the set of configurations of \( A(C) \) as follows: let \( e = (p_1, \text{label}, p_2) \), then \( (q_1, a_1, \gamma_1) \{ e \} (q_2, a_2, \gamma_2) \) if and only if:

(i) \( q_1 = p_1, q_2 = p_2 \)

and (ii) (a) if \( e \in \text{TRANS}(A(C)) \), and label = \( \not\epsilon \in C \)

then \( a_1 = \not\epsilon a_2 \)

\( \gamma_1 = \gamma_2 \)

(b) if \( e \in \text{PUSH}(A(C)) \), and label = \((f, i) \in Z \)

then \( a_1 = a_2 \)

\( \gamma_2 = (f, i) \gamma_1 \)

(c) if \( e \in \text{POP}(A(C)) \), and label = \((f, i) \in Z \)

then \( a_1 = a_2 \)

\( \gamma_1 = (f, i) \gamma_2 \)

4.2.2.6: **Definition:** An alternating sequence of \( n+1 \) configurations and \( n \) arcs of \( A(C) \):

\( (p_1, a_1, \gamma_1), e_1, (p_2, a_2, \gamma_2), \ldots, (p_n, a_n, \gamma_n), e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1}) \)

is called a **chain of length** \( n \) in \( A(C) \) from \( (p_1, a_1, \gamma_1) \) to \( (p_{n+1}, a_{n+1}, \gamma_{n+1}) \), if and only if:

\( (p_i, a_i, \gamma_i) \{ e_i \} (p_{i+1}, a_{i+1}, \gamma_{i+1}) \) for all \( i \in \{1, \ldots, n\} \)

4.2.2.7: **Definition:** For each integer \( n \geq 0 \), we define a relation \( \{ n \} \) on the set of configurations of \( A(C) \) as follows:

(i) \( (p_1, a_1, \gamma_1) \{ n \} (p_2, a_2, \gamma_2) \) if and only if \( p_1 = p_2, a_1 = a_2, \) and \( \gamma_1 = \gamma_2 \)

4.2.2.7
(ii) If $n > 0$, $(p_4, a_4, \gamma_4) \vdash^n (p_2, a_2, \gamma_2)$ if and only if there is a chain of length $n$ in $A(C)$ from $(p_4, a_4, \gamma_4)$ to $(p_2, a_2, \gamma_2)$.

We abbreviate $\vdash^1$ as $\vdash$, and also define the relation $\vdash^*$ on the set of configurations of $A(C)$ as follows:

$$(p_4, a_4, \gamma_4) \vdash^* (p_2, a_2, \gamma_2)$$

iff $$(p_4, a_4, \gamma_4) \vdash^n (p_2, a_2, \gamma_2)$$ for some $n \geq 0$

\[ 4.2.2.8 \] Some obvious consequences of the above definitions are:

(i) $$(p, a, \gamma) \vdash^* (q, b, \gamma_2)$$ if and only if

$$(p, ac, \gamma_1) \vdash^* (q, bc, \gamma_2)$$ for any $c \in C^*$

(ii) If $$(p, a, \gamma_1) \vdash^* (q, b, \gamma_2)$$ then for any $\beta \in Z^*$

$$(p, a, \gamma_1 \beta) \vdash^* (q, b, \gamma_2 \beta)$$

(iii) If $$(p, a, \gamma_1) \vdash (q, b, \gamma)$$ then either $a = b$ or $\gamma_1 = \gamma_2$

(iv) If $$(p, a, \gamma_1 \beta) \vdash (q, b, \gamma_2 \beta)$$ then $$(p, a, \gamma_1) \vdash (q, b, \gamma_2)$$

(v) $\vdash^*$ is transitive.

\[ 4.2.2.8 \] Lemma: If $$(p_4, a_4, \gamma_4), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})$$ is a chain in $A(C)$ such that $a_{n+1} = 0$, then there exists a chain $$(p_{n+1}, b_{n+1}, \gamma_{n+1}), e_n^{-1}, \ldots, e_1^{-1}, (p_4, b_4, \gamma_4)$$ where $b_4 = 0$ and $b_{n+1} = \hat{a}_4$. We define $\hat{a}$ as the word obtained by arranging the elements of $a$ in reverse order.

\[ \text{Proof:} \] By induction on $n$.

\[ \text{Basis:} \] $n = 1$, we have three cases to consider:

(a) $e_1 = (p_4, \gamma, p_2) \in \text{TRANS}(A(C))$. 

\[ 4.2.2.9 \]
In this case \( e_1 = \varepsilon, \quad \gamma_1 = \gamma_2 \) and 
\[ e_1^{-1} = (p_2, \varepsilon, p_1) \in \text{TRANS}(A(C)) \]
Clearly \((p_2, \varepsilon, \gamma_2) \uparrow e_1^{-1} \downarrow (p_1, 0, \gamma_2)\), so that 
\((p_2, \hat{a}_1, \gamma_2), \quad e_1^{-1}, \quad (p_1, \varepsilon, \gamma_1)\) is a chain in \(A(C)\).

(b) \( e_4 = (p_4, (i, i), p_2) \in \text{PUSH}(A(C))\).

In this case \( a_1 = a_2 = 0, \quad \gamma_2 = (i, i) \gamma_4 \) and 
\[ e_4^{-1} = (p_2, (i, i), p_4) \in \text{PCP}(A(C)) \]
So therefore \((p_2, \hat{a}_4, \gamma_2) \uparrow e_4^{-1} \downarrow (p_4, \hat{a}_4, \gamma_4)\), and hence 
\((p_2, \hat{a}_4, \gamma_2), \quad e_4^{-1}, \quad (p_4, \varepsilon, \gamma_4)\) is a chain in \(A(C)\).

(c) \( e_4 = (p_4, (i, i), p_2) \in \text{PUSH}(A(C))\).

In this case \( a_1 = a_2 = 0, \quad \gamma_2 = (i, i) \gamma_2 \), and 
\[ e_4^{-1} = (p_2, (i, i), p_4) \in \text{PUSH}(A(C)) \]
Therefore, \((p_2, \hat{a}_4, \gamma_2) \uparrow e_4^{-1} \downarrow (p_4, \hat{a}_4, \gamma_4)\), so that 
\((p_2, \hat{a}_4, \gamma_2), \quad e_4^{-1}, \quad (p_4, \varepsilon, \gamma_4)\) is a chain in \(A(C)\).

**Induction:** Assume the result holds for chains of length \(< n\).

Now \((p_2, a_2, \gamma_2), e_2, \cdots, e_n, (p_n, a_n, \gamma_n, \gamma_{n+1})\) is a chain of the given form of length \(n-1\), so by the induction hypothesis, there exists a chain:

\[ (1) \cdots (p_n, a_n, \gamma_n, \gamma_{n+1}), e_n^{-1}, \cdots, e_2^{-1}, (p_2, b_2, \gamma_2) \]

where \(b_{n+1} = \hat{a}_2\) and \(b_2 = \varepsilon\).

Consider the chain \((p_4, a_4, \gamma_4), e_4, (p_2, a_2, \gamma_2)\). Now if 
\( e_4 = (p_4, \varepsilon, p_2) \in \text{TRANS}(A(C))\), then \(a_2 = a_4\), \(\gamma_2 = \gamma_4\), and 
\[ e_4^{-1} = (p_2, \varepsilon, p_4)\], so that:

\[ (ii) \cdots (p_2, \varepsilon, \gamma_2) \uparrow e_4^{-1} \downarrow (p_4, \varepsilon, \gamma_4) \]
If we let \( c_j = b_j \neq \forall \) for all \( j \in \{2, \ldots, n+1\} \), we obtain from (i) and (ii), the chain:

\[
(p_{n+1}, c_{n+1}, \gamma_{n+1}), e_{n-1}, \ldots, e_1, (p_1, c_1, \gamma_1)
\]

where \( c_1 = \emptyset \), and \( c_{n+1} = b_{n+1} \neq \mathcal{A}_2 \neq \mathcal{A}_1 \).

Finally, if \( e_1 \in \text{PUSH}(A(C)) \cup \text{POP}(A(C)) \), then \( a_4 = a_2 \) so \((p_4, \emptyset, \gamma_4), e_4, (p_2, \epsilon, \gamma_2)\) is a chain of the given form.

Therefore \((p_2, \epsilon, \gamma_2), e_1, (p_4, \epsilon, \gamma_4)\) is a chain, where \( b_2 = b_4 = \emptyset \). From this and (i) we get a chain:

\[
(p_{n+1}, b_{n+1}, \gamma_{n+1}), e_{n-1}, \ldots, e_1, (p_1, b_1, \gamma_1)
\]

where \( b_4 = \emptyset \), and \( b_{n+1} = \mathcal{A}_2 = \mathcal{A}_1 \).

\[\square\]

**4.2.2.10: Definition:** If \( p \) and \( q \) are two subformulæ of \( C \), then \( p \) is said to be **attached to** \( q \) in \( A(C) \) if and only if for some \( a \in C^* \), \((p, a, \emptyset) \vdash (q, \epsilon, \emptyset)\). We denote this \( p \approx q \mod C \). We also say that \( p \) is attached to \( q \) by **the word** \( a \).

**4.2.2.11: Lemma:** \( \approx \mod C \) is an equivalence relation.

**Proof:** Since \((p, \epsilon, \emptyset) \vdash (p, \emptyset, \emptyset)\) for every subformula \( p \) of \( C \), attachment is reflexive. By 4.2.2.9, attachment is symmetric since \( p \) is attached to \( q \) by \( a \) if and only if \( q \) is attached to \( p \) by \( \mathcal{A} \). Finally, if \( p_1 \approx p_2 \mod C \) and
\[ p_2 \equiv p_3 \mod C \] then \((p_1, a_1, \emptyset) \vdash (p_2, \emptyset, \emptyset)\) and 
\((p_2, a_2, \emptyset) \vdash (p_3, \emptyset, \emptyset)\) for some \(a_1, a_2 \in C^*\). So by 
4.2.2.8(i) and (v), \((p_1, a_1a_2, \emptyset) \vdash (p_3, \emptyset, \emptyset)\). Therefore 
\(p_1 \equiv p_3 \mod C\).

The following lemma establishes the relationship between the output partitions of \text{CLASSIFY}(C) and the
automaton for \(C\), namely, that the equivalence classes under 
\(\equiv \mod C\) are exactly the classes of \(F.\text{out}\). In the
corollaries, we note the resulting uniqueness of the
partitions output by \text{CLASSIFY}, and the exact relationship
between \text{CLASSIFY} and \text{TRANSFORM}.

4.2.2.12: Lemma: If \(F.\text{out}\) is an output partition of
\text{CLASSIFY}(C), then \(p \equiv q \mod F.\text{out}\) if and only if
\(p \equiv q \mod C\).

Proof:
(A) Suppose \(p \equiv q \mod C\), then for some \(a \in C^*\) and \(n \geq 0:\)
\[(p, a, \emptyset) \vdash^n (q, \emptyset, \emptyset)\]
We use induction on \(n\) to show that \(p \equiv q \mod F.\text{out}\).

Basis: If \(n = 0\), then by the definition of \(\vdash^0\), \(p = q\) so
that \(p \equiv q \mod F.\text{out}\).

Induction: Assume the result holds for \(n < k\) and suppose
that \((p, a, \emptyset) \vdash^k (q, \emptyset, \emptyset)\). Then there is a chain
\((p_1, a_1, \gamma_1), e_1, \ldots, e_k, (p_{k+1}, a_{k+1}, \gamma_{k+1})\) in \(A(C)\) such

4.2.2.12
that \((p_1, a_1, \gamma_1) = (p, a, \emptyset)\) and \((p_{k+1}, a_{k+1}, \gamma_{k+1}) = (q, \emptyset, \emptyset)\).

There are two cases to consider:

**Case (a)\( e_4 = (p_1, e_2, p_2) \in \text{TRANS}(A(C))\)**

In this case \(\{p_1, p_2\} \in C\). Consider an execution of \text{CLASSIFY}(C). At \(C_1\), \(S\) initially contains \(\{p_1, p_2\}\) but at the last encounter with \(C_1\), \(S\) is empty, so on some loop from \(C_1\) to \(C_1\), \(\{p_1, p_2\}\) is deleted from \(S\). Either \([p_1] = [p_2]\) at the beginning of this loop, or \([p_1]\) and \([p_2]\) are merged during the loop. In either case, on returning to \(C_1\), \([p_1] = [p_2]\), so that \(p_4 \equiv p_2 \mod F\text{-out}\).

Now \((p_2, a_2, \gamma_2) \mid^{k-1} (p_{k+1}, a_{k+1}, \gamma_{k+1})\)

But \(\gamma_2 = \gamma_4\) since \(e_4 \in \text{TRANS}(A(C))\)

\[\gamma_{k+1} = \emptyset\]

\[a_{k+1} = \emptyset\]

\((p_2, a_2, \emptyset) \mid^{k-1} (p_{k+1}, \emptyset, \emptyset)\)

\[p_2 \equiv p_{k+1} \mod F\text{-out}\]

by the induction hypothesis.

\[p_4 \equiv p_{k+1} \mod F\text{-out}\]

**Case (b)\( e_4 = (p_1, (f, i), p_2) \in \text{PUSH}(A(C))\)**

Let \(m\) be the greatest integer such that \(m \geq 2\), and for all \(i \in \{2, \ldots, m\}\), \(\exists \beta_i \in \mathbb{Z}^*\) such that \(\gamma_i = \beta_i (f, i)\); then \(\gamma_m = (f, i) = \gamma_2\), \(\gamma_{m+1} = \emptyset\) and \(e_m = (p_m, (f, i), p_{m+1}) \in \text{POPF}(A(C))\).
Now \((p_j, a_j, \beta_j(f, i)) \vdash (p_{j+1}, a_j + i, \beta_{j+1}(f, i))\)
for all \(j \in \{2, \ldots, m-1\}\)

So by 4.2.2.8(ii):

\((p_j, a_j, \beta_j) \vdash (p_{j+1}, a_j + i, \beta_{j+1})\)
for all \(j \in \{2, \ldots, m-1\}\)

and \((p_2, a_2, \beta_2) \vdash_{m-1} (p_m, a_m, \beta_m)\)

But \(\beta_2 = \beta_m = 0\), and \(a_2 = ba_m\) for some \(b \in \mathbb{C}^*\). So by 4.2.2.8(i):

\((p_2, b, 0) \vdash_{m-1} (p_m, 0, 0)\)

Since \(m-2 < k\), we can apply the induction hypothesis to obtain:

\((i)\) \(p_2 \equiv p_m \mod F_{\text{out}}\)

Now \(e_1 = (p_4, (f, i), p_2) \in \text{PUSH}(A(C))\)
and \(e_m = (p_m, (f, i), p_{m+1}) \in \text{POP}(A(C))\)

So that:

\((ii)\) \(p_2 = f(q_{21}, \ldots, q_{2r})\)
and \(p_m = f(q_{m1}, \ldots, q_{mr})\)
for some \(q_{21}, \ldots, q_{2r}\)
and \(q_{m1}, \ldots, q_{mr}\)

where \(p_4 = q_{2i}\)

\(p_{m+1} = q_{mi}\)

Since \(p_2\) and \(p_m\) begin with the same function symbol, by (i) and lemma 4.2.1.5, we conclude that \(p_2 \equiv p_m \mod F_{\text{out}}\) and therefore, because of (ii), that:

\((iii)\) \(p_4 \equiv p_{m+1} \mod F_{\text{out}}\)
Finally, \((p_{m+1}, a_{m+1}, \gamma_{m+1}) \vdash_{k=0} (p_{k+1}, a_{k+1}, \gamma_{k+1})\).

But \(\gamma_{m+1} = \emptyset = \gamma_{m+1}\), and \(a_{m+1} = b a_{m+1}\) for some \(b \in C^*\), so by 4.2.2.8(i):

\[(p_{m+1}, b, \emptyset) \vdash_{k=0} (p_{k+1}, \emptyset, \emptyset)\]

\[
\vdash_{k=0} p_{m+1} \equiv p_{k+1} \mod F \cdot \text{out}
\]

From this and (iii), we obtain:

\[p_1 \equiv p_{k+1} \mod F \cdot \text{out}\]

\[\implies p \equiv q \mod F \cdot \text{out}\]

(B) Now suppose that \(p \equiv q \mod F \cdot \text{out}\). Let \(H(F, S)\) assert that:

"if \(p \equiv q \mod F\), or \(\{p, q\} \in S\)

then \(p \equiv q \mod C\)"

Consider any loop from \(C^3\) to \(C^3\) via \(PC6\), and suppose that \(H(F(C^3, 1), S(C^3, 1)) \cup \{\{p_1, p_2\}\}\) holds. We show that \(H(F(C^3, 2), S(C^3, 2)) \cup \{\{p_1, p_2\}\}\) holds. Since neither \(F\) nor \(S\) are changed on the path returning to \(C^3\) via \(PC7\), we need only consider the path returning via PC8. For this path we have:

\[F(C^3, 2) = F(C^3, 1)\]

\[S(C^3, 2) = S(C^3, 1) \cup \{\{q_{11}, q_{21}\}, \ldots, \{q_{1n}, q_{2n}\}\}\]

If \(p \equiv q \mod F(C^3, 2)\), then \(p \equiv q \mod F(C^3, 1)\), so by hypothesis, \(p \equiv q \mod C\). If \(\{p, q\} \in S(C^3, 2)\), either \(\{p, q\} \in S(C^3, 1)\), so again the result holds, by hypothesis; or \(\{p, q\} = \{q_{1i}, q_{2i}\}\) for some \(i\). Suppose without loss of generality, that \(p = q_{1i}\) and \(q = q_{2i}\); then:

(i) \(\ldots \cdot (p, (f, 1), t_e) \in \text{PUSH}(A(C))\)

and \((t_2, (f, 1), q) \in \text{FCP}(A(C))\)
Also \( t_1 \equiv p_1 \mod F(C3,1) \)
and \( p_2 \equiv t_2 \mod F(C3,1) \)

So by the induction hypothesis:

\[ t_1 \equiv p_1 \mod C \]
\[ p_2 \equiv t_2 \mod C \]
and \( p_1 \equiv p_2 \mod C \)

\[ \because t_1 \equiv t_2 \mod C \]

So for some \( a \in C^* \):

\[ (t_1,a,0) \not\succ (t_2,0,0) \]
and therefore, by 4.2.2.8(ii):

\[ (t_1,a,(f,1)) \not\succ (t_2,0,(f,1)) \]

So because of (i):

\[ (p,a,0) \not\succ (q,0,0) \]

\[ \therefore p \equiv q \mod C \]

Therefore \( H(F,S \cup \{p_1,p_2\}) \) is invariant at \( C3 \). We now show that \( H(F,S) \) always holds at \( C1 \). Initially, at \( C1 \) if \( p \equiv q \mod F_{\text{in}} \), then \( p = q \), since each class of \( F_{\text{in}} \) contains one element, so that \( p \equiv q \mod c \). If \( \not\succ \)
\( \{p,q\} \in C \), then \( (p,\not\succ,q) \in \text{TRANS}(A(C)) \) so that \( (p,\not\succ,0) \not\succ (q,0,0) \); that is, \( p \equiv q \mod C \). It remains to show that for each loop from \( C1 \) to \( C1 \), if \( H(F(C1,1),S(C1,1)) \) holds, then \( H(F(C1,2),S(C1,2)) \) holds.

(1) Path from \( C1 \) to \( C1 \) via FC3:

\[ F(C1,2) = F(C1,1) \]
\[ S(C1,2) = S(C1,1) - \{p_1,p_2\} \]
If $p \equiv q \mod F(C1,2)$, then $p \equiv q \mod F(C1,1)$; if
\[\{p,q\} \in S(C1,2),\] then $\{p,q\} \in S(C1,1)$. In either case $p \equiv q \mod C$.

(2) Path from $C1$ to $C1$ via $P(C5)$:

\[F(C3,1) = F(C1,1)\]

\[S(C3,1) = S(C1,1) - \{\{p_4,p_2\}\}\]

Since $H(F(C1,1),S(C1,1))$ holds, $H(F(C3,1),S(C3,1) \cup \{\{p_4,p_2\}\})$ also holds. Suppose $C3$ is encountered $m$ times, then by the invariance of $H(F,S \cup \{\{p_4,p_2\}\})$ at $C3$, $H(F(C3,m),S(C3,m) \cup \{\{p_4,p_2\}\})$ holds.

Now $F(C1,2) = F(C3,m) - \{\{p_4\},[p_2]\} \cup \{[p_4] \cup [p_2]\}$

\[S(C1,2) = S(C3,m)\]

If $\{p,q\} \in S(C1,2)$, then $\{p,q\} \in S(C3,m)$ so by $H(F(C3,m),S(C3,m) \cup \{\{p_4,p_2\}\})$, $p \equiv q \mod C$. If $p \equiv q \mod F(C1,2)$ we have two cases:

either $p \equiv q \mod F(C3,m)$

so by $H(F(C3,m),S(C3,m) \cup \{\{p_4,p_2\}\})$, we have:

$p \equiv q \mod C$.

or $p \equiv p_4 \mod F(C3,m)$

$q \equiv p_2 \mod F(C3,m)$

so by $H(F(C3,m),S(C3,m) \cup \{\{p_4,p_2\}\})$ we have:

$p \equiv p_4 \mod C$

$q \equiv p_2 \mod C$

$p_4 \equiv p_2 \mod C$

$\Rightarrow p \equiv q \mod C$

Hence $H(F,S)$ always holds at $C1$, so in particular:
\[ \text{p} \equiv \text{q mod F.out implies } \text{p} \equiv \text{q mod C} \]

### 4.2.2.13 Corollary:
The output partitions F.out and P.out of CLASSIFY(C) are unique: that is, independent of the choices made during execution. Because of this result, we henceforth refer to F.out and P.out, output by CLASSIFY(C), as F.out(C) and P.out(C).

**Proof:** The uniqueness of F.cut is obvious from lemma 4.2.2.12, and implies the uniqueness of F.out by lemma 4.2.1.5.

### 4.2.2.14: Corollary:
TRANSFORM(C) succeeds, returning partition F.out if and only if CLASSIFY(C) returns partition F.out where each class of F.out contains at most one class of P.out.

**Proof:** By applying corollary 4.2.2.13 to lemma 4.2.1.6.

Note that corollary 4.2.2.14 establishes the uniqueness of the output of TRANSFORM(C) and of the digraph D(C).

### 4.2.2.15: Example:
Consider the set of constraints C of example 4.2.1.2. We denote the constraints in this set by \( \mathcal{C}_1, \ldots, \mathcal{C}_4 \) as follows:

\( \mathcal{C}_1: \{F(x,x), v\} \)

\( \mathcal{C}_2: \{v, F(f(a), h(y))\} \)
\[ \mathcal{E}_3: \{F(u, f(y)), F(h(g(b)), u)\} \]
\[ \mathcal{E}_4: \{h(u), h(f(a))\} \]

The automaton \( A(C) \) for this set is illustrated in figure 4.4. Note from figure 4.3 that:
\[ a \equiv g(b) \mod F.\text{out}(C) \]
so by lemma 4.2.2.12:
\[ a \equiv g(b) \mod C \]

Figure 4.5 shows a chain demonstrating this attachment.

4.2.3: The Unification Graph for \( C \)

Recall that if the transformational stage of the Baxter algorithm succeeds, the resulting partition is used to construct a digraph which must be topologically sorted (section 4.1.2). Similarly, from the output partition \( F.\text{out}(C) \) of CLASSIFY(C), we construct a labelled digraph \( U(C) \).

4.2.3.1: Definition: If \( C \) is a set of constraints, the \( \text{unification graph} U(C) \) for \( C \), is a labelled, directed graph, where:
\[ V(U(C)) = F.\text{out}(C) \]
\[ I(U(C)) = W(C) \quad (\text{definition 4.2.2.1}) \]

and the arc set is defined as follows. Suppose there are \( m \) classes in \( F.\text{out}(C) \), and let \( t_1, \ldots, t_m \) be \( m \) terms such that
\[ \langle t_i \rangle = \langle t_j \rangle \quad \text{if and only if} \quad i = j. \]
Suppose
\[ t_i = f_i(p_{i1}, \ldots, p_{in_i}) \quad \text{for all} \quad i \in \{1, \ldots, m\}, \]
then:

4.2.3.1
The automaton $A(C)$ for the set of constraints $C = \{\epsilon_4, \epsilon_2, \epsilon_3, \epsilon_4\}$ of example 4.2.2.15. An unarrowed line between vertices $p$ and $q$ represents two TRANS arcs from $p$ to $q$ and $q$ to $p$. An arrowed line from $p$ to $q$ represents a PUSH arc from $p$ to $q$ and the corresponding FCQ arc from $q$ to $p$.

Figure 4.4
<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( f_4 f_3 f_2 f_1 f_0 )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>f(a)</td>
<td>( f_4 f_3 f_2 f_1 f_0 )</td>
<td>( (f,1) )</td>
</tr>
<tr>
<td>h(f(a))</td>
<td>( f_4 f_3 f_2 f_1 f_0 )</td>
<td>( (h,1)(f,1) )</td>
</tr>
<tr>
<td>h(u)</td>
<td>( f_3 f_2 f_1 f_0 )</td>
<td>( (h,1)(f,1) )</td>
</tr>
<tr>
<td>u</td>
<td>( f_3 f_2 f_1 f_0 )</td>
<td>( (f,1) )</td>
</tr>
<tr>
<td>F(h(g(b)), u)</td>
<td>( f_3 f_2 f_1 f_0 )</td>
<td>( (F,2)(f,1) )</td>
</tr>
<tr>
<td>F(u, f(y))</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (F,2)(f,1) )</td>
</tr>
<tr>
<td>f(y)</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (f,1) )</td>
</tr>
<tr>
<td>y</td>
<td>( f_2 f_1 f_0 )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>h(y)</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (h,1) )</td>
</tr>
<tr>
<td>F(f(a), h(y))</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (F,2)(h,1) )</td>
</tr>
<tr>
<td>v</td>
<td>( f_4 f_3 f_2 f_1 f_0 )</td>
<td>( (F,2)(h,1) )</td>
</tr>
<tr>
<td>F(x, x)</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (F,2)(h,1) )</td>
</tr>
<tr>
<td>x</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (h,1) )</td>
</tr>
<tr>
<td>F(x, x)</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (F,1)(h,1) )</td>
</tr>
<tr>
<td>v</td>
<td>( f_2 f_1 f_0 )</td>
<td>( (F,1)(h,1) )</td>
</tr>
</tbody>
</table>

This table shows the configurations of a chain of length 24 in the automaton illustrated in figure 4.4. The chain demonstrates that a is attached to \( g(b) \) by \( f_4 f_3 f_2 f_1 f_0 \) and \( C^* \), where \( C \) is the set of constraints of example 4.2.2.15. (Continued on page 144.)

**Figure 4.5**
4.3.2: Lemma: If $C$ is a set of constraints and $U(C)$ contains a closed walk from $[p]$ to $[p]$, then there is a loop on $p$ in $A(C)$.

Proof: Let $[p_1], e_1, \ldots, e_n, [p_{n+1}]$ be a walk in $U(C)$. We show by induction on $n$ that there is a chain in $A(C)$ from $(p_{n+1}, u, \emptyset)$ to $(p_k, e, \gamma)$ for some $u \in C^*$ and $\gamma \neq \emptyset$.

**Basis:** $n=1$.

Suppose $e_1 = ([p_1], f, [p_2])$. Then there exists some term $t \in [p_1]$ such that $t = f(q_1, \ldots, q_k)$, where $k = \text{degree}(f)$ and $q_i \in [p_2]$ for some $i \in \{1, \ldots, k\}$. By lemma 4.2.2.12, $\exists a_1, a_2 \in C^*$ such that:

$$(p_2, a_2, \emptyset) \not\vdash (q_1, \emptyset, \emptyset)$$

and $$(t, a_1, \emptyset) \not\vdash (p_1, \emptyset, \emptyset)$$

But $(q_1, (f, i), t) \in \text{PUSH}(A(C))$.

$$\vdash (q_1, \emptyset, \emptyset) \not\vdash (t, \emptyset, (f, i))$$

Hence by 4.2.2.8(i), (ii) and (v):

$$(p_2, a_2, a_1, \emptyset) \not\vdash (p_1, \emptyset, (f, i))$$

So in this case a chain with the required properties exists.

**Induction:** Assume that the result holds for walks of length $k < n$.

Now $[p_1], e_1, \ldots, e_{n-1}, [p_n]$ is a walk of length $n-1 < n$, and $[p_n], e_n, [p_{n+1}]$ is a walk of length $1 < n$. So by hypothesis, for some $a_1, a_2 \in C^*$ and some $\gamma_1, \gamma_2 \neq \emptyset$

$$(p_{n+1}, a_1, \emptyset) \not\vdash (p_n, \emptyset, \gamma_1)$$

and $$(p_n, a_2, \emptyset) \not\vdash (p_{n+1}, \emptyset, \gamma_2)$$

Therefore, by 4.2.2.8(i), (ii) and (v):
\[(p_{n+1}, a_1 a_2, \gamma_2) \setminus (p_4, \emptyset, \gamma_4 \gamma_2)\]

Again, a chain with the required properties exists.

Consequently, if \(U(C)\) contains a closed walk from \([p]\) to \([p]\), then there is a loop on \(p\) in \(A(C)\).

\[\text{\textbf{4.3.3: Lemma:}}\]

If \((p_1, a_1, \gamma_1), a_1, \ldots, a_n, (p_{n+1}, a, \gamma_{n+1})\) is a chain in \(A(C)\) such that \(\gamma_1 = \emptyset\), and \(\gamma_{n+1} = (f_m, J_m, \ldots, f_k, J_k)\), then there are \(m\) unique states \(q_1, \ldots, q_m\), \(m\) stacks \(a_1, \ldots, a_m\), and \(m\) integers \(i_1, \ldots, i_m\) such that:

1. \(1 < i_1 < i_2 < \cdots < i_m \leq n+1\)
2. \(q_j = p_{i_j} \land a_j = \gamma_{i_j}\) for all \(j \in \{1, \ldots, m\}\)
3. \(a_j\) is a term beginning with \(f_j\), for all \(j \in \{1, \ldots, m\}\)
4. \(|\gamma_k| \geq |a_j|\) for \(k \geq i_j\)
5. if \(m > 1\), \([(q_1), f_j, [(q_1 = \emptyset)] \in E(U(C))\) for all \(j \in \{2, \ldots, m\}\)
6. \([q_m] = [p_{n+1}]\)
7. \([(q_k), f_1, [p_4]] \in E(U(C))\)

\textbf{Proof:} We use induction on \(m\).

\textbf{Basis:} \(m=1\). Let \(r\) be the largest integer such that \(\gamma_r = \emptyset\); let \(i_4 = r+1\), \(q_4 = p_r \circ a\), and \(a_4 = \gamma_r \circ a\), then:

1. \(1 < i_4 \leq n+1\)
2. \(\text{holds trivially}\)
3. Since \(\gamma_r = \emptyset\) and \(\gamma_r \circ a = \emptyset\): \(a_r = (p_r, (f_m, J_m), p_r \circ a) \in \text{PUSH}(A(C))\)
   \(a_r\) is a term beginning with \(f_r\)

4.3.3
\*\* q_\* is a term beginning with f_\*

(4) Since y_i = \emptyset and y_k \neq \emptyset for k > r, we can, for each k \geq r+1, write y_k in the form \beta_k(f_\*, j_\*) for some \beta_k \in \mathbb{Z}^*.

But y_{r+1} = (f_\*, j_\*) = a_\*

\*\* |y_k| \geq |a_\*| for k \geq i_\*

(6)

(p_k, a_k, y_k) \rightarrow_k (p_{k+1}, a_{k+1}, y_{k+1})

for all k \in \{1, \ldots, n\}

But y_k = \beta_k(f_\*, j_\*)

\*\* \*\* (p_k, a_k, \beta_k) \rightarrow_k (p_{k+1}, a_{k+1}, \beta_{k+1})

for all k \in \{r+1, \ldots, n\}

Now \beta_{r+1} = \emptyset, \beta_{n+1} = \emptyset, and a_{r+1} = b a_{n+1} for some b \in C^*

\*\* \*\* (p_{r+1}, a_{r+1}, \emptyset) \rightarrow (p_{n+1}, \emptyset, \emptyset)

So by lemma 4.2.2.12:

[p_{r+1}] = [p_{n+1}]

\*\*\* [q_\*] = [p_{n+1}]

(7)

(p_k, a_k, y_k) \rightarrow_k (p_{k+1}, a_{k+1}, y_{k+1})

for all k \in \{1, \ldots, r-1\}

But y_\* = y_r = \emptyset, and a_\* = b a_\* for some b \in C^*

\*\* \*\* (p_\*, b, \emptyset) \rightarrow (p_\*, \emptyset, \emptyset)

So by lemma 4.2.2.12:

[p_\*] = [p_\*]

Now e_\* = (p_\*, (f_\*, j_\*) \_ p_\* \_ p_\*) \in \text{PUSH}(A(C))

\*\* \*\* p_{r+1} = f_\*(q_\*, \ldots, q_k)

for some q_\*, \ldots, q_k
and \( p_r = q_{1} \)

So \( ([p_{r+1}], f_4, [p_r]) \in E(U(C)) \)

**Induction:** assume the result holds for chains whose final configurations have stacks of length less than \( m \).

Let \( r \) be the greatest integer such that:

\[
\gamma_{r} = (f_{m-1}, j_{m-1}) \cdots (f_4, j_4)
\]

Then \( (p_4, a_4, \gamma_4), e_4, \ldots, e_{r-1}, (p_r, a_r, \gamma_r) \) is a chain such that \( \gamma_4 = \emptyset \) and \( |\gamma_r| < m \). So by the induction hypothesis, states \( q_4, \ldots, q_{m-4} \), stacks \( a_4, \ldots, a_{m-1} \), and integers \( i_4, \ldots, i_{m-1} \) exist, satisfying the conditions.

Let \( i_m = r+1 \), \( q_m = p_{r+1} \) and \( a_m = \gamma_{r+1} \), then:

1. \( 1 < i_4 < \cdots < i_{m-1} < r \) by the induction hypothesis

\[
\therefore 1 < i_4 < \cdots, i_m = r+1 \leq n+1
\]

2. holds trivially

(3) Obviously \( \gamma_{r+1} = (i_m, j_m) \cdots (f_4, j_4) \)

\[
\therefore e_r = (p_r, (f_4, j_4), p_{r+1}) \in \text{PUSH}(A(C))
\]

\[
\therefore p_{r+1} \text{ is a term beginning with } f_4
\]

**Induction:** assume the result holds for chains whose final configurations have stacks of length less than \( m \).

Let \( r \) be the greatest integer such that:

\[
\gamma_{r} = (f_{m-1}, j_{m-1}) \cdots (f_4, j_4)
\]

Then \( (p_4, a_4, \gamma_4), e_4, \ldots, e_{r-1}, (p_r, a_r, \gamma_r) \) is a chain such that \( \gamma_4 = \emptyset \) and \( |\gamma_r| < m \). So by the induction hypothesis, states \( q_4, \ldots, q_{m-4} \), stacks \( a_4, \ldots, a_{m-1} \), and integers \( i_4, \ldots, i_{m-1} \) exist, satisfying the conditions.

Let \( i_m = r+1 \), \( q_m = p_{r+1} \) and \( a_m = \gamma_{r+1} \), then:

1. \( 1 < i_4 < \cdots < i_{m-1} < r \) by the induction hypothesis

\[
\therefore 1 < i_4 < \cdots, i_m = r+1 \leq n+1
\]

2. holds trivially

(3) Obviously \( \gamma_{r+1} = (i_m, j_m) \cdots (f_4, j_4) \)

\[
\therefore e_r = (p_r, (f_4, j_4), p_{r+1}) \in \text{PUSH}(A(C))
\]

\[
\therefore p_{r+1} \text{ is a term beginning with } f_4
\]

(4) We can write \( \gamma_k \) in the form \( \beta_k \gamma_{r+1} \) for each \( k \geq r+1 \)

\[
\therefore |\gamma_k| \geq |\gamma_{r+1}| = |a_m| \text{ for } k \geq m
\]

(5) For \( j \in \{i, \ldots, a-1\} \), by the induction hypothesis:

\[
([q_j], f_1, [q_{j-1}]) \in E(U(C))
\]

Now \( e_r = (p_r, (f_m, j_m), p_{r+1}) \in \text{PUSH}(A(C)) \)

So \( p_{r+1} = f_m(q_{k}, \ldots, q_k) \)

for some \( q_4, \ldots, q_k \)

and \( p_r = q_{j_4} \)
By the induction hypothesis, \([q_{m-1}] = [p_r]\)

\[
([q_m], t_m, [q_{m-1}]) \in E(U(C))
\]

(6) \((p_k, a_k, \gamma_k) \in E_k \ (p_{k+1}, a_{k+1}, \gamma_{k+1})\)

for all \(k \in \{1, \ldots, n\}\)

But \(\gamma_k = \beta_k \gamma_{k+1}\)

for all \(k \in \{r+1, \ldots, n\}\)

Now \(\beta_r = \emptyset, \beta_n = \emptyset,\) and \(a_r = b n+1\) for some \(b \in C^*\)

\[
([p_r], a, a, \emptyset) \in E([p_n], \emptyset, \emptyset)
\]

So by lemma 4.2.2.12:

\([p_r] = [p_n]\)

\([q_m] = [p_n]\)

(7) Holds by the induction hypothesis.

4.3.4: Corollary: If the chain in 4.3.3 is a loop, conditions (5), (6) and (7) can be replaced by:

(5') \(([q_j], t_j, [q_{j-1}]) \in E(U(C))\) for all \(j \in \{2, \ldots, n\}\)

\(([q_m], t_m, [q_m]) \in E(U(C))\)

Proof: If the chain is a loop, \(p_i = p_{n+1}\), so from (6) and (7):

\(([q_i], t_i, [q_m]) \in E(U(C))\)

4.3.5: Corollary: \(U(C)\) has a closed walk if and only if \(A(C)\) has a loop.
Proof: If $U(C)$ has a closed walk, then $A(C)$ has a loop by lemma 4.3.2. If $A(C)$ has a loop, then $U(C)$ has a closed walk by corollary 4.3.4.

4.3.6: Definition: If $A(C)$ has a loop $(p_{i}, a_{k}, \gamma_{i}), e_{1}, \ldots, e_{n}, (p_{n+1}, a_{n+1}, \gamma_{n+1})$, where $|\gamma_{n+1}| = m$, then the $m$ states $q_1, \ldots, q_m$ defined in lemma 4.3.3 and corollary 4.3.4 are called the characteristic states of the loop. For each characteristic state $q_k$, we define a loop on $q_k$ called the $q_k$-canonical form of the original loop, as follows. Suppose $q_k = p_i$, then $|\gamma_i| \geq |\gamma_j|$ for $i \geq j$. Therefore, for each $i \geq j$, we can write $\gamma_i$ in the form $\beta_i \gamma_j$ for some $\beta_i \in \mathbb{Z}^\times$. Also, $a_k = c a_j$ for some $c \in C^\times$. Then the $q_k$-canonical form of the original loop is:

$$(p_i, a_k, c, \beta_j), e_1, \ldots, e_n, (p_{n+1}, c, \beta_{n+1}), e_{l+1}$$

$$(p_{2m}, c, \gamma_2 \beta_{n+1}), e_{2m}, \ldots, e_{l+1}, (p_{l+1}, c, \gamma_j \beta_{n+1}),$$

which is a loop since $\beta_j = \emptyset$ and $\gamma_j \beta_{n+1} \neq \emptyset$ (since $\gamma_{n+1} = \beta_{n+1} \gamma_j$ and $\gamma_{n+1} \neq \emptyset$).

Note that:

$$\{c \mid c \text{ occurs in } a_k\} = \{c \mid c \text{ occurs in } a_j c\}$$

4.3.7: Definition: A chain in $A(C)$:

$$(p_1, a_{k}, \gamma_1), e_1, \ldots, e_n (p_{n+1}, a_{n+1}, \gamma_{n+1})$$

is said to be semi-simple if and only if for all $i$ and $j$ such that $1 \leq i \leq n+1$, either $p_i \neq p_j$ or $\gamma_i \neq \gamma_j$. 

4.3.7
4.3.8: **Definition:** A chain in \( A(C) \):

\[(p_1, a_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})\]

is said to be **simple** if and only if it is semi-simple, and for all \( i \) and \( j \) such that either \( 1 \leq i < j < n+1 \) or \( 1 < i < j \leq n+1 \):

\[
\text{if } p_i = p_j \text{ then } \exists k \in \{i+1, \ldots, j-1\} \text{ such that:} \\
|\gamma_k| < |\gamma_i| \\
\text{and } |\gamma_k| < |\gamma_j|
\]

4.3.9: **Lemma:** Let \( \Delta \) be the set of all simple chains in \( A(C) \) for which the stack of the initial configuration is \( \emptyset \), and the input of the final configuration is \( \emptyset \). Then \( \Delta \) is finite.

**Proof:** Suppose \((p_1, a_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})\) is a simple chain in \( A(C) \), where \( \gamma_1 = \emptyset \) and \( |\gamma_{n+1}| > |\text{PUSH}(A(C))| + 1 \).

Let \( m \) be the greatest integer such that \( \gamma_m \neq \gamma_{n+1} \), then \( |\gamma_m| > |\text{PUSH}(A(C))| \).

Since \( \gamma_1 = \emptyset \), each element of \( \gamma_m \) is added by some arc \( e \) in the chain, where \( e \in \text{PUSH}(A(C)) \). Hence, since \( |\gamma_m| > |\text{PUSH}(A(C))| \), two elements of \( \gamma_m \) are added by the same arc in \( \text{PUSH}(A(C)) \), so therefore there are two integers \( i \) and \( j \) such that \( 1 \leq i < j < n+1 \) and:

(i) \( e_i = e_j \in \text{PUSH}(A(C)) \)

and (ii) for all \( k \in \{i+1, \ldots, n\} \),

\[ \exists \beta_k \in \mathbb{Z}^* \text{ such that } \gamma_k = \beta_k \gamma_i \]
Because of (i), we have $p_i = e$; and from (ii), $|\gamma_k| \geq |\gamma_l|$ for all $k \in \{i+1, \ldots, j-1\}$, contradicting the fact that the chain is simple.

Therefore, the length of the stack in any configuration of a simple chain is bounded by $|\text{PUSH}(A(C))| + 1$. But the stack alphabet is finite so that the set:

$$\mathcal{V}(A(C)) \times \{\gamma \mid |\gamma| \leq |\text{PUSH}(A(C))| + 1\},$$

is finite. Let the size of this set be $m$, then every simple chain has length $\leq m$, since in any longer chain, there are two configurations with the same state and stack.

Now suppose $(p_1, a_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})$ and $(p_1, b_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, b_{n+1}, \gamma_{n+1})$ are two chains in the set $\Delta$. Since $a_{n+1} = b_{n+1} = e$, it is obvious that $a_i = b_i$ for all $i \in \{1, \ldots, n+1\}$. Hence the sequence $(p_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, \gamma_{n+1})$ uniquely defines a chain in the set $\Delta$. But as we have already shown, each such sequence has length $\leq m$, so since the number of state-stack pairs and the number of arcs in $E(A(C))$ is finite, the number of chains in the set $\Delta$ is finite.

\[\]

4.3.10: **Lemma:** If there is a chain of length $n$ in $A(C)$ from $(p,a,\gamma)$ to $(q,b,\alpha)$, where either $p \neq q$ or $\gamma \neq \alpha$, then there is a semi-simple chain of length $\leq n$ in $A(C)$ from $(p,c,\gamma)$ to $(q,b,\alpha)$, for some $c \in C^*$. 

4.3.10
**Proof:** Suppose \((p_1, a_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})\) is not semi-simple, and that either \(p_i \neq p_{n+1}\) or \(\gamma_i \neq \gamma_{n+1}\), then \(p_i = p_j\) and \(\gamma_i = \gamma_j\) for some \(i\) and \(j\) such that \(i < j\) and either \(i \neq j\) or \(j \neq n+1\). Now \(a_i = da_j\) and \(a_j = ba_j\), for some \(d, b \in C^\times\).

\[ (p_1, da_j, \gamma_1) \downarrow^{-1} (p_i, a_j, \gamma_j) = (p_j, a_j, \gamma_j) \]

Also \((p_j, a_j, \gamma_j) \downarrow^{-1} (p_{n+1}, a_{n+1}, \gamma_{n+1})\)

\[ (p_1, da_j, \gamma_1) \downarrow^{-1} (p_{n+1}, a_{n+1}, \gamma_{n+1}) \]

Hence there is a chain of length \(n-(j-i) < n\), from \((p_1, da_j, \gamma_1)\) to \((p_{n+1}, a_{n+1}, \gamma_{n+1})\).

Since the original chain is finite, after a finite number of applications of this process, we must obtain a semi-simple chain satisfying the required conditions.

\[ \square \]

**4.3.11 Lemma:** If \(A(C)\) has a semi-simple chain of length \(n\) that is not simple, then \(A(C)\) has a simple loop of length \(<n>\).

**Proof:** Suppose \((p_1, a_1, \gamma_1), e_1, \ldots, e_n, (p_{n+1}, a_{n+1}, \gamma_{n+1})\) is a chain in \(A(C)\) that is semi-simple but not simple. Then there exist integers \(i\) and \(j\) such that \(i < j\), either \(i \neq j\) or \(j \neq n+1\), \(p_i = p_j\), \(\gamma_i \neq \gamma_j\) and for all \(k \in \{i, \ldots, j\}\), either \(|\gamma_k| \geq |\gamma_i|\) or \(|\gamma_k| \geq |\gamma_j|\).

**case(a):** \(|\gamma_i| < |\gamma_j|\)

In this case, \(|\gamma_k| \geq |\gamma_i|\) for all \(k \in \{i, \ldots, j\}\), so for each \(k \in \{i, \ldots, j\}\), \(\exists \beta_k \in \mathbb{Z}^\times\) such that \(\gamma_k = \beta_k \gamma_i\).

But \((p_k, a_k, \gamma_k) \downarrow e_k^{-1} (p_{k+1}, a_{k+1}, \gamma_{k+1})\)

for all \(k \in \{i, \ldots, j-1\}\).
. \* \cdot (p_k, a_k, \beta_k) \mathbin{\rightarrow} e_k \downarrow (p_k + a, a_k + a, \beta_k + a)

for all \( k \in \{1, \ldots, j-1\} \)

But for all \( k \in \{i, \ldots, j\} \), \( \exists b_k \in C^* \) such that \( a_k = b_k a_j \).

. \* \cdot (p_k, b_k, \beta_k) \mathbin{\rightarrow} e_k \downarrow (p_k + a, b_k + a, \beta_k + a)

for all \( k \in \{i, \ldots, j-1\} \)

Therefore \((p_i, b_i, \beta_i), e_i, \ldots, e_{j-1}, (p_j, b_j, \beta_j)\) is a chain in \( A(C) \) such that \( \beta_i = \emptyset, b_j = \emptyset, p_i = p_j \), and \( \beta_j \neq \emptyset \) since \( \gamma_i \neq \gamma_j \).

The length of this chain is \((j-1)-i+1 = j-i\) which is less than \( n \) since either \( i \neq 1 \) or \( j \neq n+1 \). Hence there is a loop in \( A(C) \) of length \(< n \).

**case(h):** \( |\gamma_j| \leq |\gamma_i| \)

In this case \( |\gamma_k| \geq |\gamma_j| \) for all \( k \in \{i, \ldots, j\} \), and by pursuing an argument similar to that presented in case (a), we obtain a chain in \( A(C) \):

\((p_i, b_i, \beta_i), e_i, \ldots, e_{j-1}, (p_j, b_j, \beta_j)\)

where \( \beta_i = \emptyset, b_j = \emptyset, p_i = p_j, \beta_j \neq \emptyset \).

By lemma 4.2.2.9, therefore, there is a chain in \( A(C) \):

\((p_i, c_i, \beta_i), e_{i-1}^{-1}, \ldots, e_1^{-1}, (p_i, c_i, \beta_i)\)

such that \( c_i = \emptyset \).

The length of this chain is \( j-1 < n \). Therefore there is a loop in \( A(C) \) of length \(< n \).

Since the original chain is finite, after a finite number of applications of this process, we must obtain a simple loop of length \(< n \).
4.3.12: **Definition:** A loop in $A(C)$ is said to be **fundamental** if all its canonical forms are simple.

4.3.13: **Lemma:** If $A(C)$ has a loop, then $A(C)$ has a fundamental loop.

**Proof:** Suppose $A(C)$ has a loop of length $n$ that is not fundamental; then one of its canonical forms, which is also of length $n$, is not simple. So by lemma 4.3.10, $A(C)$ has a semi-simple loop of length $\leq n$, and by lemma 4.3.11, $A(C)$ has a simple loop of length $< n$.

Since the original loop is finite, a finite number of applications of this process must result in a fundamental loop.

4.3.14: **Definition:** If $p$ and $q$ are subformulas of $C$ and there is a simple chain from $(p, a, \emptyset)$ to $(q, a, \emptyset)$ for some $a \in C^*$, then $p$ is said to be **simply attached** to $q$ by $a$ in $A(C)$. It is easy to verify that simple attachment is an equivalence relation.

4.3.15: **Lemma:**

(i) If $p$ is simply attached to $q$ by $a$, and $p \neq q$, then $a \neq \emptyset$.

(ii) If there is a simple loop on $p$ with value $a$, then $a \neq \emptyset$.

**Proof:** Let $(\gamma_1, a_1, \gamma_2), e_1, \ldots, e_n, (\gamma_n, a_n, \gamma_{n+1})$ be a simple chain such that $\gamma_n = \emptyset$, and $a_n = \emptyset$, then:

$$e_1 \in \text{FLUSH}(A(C)) \cup \text{POP}(A(C))$$
for all $i \in \{1, \ldots, n\}$

Now $\gamma_0 = \emptyset$, $e_0 \in \text{PUSH}(A(C))$, so let $k$ be the smallest integer such that $e_k \in \text{FCP}(A(C))$, then:

$$e_{k-1} = (p_{k-1}, (i, i), p_k) \in \text{PUSH}(A(C))$$

and $e_k = (p_k, (i, i), p_{k+1}) \in \text{FCP}(A(C))$

and $\gamma_{k-1} = \gamma_k$

Now $p_k = f(q_1, \ldots, q_m)$

for some $q_1, \ldots, q_m$

and $p_{k-1} = q_j = p_{k+1}$ for some $j \in \{1, \ldots, m\}$

• $p_{k-1} = p_{k+1}$

and $\gamma_{k-1} = \gamma_k$

which contradicts the fact that the chain is simple.

• $e_i \in \text{PUSH}(A(C))$ for all $i \in \{1, \ldots, n\}$

(i) Suppose the chain demonstrates that $p_k$ and $p_{k+1}$ are simply attached by $a_1$, then $\gamma_k = \gamma_{k+1} = \emptyset$. If $a_1 = \emptyset$, then as shown above, $e_i \in \text{PUSH}(A(C))$ for all $i \in \{1, \ldots, n\}$, which implies that $\gamma_{n+1} \neq \emptyset$; a contradiction.

• $a_1 \neq \emptyset$

(ii) Suppose the above chain is a loop, then $p_1 = p_{n+1}$ and $\gamma_0 = \emptyset$. If $a_1 = \emptyset$, then as shown above, $e \in \text{PUSH}(A(C))$ for all $i \in \{1, \ldots, n\}$.

Let $e_i = (p_i, (i, i), p_{i+1})$

for all $i \in \{1, \ldots, n\}$

Then $p_{i+1} = f_i(q_{i1}, \ldots, q_{in_i})$

for some $q_{i1}, \ldots, q_{in_i}$

and for all $i \in \{1, \ldots, n\}$
and \( p_i = q_{i_j} \) for some \( j \in \{1, \ldots, n_i \} \)

- \( \text{ord}(p_i) < \text{ord}(p_i + \epsilon) \) for all \( i \in \{1, \ldots, n\} \)

- \( \text{ord}(p_4) < \text{ord}(p_{n+4}) \)

which contradicts the fact that \( p_4 = p_{n+4} \)

- \( a_4 \neq \emptyset \)

\[ \] 4.3.16: Lemma: If \( C_4 \subseteq C \) and \( a \in C_4^+ \), then:

(i) If \( p \) is simply attached to \( q \) by \( a \) in \( A(C) \), where \( p \neq q \),

then \( p \) is simply attached to \( q \) in \( A(C_4) \).

(ii) If there is a simple loop on \( p \) with value \( a \) in \( A(C) \),

then there is a simple loop on \( p \) with value \( a \) in \( A(C_4) \).

Proof: Suppose \( (p_4, a_4, \gamma_4), e_4, \ldots, e_n, (p_{n+4}, a_{n+4}, \gamma_{n+4}) \) is a simple chain in \( A(C) \) such that \( \gamma_4 = \emptyset \), \( a_{n+4} = \emptyset \) and \( a_4 = a \). We note the following facts:

(a) \( e_i \in \text{TRANS}(A(C)) \) implies \( p_i, p_i + \epsilon \in V(C_4) \)

Proof: Suppose \( e_i = (p_i, f, p_i + \epsilon) \)

Then \( \emptyset \in C_4 \), since \( a_4 \in C_4^+ \)

But \( \{p_i, p_i + \epsilon\} = \emptyset \)

- \( p_i, p_i + \epsilon \in V(A(C_4)) \)

(b) \( e_i \in \text{PUSH}(A(C)) \) implies \( e_i + \epsilon \notin \text{POP}(A(C)) \)

Proof: Suppose \( e_i = (p_i, (f, j), p_i + \epsilon) \in \text{PUSH}(A(C)) \)

and \( e_i + \epsilon = (p_i + \epsilon, (f, j), p_i + 2) \in \text{POP}(A(C)) \)

then \( p_i + \epsilon = f(q_4, \ldots, q_m) \)

for some \( q_4, \ldots, q_m \)

and \( p_i = q_i = p_i + 2 \)

also \( \gamma_i = \gamma_i + 2 \)
contradicting the fact that the chain is simple.

(c) If $p_i \notin V(A(C_4))$ and $e_i \in \text{PUSH}(A(C))$

then $p_{i+1} \notin V(A(C_4))$ and $e_{i+1} \in \text{PUSH}(A(C))$

Proof: Suppose $p_{i+1} \in V(A(C_4))$

Now $e_i \in \text{PUSH}(A(C))$

say $e_i = (p_i,(f,j),p_{i+1})$

= $p_{i+1} = f(q_4,\ldots,q_m)$

for some $q_4,\ldots,q_m$

and $q_i = p_i$

= $p_i \in V(A(C_4))$

which is a contradiction.

Therefore, by (a), since $p_{i+1} \notin V(A(C_4))$:

$e_{i+1} \notin \text{TEANS}(A(C))$

But $e_i \in \text{PUSH}(A(C))$, so by (b):

$e_{i+1} \notin \text{POP}(A(C))$

= $e_{i+1} \in \text{PUSH}(A(C))$

(d) If $p_i \notin V(A(C_4))$ and $e_{i-1} \in \text{POP}(A(C))$

then $p_{i-1} \notin V(A(C_4))$ and $e_{i-2} \in \text{POP}(A(C))$

Proof: Suppose $p_{i-1} \in V(A(C_1))$

Now $e_{i-1} \in \text{POP}(A(C))$

say $e_{i-1} = (p_{i-1},(f,j),p_i)$

= $p_{i-1} = f(q_4,\ldots,q_m)$

for some $q_4,\ldots,q_m$

and $p_i = q_i$

= $p_i \in V(A(C_4))$

which is a contradiction.

Therefore by (a), since $p_{i-1} \notin V(A(C_4))$:
\[ e_{1-2} \notin \text{TRANS}(A(C)) \]

But \( e_{1-4} \notin \text{POP}(A(C)) \), so by (b):
\[ e_{1-2} \notin \text{PUSH}(A(C)) \]
\[ \therefore e_{1-2} \in \text{POP}(A(C)) \]

We now prove the required results.

(1) Now suppose \( p_4 \notin V(A(C_4)) \)

then \( e_4 \notin \text{TRANS}(A(C)) \) by (a)
\[ \therefore e_4 \in \text{PUSH}(A(C)) \]

So by induction, using (c):
\[ e_i \in \text{PUSH}(A(C)) \]
for all \( i \in \{1, \ldots, n+1\} \)

So since \( a_{n+1} = \emptyset, a_{n} = \emptyset \), which is a contradiction.
\[ \therefore p_4 \in V(A(C_4)) \]

(2) Suppose for some \( i \in \{2, \ldots, n\} \), that \( p_i \notin V(A(C_i)) \).

Then by (a), neither \( e_{1-4} \) nor \( e_i \) are in \( \text{TRANS}(A(C)) \).

(A) Suppose \( e_{1-4} \in \text{PUSH}(A(C)) \)

then \( e_i \in \text{PUSH}(A(C)) \) by (b)

Therefore by induction using (c):
\[ e_{n+1} \in \text{PUSH}(A(C)) \]
and \( p_{n+1} \notin V(A(C_{n+1})) \)

In case (i) of the lemma, \( \gamma_{n+1} = \emptyset \), so that:
\[ e_{n+1} \notin \text{PUSH}(A(C)) \]
which is a contradiction.

In case (ii):
\[ p_{n+1} = p_1 \in V(A(C_1)) \]
which is again a contradiction.
(B) Now suppose $e_{i-1} \in \text{PCP}(A(C))$, then by induction using (d), $e_4 \in \text{POP}(A(C))$, which is impossible since $\gamma_4 = \emptyset$.

(3) Finally, for case (i) of the lemma, suppose:

\[ p_{n+1} \notin V(A(C_4)) \]

Then $e_n \notin \text{TRANS}(A(C))$ by (a) and $e_n \notin \text{PLSH}(A(C))$ since $\gamma_{n+1} = \emptyset$.

So by induction using (d), $e_4 \in \text{PCP}(A(C))$, which is impossible since $\gamma_4 = \emptyset$.

Therefore, $p_{n+1} \notin V(A(C_4))$ and for all $i \in \{1, \ldots, n\}$, $p_i \notin V(A(C_4))$ and $e_i \in E(A(C_4))$. So in both (i) and (ii), the chain in $A(C)$ is also in $A(C_4)$.

\[ \square \]

**4.3.17: Lemma:** If $C_4$ and $C_2$ are sets of constraints such that $C_4 \subseteq C_2$, then every chain in $A(C_4)$ is also a chain in $A(C_2)$.

**Proof:** $A(C_4)$ is a subgraph of $A(C_2)$, so the result is obvious.

\[ \square \]

**4.3.18: Definition:** If $C$ is a nonempty set of constraints, a **labelling for $C$** is an ordered pair $[L, \text{LEL}]$, where:

(i) $L$ is a finite set called the **label set**,
(ii) LBL is a function from \( C \) into \( 2^L - \{\emptyset\} \), called the **label function**, such that:

\[
\bigcup_{\varepsilon \in C} LBL(\varepsilon) = L
\]

Note that every set of constraints has a labelling; namely \([C, LBL]\), where \( LBL(\varepsilon) = \{\varepsilon\} \) for all \( \varepsilon \in C \)

**4.3.19: Definition:** If \([L, LBL]\) is a labelling for a set of constraints \( C \), and \( L_4 \subseteq L \), we define a subset \( C[L_4, LBL] \)
called the **restriction of \( C \) to \( L_4 \) under \([L, LBL]\)**, by:

\[
C[L_4, LBL] = \{ \varepsilon \mid \varepsilon \in C \text{ and } LBL(\varepsilon) \cap L_4 \neq \emptyset \}
\]

Since we never have occasion to consider more than one labelling at a time, we abbreviate this to \( C[L_4] \), and say that \( C[L_4] \) is the **restriction of \( C \) to \( L_4 \)**. If \( C_4 \) is a subset of \( C \) such that \( C_4 = C[L_4] \) for some \( L_4 \subseteq L \), we say that \( C_4 \) is a **restriction of \( C \)**. Note that \( C \) is a restriction of itself, since \( C = C[L] \), and that \( L_4 \subseteq L_2 \) implies \( C[L_4] \subseteq C[L_2] \). If \( L_4 \subseteq L \), then the labelling \([L, LBL]\) of \( C \) induces a labelling \([L_4, LBL_4]\) of \( C_4 = C[L_4] \), where \( LBL_4 : C_4 \rightarrow 2^{L_4} - \{\emptyset\} \) is defined by:

\[
LBL_4(\varepsilon) = LBL(\varepsilon) \cap L_4 \text{ for all } \varepsilon \in C_4
\]

**4.3.20: Definition:** If \([L, LBL]\) is a labelling for a set of constraints \( C \), and \( L_4 \subseteq L \), then \( L_4 \) is said to be **correct** if \( C[L_4] \) is unifiable. \( L_4 \) is **maximally correct with respect to \( L \)** if there is no \( L_2 \subseteq L \) such that \( L_4 \neq L_2 \), \( L_4 \subseteq L_2 \) and \( L_2 \) is correct. If \( C \) is unifiable, \( L \) has only one maximally correct subset, namely \( L \) itself.
The relevance of constraint labelings to deduction plans is as follows. If \( G \) is a plan, then its constraint set \( C(G) \) has a labeling \([SCL(G), LBL]\), where:

\[
LBL(\emptyset) = \{ e | e \in SCL(G) \text{ and } \emptyset \not\in C(e) \}
\]

The "label" attached to each constraint in \( C(G) \) is the set of all arcs in \( SCL(G) \) which caused the introduction of that constraint. If the plan \( G \) is not correct, then \( SCL(G) \), considered as the label set in the above labeling, is not correct. So if we can find a maximally correct subset \( E' \) of \( SOL(G) \), then we know that the graph which results when we remove the arcs \( SCL(G)-E' \) from \( G \) is correct, and that we cannot obtain a correct graph by removing any subset of \( SOL(G)-E' \). Our next task is to show how the maximally correct subsets of a label set can be found.

In the following, we assume a familiarity with Boolean algebra, a good description of which may be found in \([9]\)

4.3.21: Definition: If \( L \) is any finite set, denote by \( \mathcal{B}(L) \), the set of all Boolean expression over \( L \) constructed without complementation. If \( B \in \mathcal{B}(L) \), denote by \([B]\) the function from \( 2^L \rightarrow \{0,1\} \) defined by:

\[
[B](L_4) =
\begin{cases} 0 & \text{for all } L_4 \subseteq L \\ 1 & \text{for all } L_4 \subseteq L \\ 0 & \text{iff } l \not\in L_4 \\
\end{cases}
\]

\[
[B_1+B_2](L_4) = [B_1](L_4) + [B_2](L_4)
\]

\[
[B_1 \cdot B_2](L_4) = [B_1](L_4) \cdot [B_2](L_4)
\]

4.3.21
4.3.22: **Definition:** If \( C \) is a set of constraints we define several sets as follows:

(i) If \( p \) and \( q \) are subformulae of \( C \):

\[
\text{ATTACH}(p,q) = \{ a \mid p \text{ is simply attached to } q \text{ by } a \}
\]

(ii) \( \text{CONFLICT} = \{ [p,q] \mid \text{[p]} = \text{[q]} \text{ and } \langle p \rangle \neq \langle q \rangle \} \)

(iii) For any subformula \( p \) of \( C \):

\[
\text{LOOP}(p) = \{ a \mid \exists \text{ a simple loop on } p \text{ with value } a \}
\]

(iv) For any arc \( e \in E(U(C)) \):

\[
\text{TAIL}(e) = \{ t \mid t \text{ begins with } f, \text{ where } e = ([t],f,[p]) \}
\]

(v) \( \text{CIR} = \text{set of all cycles of } U(C) \)

(vi) Let \( B \) be the assertion defined by:

\[
B(\varepsilon) \text{ iff for all } k \in \text{CIR}, \varepsilon \cap E(k) \neq \emptyset
\]

Then let \( \text{COVER} \) be any subset of \( E(U(C)) \) satisfying the condition:

\[
|\text{COVER}| = \min \{ |\varepsilon| \mid \varepsilon \subseteq E(U(C)) \text{ and } B(\varepsilon) \}
\]

Clearly if \( \text{CIR} = \emptyset \), \( \text{COVER} = \emptyset \)

4.3.23: **Definition:** We now define several Boolean expressions over \( L \) as follows:

(i) If \( \varepsilon \in C^+ \), \( B_w(a) = \sum \prod \_l \_occurs \_in \_a \)
(ii) If $p$ and $q$ are distinct subformulae of $C$:

$$B_A(p,q) = \begin{cases} 
1 & \text{if } ATTACH(p,q) = \emptyset \\
\prod_{a \in ATTACH(p,q)} B_w(a) & \text{otherwise}
\end{cases}$$

Note that by lemma 4.3.15 if $a \in ATTACH(p,q)$ then $a \neq \emptyset$, so that $B_w(a)$ is defined. Also, by lemma 4.3.9, $ATTACH(p,q)$ is finite.

(iii) $E_{CON} = \begin{cases} 
1 & \text{if } CONFLICT = \emptyset \\
\prod_{\{p,q\} \in CONFLICT} B_A(p,q) & \text{otherwise}
\end{cases}$

(iv) For any subformula $p$ of $C$:

$$B_L(p) = \begin{cases} 
1 & \text{if } LOOP(p) = \emptyset \\
\prod_{a \in LOOP(p)} B_w(a) & \text{otherwise}
\end{cases}$$

Note that by lemma 4.3.15, if $a \in LOOP(p)$, then $a \neq \emptyset$, so that $B_w(a)$ is defined; and by lemma 4.3.8, $LOOP(p)$ is finite.

(v) For any $e \in E(U(C))$:

$$B_T(e) = \prod_{t \in TAIL(e)} B_L(t)$$

Note that $TAIL(e) \neq \emptyset$
(vi) \[ B_{\text{CYC}} = \begin{cases} 1 & \text{if } \text{COVER} = \emptyset \\ & \cup_{t \in \text{COVER}} B_{\tau}(t) \text{ otherwise} \end{cases} \]

(vii) \[ B_{\text{UNIF}} = B_{\text{CON} \ast B_{\text{CYC}}} \]

In the rest of this section, we consider a set of constraints \(C\) with a labelling \([L, LBL]\).

4.3.24: Lemma: If \(L_a \subseteq L\), and \(\exists C^+\), then:

\[ [B_w(a)](L_a) = 0 \text{ if and only if } a \in (C|L_a)^+ \]

Proof: For all \(\xi \in C\):

\(\xi \in C|L_a \iff LBL(\xi) \cap L_a \neq \emptyset\)

\(\iff \exists l \in LBL(\xi) \cap L_a\)

\(\iff \exists l \in LBL(\xi) \text{ such that } [l](L_1) = 0\)

\(\iff \prod_{l \in LBL(\xi)} [l](L_a) = 0\)

\(\therefore a \in (C|L_a)^+ \iff \xi \in C|L_a \text{ for all } \xi \text{ occurring in } a\)

\[ \prod_{l \in LBL(\xi)} [l](L_a) = 0 \]

for all \(\xi\) occurring in a

\[ [B_w(a)](L_a) = 0 \]

\[ \Box \]
**4.3.25: Lemma:** If $L_{A} \subseteq L$, and $A(C|L_{1})$ either has a loop, or has a chain that is semi-simple but not simple, then 
$[B_{UNIF}](L_{A}) = 0$.

**Proof:** If $A(C|L_{1})$ has a semi-simple chain that is not simple, by lemma 4.3.11, $A(C|L_{1})$ has a loop.

If $A(C|L_{1})$ has a loop, then by lemma 4.3.13, $A(C|L_{1})$ has a fundamental loop. Suppose this loop has value a $e \in (C|L_{1})^{+}$. By corollary 4.3.18, this loop is also in $A(C)$. Let $q_{1}, \ldots, q_{m}$ be the characteristic states of this loop, then by corollary 4.3.4, there is a closed walk $[q_{1}], e_{1}, [q_{2}], \ldots, [q_{m}], e_{m}, [q_{1}]$ in $U(C)$. Either this walk is a cycle, or some subset of its arcs form a cycle. In either case, for some $j \in \{1, \ldots, m\}$, $e_{j} \in COVER$ and $q_{j} \in TAIL(e_{j})$ by lemma 4.3.3 condition (3). Since the loop in $A(C)$ is fundamental, its $q_{j}$-canonical form is simple, so that $B_{W}(b)$ is a factor of the product $B_{UNIF}$ where $b$ is the value of this canonical form. Also:

$\{x \mid x \text{ occurs in } b\} = \{x \mid x \text{ occurs in } a\} \subseteq C|L_{1}$, so that $[B_{W}(b)](L_{A}) = 0$, by lemma 4.3.24

$\therefore [B_{UNIF}](L_{A}) = 0$

**4.3.26: Lemma:** If $L_{A} \subseteq L$, then:

$[B_{UNIF}](L_{A}) = 1$ iff $C|L_{A}$ is unifiable

**Proof:**

(A) Suppose $C|L_{A}$ is not unifiable, then by theorem 4.2.3.3, we have two cases:
case(a): There exist subformulae p and q such that
p ≡ q mod F\cdot out(C|L₄) and p \not\equiv q mod P\cdot out(C|L₄). By
lemma 4.2.2.12, p ≡ q mod C|L₄, so there exists a
chain in A(C|L₄) from (p,a,∅) to (q,∅,∅) for some
a ∈ (C|L₄)*, so by lemma 4.3.10, there is a
semi-simple chain in A(C|L₄) from (p,b,∅) to (q,∅,∅).
Note that b ∈ (C|L₄)*. We have two cases:
(i) Suppose this chain is simple. By lemma 4.3.18,
it is a chain in A(C); and by lemma 4.2.1.5,
since p and q begin with different function
symbols, p \not\equiv q mod P\cdot out(C)
"* [p,q] ∈ CONFLICT
and b ∈ ATTACH(p,q)
Therefore B'(b) is a factor in the product E_{UNIF}^*
But b ∈ (C|L₄)*, so by lemma 4.3.24:
[B'(b)](L₄) = 0
"* [E_{UNIF}](L₄) = 0
(ii) If this chain is not simple, then by lemma
4.3.25:
[E_{UNIF}](L₄) = 0
case(b): U(C|L₄) has a cycle. In this case, by
corollary 4.3.5, A(C|L₄) has a loop, so by lemma
4.3.25:
[E_{UNIF}](L₄) = 0
(B) Now suppose that [E_{UNIF}](L₄) = 0. Then we have two
cases:
case(a): [E_{CYC}](L₄) = 0

4.3.26
In this case, for some subformula $p$ of $C$, there is a simple loop in $A(C)$ on $p$ with value $a \in C^*$, such that $[B_w(a)](L_a) = 0$. Now by lemma 4.3.15, $a \in C^*$, so by lemma 4.3.24, $a \in (C|L_a)^*$, and therefore by lemma 4.3.16, there is a simple loop on $p$ with value $a$ in $A(C|L_a)$. Consequently, $U(C|L_a)$ has a cycle (corollary 4.3.5) so finally, by theorem 4.2.3.3, $C|L_a$ is nonunifiable.

**case (b)** $[B_{CON}](L_a) = 0$

In this case, for some subformulæ $p$ and $q$ of $C$, $p \equiv q \mod F_{\text{out}}(C)$, $p \not\equiv q \mod F_{\text{out}}(C)$, and $p$ is simply attached to $q$ by $a \in C^*$ in $A(C)$, where $[B_w(a)] = 0$. By lemma 4.3.15, $a \in C^*$, so by lemma 4.3.24, $a \in (C|L_a)^*$, and therefore by lemma 4.3.16, $p$ is simply attached to $q$ by $a$ in $A(C|L_a)$. Hence $p \equiv q \mod C|L_a$, so by lemma 4.2.2.12, $p \equiv q \mod F_{\text{out}}(C|L_a)$. But $p$ and $q$ begin with different function symbols, by lemma 4.2.1.5, so by the same lemma $p \not\equiv q \mod F_{\text{out}}(C|L_a)$. Therefore, by theorem 4.2.3.3, $C|L_a$ is nonunifiable.

[]

If $B$ is a Boolean expression constructed without complementation, then there exists [9] a unique (modulo commutativity of Boolean sum and product), sum of products expression $B'$ with the properties:

(a) No product in $B'$ subsumes any other product in $B'$.  

4.3.26
(b) No product in \( E^r \) contains repeated variables.

(c) \( E^r \) defines the same Boolean function as \( E \).

Also, for any two Boolean expressions \( E_1 \) and \( E_2 \),

\[ [E_1] = [E_2] \]

if and only if \( E_1 \) and \( E_2 \) define the same Boolean function.

We are now in a position to prove the main result of this chapter. This theorem allows us to find all the maximally correct subsets of the label set of a set of constraints.

4.3.27: Theorem: If \([L, LBL]\) is a labelling for a set of constraints \( C \), then \( L_1 \subseteq L \) is maximally correct if and only if \( L_1 = L - \{l_1, \ldots, l_n\} \) where \( l_1 \ldots l_n \) is a product in \( E_1^r \).

Proof:

(A) Suppose \( L_1 = L - \{l_1, \ldots, l_n\} \) where \( l_1 \ldots l_n \) is a product in \( E_1^r \).

Then \( [l_1 \ldots l_n](L_1) = \prod_{i=1}^{n} [l_i](L_1) \)  

\[ = 1 \text{ since } l_i \notin L \]  

for all \( i \in \{1, \ldots, n\} \)

\[ \therefore [E_1^r](L_1) = 1 \]

\[ \therefore L_1 \text{ is correct} \]

Now suppose \( L_1 \subseteq L_2 \), and \( L_2 \) is correct, then there exists a product \( m_1 \ldots m_k \) in \( E_1^r \) such that:

\[ [m_1 \ldots m_k](L_2) = 1 \]

\[ \therefore m_i \notin L_2 \text{ for all } i \in \{1, \ldots, k\} \]
\[
\begin{align*}
\text{••• } m_i & \notin L_d \text{ for all } i \in \{1, \ldots, k\} \\
\text{••• } \{m_1, \ldots, m_k\} \subseteq \{l_1, \ldots, l_n\}
\end{align*}
\]

If these sets are not equal, then the product \(m_1 \cdots m_k\) subsumes the product \(l_1 \cdots l_n\), which is impossible. Therefore \(\{m_1, \ldots, m_k\} = \{l_1, \ldots, l_n\}\), so that \(L_d = L_2\).

\text{••• } L_d \text{ is maximally correct}

(B) Suppose \(L_d \subseteq L \text{ is maximally correct.}\)

Let \(L_d = L - \{l_1, \ldots, l_n\}\). Now \([B^d_{UNIF}(L_d)](L_d) = 1\), since \(L_d\) is correct. Therefore there exists a product \(m_1 \cdots m_k\) in \(B^d_{UNIF}\) such that:

\[
\prod_{i=1}^{k} [m_i](L_d) = 1
\]

\text{••• } m_i \notin L_d \text{ for all } i \in \{1, \ldots, k\} \\
\text{••• } \{m_1, \ldots, m_k\} \subseteq \{l_1, \ldots, l_n\}

Let \(L_2 = L - \{m_1, \ldots, m_k\}\). Then \(L_2\) is correct by (A), and \(L_d \subseteq L_2\). But \(L_d\) is maximally correct, so that \(L_2 = l_1\).

\text{••• } l_1 \cdots l_n \text{ is a product in } B^d_{UNIF}

\[\square\]

**4.3.28: Example:** Consider the set of constraints \(C\) as follows:

\(C_1: \{G(s, z), G(v, F(y, y))\}\)
\(C_2: \{u, F(y, G(s, z))\}\)
\(C_3: \{u, F(H(w), G(x, r))\}\)
\(C_4: \{F(H(H(u)), H(u)), F(H(H(v)), v)\}\)
\(C_5: \{v, F(y, y)\}\)
The unification graph for the constraint set of example 4.3.28. By the definition of U(C), each vertex is a class of F.out(C). The sets within each vertex are the classes of F.out(C). The arcs are labelled with function symbols according to the definition of U(C); also, integer labels 1,...,7 are attached in order that we can refer to the arcs as e₁,...,etc.

Figure 4.6
The automaton for the constraint set of example 4.3.28

Figure 4.7
The unification graph U(C) for this set of constraints is shown in figure 4.6. The classes of the partition P_{out}(C) are specified within the nodes of U(C).

The set of pairs of incompatible terms is:

\[
\text{CONFLICT} = \{ [F(H(w), G(x, r)), H(u)], \\
               [F(H(w), G(x, r)), H(v)], \\
               [F(y, G(s, z)), H(u)], \\
               [F(y, G(s, z)), H(v)], \\
               [F(y, y), H(u)], \\
               [F(y, y), H(v)], \\
               [C(s, z), H(w)], \\
               [G(v, F(y, y)), H(w)], \\
               [G(x, r), H(w)] \}
\]

The set of cycles of U(C) is:

\[
\text{CIR} = \{ ([u], e_4, [y], e_2, [u]), ([u], e_7, [u]) \}
\]

Of the two possible "coverings" for CIR, we choose:

\[
\text{COVER} = \{ e_4, e_7 \}
\]

So the sets of states of A(C) which must be investigated for loops is:

\[
\text{TAIL}(e_4) = \{ F(y, y), F(H(w), G(x, r)), F(y, G(s, z)) \}
\]
\[
\text{TAIL}(e_7) = \{ H(u), H(v) \}
\]

By investigating the automaton A(C) (figure 4.7) we obtain the following:

\[
\text{ATTACH}(F(H(w), G(x, r)), H(u)) = \{ s_3 e_4 e_4 \}
\]
\[
\text{ATTACH}(F(H(w), G(x, r)), H(v)) = \{ s_3 e_4 e_4 e_4 \}
\]
\[
\text{ATTACH}(F(y, G(s, z)), H(u)) = \{ s_2 e_4 e_4 \}
\]
\[
\text{ATTACH}(F(y, G(s, z)), H(v)) = \{ s_2 e_4 e_4 e_4 \}
\]
\[
\text{ATTACH}(F(y, y), H(u)) = \{\varepsilon_4\}
\]
\[
\text{ATTACH}(F(y, y), H(v)) = \{\varepsilon_4 \varepsilon_4\}
\]
\[
\text{ATTACH}(G(s, z), H(w)) = \{\varepsilon_2 \varepsilon_4 \varepsilon_5 \varepsilon_5 \varepsilon_4 \varepsilon_3, \varepsilon_2 \varepsilon_4 \varepsilon_5 \varepsilon_2 \varepsilon_3\}
\]
\[
\text{ATTACH}(G(v, F(y, y)), H(w)) = \{\varepsilon_3 \varepsilon_2 \varepsilon_5 \varepsilon_4 \varepsilon_2 \varepsilon_1, \varepsilon_3 \varepsilon_4 \varepsilon_5 \varepsilon_4 \varepsilon_2 \varepsilon_1\}
\]
\[
\text{ATTACH}(G(x, r), H(w)) = \{\varepsilon_3 \varepsilon_4 \varepsilon_5 \varepsilon_5 \varepsilon_4 \varepsilon_3, \varepsilon_3 \varepsilon_2 \varepsilon_5 \varepsilon_4 \varepsilon_3\}
\]

\[
\text{LCCP}(F(y, y)) = \{\varepsilon_1 \varepsilon_2 \varepsilon_4 \varepsilon_5\}
\]
\[
\text{LOOP}(F(H(w), G(x, r))) = \emptyset
\]
\[
\text{LOOP}(F(y, G(s, z))) = \{\varepsilon_2 \varepsilon_4 \varepsilon_5 \varepsilon_4\}
\]
\[
\text{LCCP}(H(u)) = \{\varepsilon_5 \varepsilon_1 \varepsilon_2, \varepsilon_1 \varepsilon_2\}
\]
\[
\text{LCCP}(H(v)) = \{\varepsilon_4\}
\]

Now consider the labelling \([C, LBL]\) for \(C\), where for all \(\varepsilon \in C\), \(LBL(\varepsilon) = \{\varepsilon\}\). Note that this labelling is not the one we will use when deciding how to prune a plan. In that case, we will use the labelling defined following 4.3.20. In this example, the set of constraints we are processing does not originate from a plan, so we use the simplest possible labelling for illustrative purposes.

We obtain the Boolean sum of products over \(C\):

\[
\text{BUNIF} = \varepsilon_4 \varepsilon_1 + \varepsilon_4 \varepsilon_2
\]

Therefore the label set \(C\) has two maximally correct subsets:

\[
\{\varepsilon_2, \varepsilon_3, \varepsilon_5\}
\]

and \(\{\varepsilon_4, \varepsilon_3, \varepsilon_5\}\)

which are the maximal unifiable subsets of \(C\).
CHAPTER 5

Illustrations and Conclusions

In this chapter we give examples to demonstrate the features of deduction plans, and compare plans to other deduction systems.

§ 1: Backtracking

We now informally describe the operation of a constraint processing system based on the results of chapter 4. As we remarked following 4.3.20, the labelling used is [SOL(G), LBL], where for all \( x \in C(G) \):

\[
LBL(x) = \{ e | x \in C(e) \}
\]

The following description has an inductive structure to parallel the construction of plans.

**Basis:** If \( G \) is a basic plan, \( C(G) = \emptyset \), so the input set \( S \) to CLASSIFY is empty, and the automaton for \( C(G) \) is a null graph.

**Induction:** Suppose that a correct plan \( G \) has been constructed and that its constraint set \( C(G) \) has been processed by CLASSIFY, producing the partitions \( P.out(C(G)) \) and \( P.out(C(G)) \). The input set of constraints \( S \) to CLASSIFY is currently empty, and the theorem-prover is attempting to
close an open subproblem u of G by adding a new arc e to
SCL(G), producing a plan G'. Then:

(a) (i) The new constraints C(e) are added to S.

(ii) If p is a new subformula introduced by these new
constraints, then [p] is added to F.out(C(G)), and
if p is a term [p] is added to F.out(C(G)).

(iii) The automaton for C(G') is constructed from
A(C(G)) by adding a new vertex for every new
subformula, and adding the corresponding new PUSH
and FOP arcs. If \( p = \{p_1, p_2\} \in C(e) \cap C(G) \), two new
arcs \((p_1, \epsilon, p_2)\) and \((p_2, \epsilon, p_1)\) are added to
TRANS(A(C(G))), and the label for \( \epsilon \) is defined by:

\[
\text{LAB}(\epsilon) \leftarrow \{e\}
\]

If \( \epsilon \in C(e) \cap C(G) \), \text{LAB}(\epsilon) is updated as follows:

\[
\text{LAB}(\epsilon) \leftarrow \text{LAB}(\epsilon) \cup \{e\}
\]

(b) Execution of CLASSIFY is resumed at point \( C_4 \) of the
flowchart and the new partitions F.out(C(G')) and
P.out(C(G')) are produced. If any class of F.out(C(G'))
is found to contain more than one class of P.out(C(G')),
this class is marked, and the consequent nonunifiability
noted.

(c) The unification graph U(C(G')) is constructed, and its
cycles (if any) are enumerated by one of the well-known
algorithms [40, 17, 34].

(d) If C(G') is found to be nonunifiable during step (b),
the marked classes of F.out(C(G')) are investigated to
find all pairs \( \{p,q\} \) of incompatible terms. \( B_{CON} \) is then calculated.

(e) If \( C(G') \) is found to be nonunifiable during step (c), a set \( \text{COVER} \) of arcs of \( U(C(G')) \) is found which contains an arc of every cycle. \( B_{CYC} \) is calculated.

(f) \( B_{UNIF} \) is determined and simplified to \( B_{UNIF'} \).

Once this information has been supplied by the constraint processing system, it is the task of the theorem-prover to decide how to use it. Removing the arcs from \( G' \) which comprise one of the products of \( B_{UNIF} \) results in a subgraph with a unifiable constraint set. This subgraph will not in general be a subplan, since if one of the arcs removed is in \( \text{REPL}(G') \), \( (v,w) \) say, then the subgraph obtained still contains the arcs into and out of the direct descendants of \( v \). Furthermore, if the subgraph is further pruned in order to obtain a subplan, it is possible that some subset of the arcs removed in this pruning actually constitutes another product of \( B_{UNIF} \), in which case, we end up with a subplan of a larger correct subplan. Consequently, we define a new Boolean expression \( E_{MAX} \) over \( \text{SOL}(G') \) as follows. If \( e' = (v,w) \in \text{REPL}(G') \), let \( \{e_1, \ldots, e_n\} \) be the set of all arcs which either close a direct descendant of \( v \), or factor some subproblem to a direct descendant of \( v \). Denote by \( S' \), the Boolean product \( e' \cdot e_1 \cdot \ldots \cdot e_n \). Now we replace every \( e'' \) in \( B_{UNIF} \), if it is in
REEPL(G'), by $s^{n}$; the resulting sum of products is $B_{\text{MAX}}$. Clearly, if we now remove from $G'$ all the arcs in some product of $B_{\text{MAX}}$, we obtain a correct subplan of $G'$ which is maximal in the sense that it is not a subplan of any larger correct subplan. Note that since $C(G)$ is unifiable and $C(G')$ is not, $G$ is one of the maximal correct subplans of $G'$; hence one of the products of $B_{\text{MAX}}$ will consist of the single arc $e$, with which the open subproblem $u$ of $G$ was closed to obtain $G'$. If there are no untried alternative solutions to $u$, this product should be ignored.

We illustrate the backtracking process described above, with the following example.

5.1.1: Example: Let $\#$ be the set of clauses:

\[
\{ \{Q(x,y), P(x), P(a)\}, \\
\{\neg P(x)\}, \\
\{\neg Q(x,y), S(y), R(x,y)\}, \\
\{\neg R(x,x), \neg Q(y,z), P(x)\}, \\
\{\neg S(x), T(x)\}, \\
\{\neg T(x), M(x), M(y)\}, \\
\{\neg M(b)\} \}
\]

where $a$ and $b$ are constants. Figure 5.1 illustrates a closed plan $G$ for $\#$, in which the arcs in $\text{SOL}(G)$ are labelled with integers indicating the order in which they were constructed. The constraint set $C(G)$ for this plan is shown in figure 5.2, and the partition $F_{\text{out}}(C(G))$ is given in figure 5.3.
A closed plan $G$ for the set of clauses of example 5.1.1. The integer labels on the arcs of SOL($G$) indicate the order of construction of $G$.

Figure 5.1
\[ z_1: \{P(a), P(x_4)\} \]
\[ z_2: \{P(x_4), P(x_2)\} \]
\[ z_3: \{Q(x_6, y_1), Q(x_3, y_3)\} \]
\[ z_4: \{R(x_3, y_3), R(x_4, x_4)\} \]
\[ z_5: \{P(x_4, P(x_4)\} \]
\[ z_6: \{-Q(y_4, x_4), -Q(x_1, y_1)\} \]
\[ z_7: \{S(y_3), S(x_5)\} \]
\[ z_8: \{T(x_5), T(x_6)\} \]
\[ z_9: \{M(y_6), M(x_6)\} \]
\[ z_{10}: \{M(x_6), M(b)\} \]

The constraint set \( C(G) \) for the plan \( G \) of figure 5.1

**Figure 5.2**

G is not correct since \( a \equiv b \mod F.out(C(G)) \). Note that \( LBL(z_i) = i \) for all \( z_i \in C(G) \). By applying the procedure described above, we obtain:

\[ B^*_{UNIF} = 1 + 4 + 7 + 8 + 10 + 3.5 \]
\[ \therefore B_{MAX} = 1 + 4 + 7 + 8 + 10 + 3.5 \]

where \( 4 = 4.5.6 \)

\[ 7 = 7.8.9.10 \]
\[ 8 = 8.9.10 \]
\[ 9 = 3.4.5.6.7.8.9.10 \]
\[ \therefore B^*_{MAX} = 1 + 4.5.6 + 10 \]
\{ P(a), P(x_1), P(x_2), P(x_4) \}
\{ C(x_1,y_1), Q(x_3,y_3), Q(y_4,z_4) \}
\{ -Q(y_4,z_4), -Q(x_1,y_1) \}
\{ R(x_3,y_3), R(x_4,x_4) \}
\{ S(y_3), S(x_5) \}
\{ T(x_5), T(x_6) \}
\{ W(y_6), W(x_6), W(b) \}
\{ x_1, x_2, x_3, x_4, x_5, x_6, y_1, y_3, y_4, y_6, z_4, a, b \}

The partition \text{P.out}(C(G)) for the constraint set \text{C(G)} of figure 5.2

\textbf{Figure 5.3}

Since \text{W}(x_6) has no other solution than that represented by the arc 10, we will not backtrack by removing this arc. This leaves two choices: remove arcs 4, 5 and 6, or remove arc 1. If the strategy employed is to remove as little as possible, we would remove arc 1. There is then only one choice for closing \text{P}(a); that is, by replacement using the clause \{-P(x)\}.

As the reader has probably noticed, backtracking to one of the maximal correct subplans could result in the system eventually generating a graph which is not a plan, since not all subplans are plans. This will not cause unsoundness, however, in view of lemma 3.3.2.
In example 5.1.1, we used the criterion "remove as little as possible" to decide how to prune the plan: there are undoubtedly many ways to make this choice. We will not investigate here the problem of choosing between the maximal correct subplans in backtracking, but suggest this as a worthwhile topic for further research.

The utility of the system described in this section, obviously depends quite heavily on the existence of a good algorithm for enumerating simple chains in the automaton between given configurations. The existence of an algorithm is evident from the finiteness result of lemma 4.3.9. Developing a practical algorithm would involve a detailed discussion of data structures, which is beyond the scope of this thesis.

Similarly, we omit the investigation of another topic of practical importance: namely, how to salvage as much information as possible after a plan is pruned. We can of course, completely reprocess the remaining constraints to obtain the new partitions \( F \) and \( P \), then build the corresponding unification graph; however, depending on the data structure, it may be possible to reduce this reprocessing in some way.

Again, we suggest both the above as areas worthy of further research.
5.2: Deduction plans and linear deduction

To each linear deduction rule there corresponds a rule for plan construction; however, one of our rules, backfactoring, has no equivalent in existing deduction systems. Backfactoring requires that a record is kept of subproblems that have been solved. The linear systems which have a reduction rule are the only ones which keep a record of some solved subproblems, but those which are kept are ancestors of the rightmost literal of a chain and so cannot be used in factoring. Hence any factoring in a linear deduction system is simple.

An interesting property of the factoring rule for plan construction is that in any complete subset of the rules which contains factoring, completeness is preserved regardless of which factoring rule we use; so we can actually limit ourselves to factoring only to subproblems which have been closed. This suggests strategies for choosing clauses for use in replacement according to what closed subproblems are available for backfactoring.

Of the three minimal complete subsets of the rules, \{(1)A, (2)\} corresponds to the ME-deduction system of Loveland [24, 25, 27], and the SL-resolution system of Kowalski and Kuehner [23]. The set \{(1), (3)A\}, although similar to the original simple linear deduction system of Loveland [26], Luckham [28], and Zamol and Sharonov [44], is actually more powerful in that more lemmas are available for use in ancestor replacement (see below).
The backtracking behaviour of plans is clearly superior to that of existing deduction systems. This is illustrated by the following example.

5.2.1: **Example:** Consider the set of clauses \( S \) of example 5.1.1. We omit the set braces and number the clauses thus:

1. \( Q(x,y), P(x), P(a) \)
2. \(-P(x)\)
3. \(-Q(x,y), S(y), R(x,y)\)
4. \(-R(x,x), -C(y,z), P(x)\)
5. \(-S(x), T(x)\)
6. \(-T(x), M(x), M(y)\)
7. \(-M(b)\)

Recall that \( a \) and \( b \) are constants. In generating the plan \( G \) of figure 5.1, the order of closure of subproblems is right to left, the order of application of the rules is factoring, reduction, replacement, and the clauses for (simple) replacement are chosen in the above order. We now present a deduction from \( S \) using model elimination with factoring, with the same ordering of subproblems, a similar ordering of the rules (contraction, factoring, reduction, extension) and the same order of selection of input clauses for extension. In the following search, \( A \)-literals are framed, and the rules applied are recorded to the right in abbreviated form: for example "ext(1)" means extension using clause (1).

1. \( Q(x,y), P(x), P(a) \) \hspace{1cm} \text{top}
2. \( Q(a,y), P(a) \) \hspace{1cm} \text{fact}
(9) \(Q(a,y), P(a)\)
(10) \(Q(a,y)\)
(11) \(Q(a,y), S(y), R(a,y)\)
(12) \(Q(a,a), S(a), R(a,a), \neg Q(w,z), P(a)\)
(13) \(Q(a,a), S(a), R(a,a), \neg Q(w,z), P(a)\)
(14) \(Q(a,a), S(a), R(a,a), \neg Q(w,z)\)
(15) \(Q(a,a), S(a), R(a,a)\)
(16) \(Q(a,a), S(a)\)
(17) \(Q(a,a), S(a), T(a)\)
(18) \(Q(a,a), S(a), T(a), M(a), M(y)\)
(19) \(Q(a,a), S(a), T(a), M(a)\)

backtrack to (18)

(20) \(Q(a,a), S(a), T(a), M(a), H(b)\)
(21) \(Q(a,a), S(a), T(a), M(a)\)

backtrack to (14)

(22) \(Q(a,a), S(a), R(a,a), Q(w,z), P(w), P(a)\)
(23) \(Q(a,a), S(a), R(a,a), \neg Q(a,z), P(a)\)
(24) \(Q(a,a), S(a), R(a,a), \neg Q(a,z), P(a)\)
(25) \(Q(a,a), S(a), R(a,a), \neg Q(a,z)\)
(26) = (15)
(27) = (16)
(28) = (17)
(29) = (18)
(30) = (19)

backtrack to (28)

(31) = (20)
(32) = (21)
backtrack to (22)

(33) \[ Q(a,a), S(a), [R(a,a)], [-Q(w,z)], P(w), \overline{P(a)} \] ext(2)

(34) \[ Q(a,a), S(a), [R(a,a)], [-Q(w,z)], P(w) \] cont

(35) \[ Q(a,a), S(a), [R(a,a)], [-Q(w,z)], P(w) \] ext(2)

(36) \[ Q(a,a), S(a), [R(a,a)], [-Q(w,z)] \] cont

(37) = (15) cont

(38) = (16) cont

(39) = (17) cont

(40) = (18) ext(5) ext(6)

(41) = (19) fact

backtrack to (40)

(42) = (20) ext(7)

(43) = (21) cont

backtrack to (1)

(44) \[ Q(x,y), F(x), \overline{P(a)} \] ext(2)

(45) \[ Q(x,y), F(x) \] cont

(46) \[ Q(x,y), \overline{P(x)} \] ext(2)

(47) \[ Q(x,y) \] cont

(48) \[ Q(x,y), S(y), R(x,y) \] ext(3)

(49) \[ Q(x,y), S(x), \overline{R(x,y)}, -Q(w,z), P(x) \] ext(4)

(50) \[ Q(x,y), S(x), \overline{R(x,y)}, -Q(w,z), \overline{P(x)} \] ext(2)

(51) \[ Q(x,y), S(x), \overline{R(x,y)}, -Q(w,z) \] cont

(52) \[ Q(x,y), S(x), \overline{R(x,y)} \] red

(53) \[ Q(x,y), S(x) \] cont

(54) \[ Q(x,y), [S(x)], T(x) \] ext(5)

(55) \[ Q(x,y), [S(x)], T(x), M(x), M(y) \] ext(6)

(56) \[ Q(x,y), [S(x)], T(x), M(x) \] red
Six backtrackings are performed before the source of nonunifiability is discovered; compare this with the backtracking performed in the construction of the corresponding plan, in example 5.1.1. Note also that when the correct cutting point is finally discovered at clause (43), in the above deduction, all previously found subproofs are lost even though they are correct, and are reproduced in clauses (45) to (56). In fact, between the various backtrackings, parts of the proof are generated several times: for instance, the subproblem $M(x)$ corresponding to the third literal of clause (6) in $\gamma$, is closed seven times.

Most linear deduction systems allow the use of lemmas: that is, any clause which has been deduced in the course of the current proof may be used as an input clause, as though it belongs to the set $\Gamma$, whose unsatisfiability the system is trying to establish. The linear structure of these systems, however, precludes the use of many lemmas which are available in the construction of plans.
5.2.2: Example: Let $\mathcal{S}$ be the set of clauses:

$$
\{ [-P(x), Q(x)],
[Q(y), P(f(y))],
[P(z), -Q(h(z))],
[-P(f(f(b))],
{Q(h(h(h)))} \}
$$

where $b$ is a constant. Figure 5.4 illustrates a closed, correct plan for $\mathcal{S}$ the construction of which requires two ancestor replacements using variants $\mathcal{C}_1$ and $\mathcal{C}_2$ of clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ deduced by subplans as indicated in the diagram. In a linear system, once $-P(x)$ in the top clause is closed, it is no longer available for use in a lemma; similarly for $Q(x)$. One of these subproblems has to be solved first, however, so that only one of the two lemmas $\mathcal{C}_1$ and $\mathcal{C}_2$ used in generating the plan is available.

If a deduction system is to have access to the variety of lemmas which are available in plan construction, each literal used in a proof in that deduction system must be represented at least once. In a plan, each literal is represented exactly once, so among systems which use ancestor replacement, ours attains the best possible economy of representation.

There have been other attempts at representational economy in theorem-proving programs. Boyer and Moore in
A plan for the set of clauses of example 5.2.2.

Figure 5.4
[8], suggested a method for representing resolvents of clauses by a system of pointers to parent clauses, and to resolved literals. In their system, as in ours, each literal is represented only once: theirs, however, is strictly a method of representation, and solves none of the problems associated with efficient backtracking, use of lemmas, ordering of subgoals, etc. Although clauses are not explicitly created, they exist implicitly; also, substitutions are performed implicitly. Therefore, in order to perform a resolution, it is necessary to search recursively through the structure to carry out the unification and implicit construction of the resolvent.

The use of unification is also more economical in plans than in other deduction systems, since the unification algorithm is used only to verify the applicability of the rules: whenever a plan is closed, we have a refutation provided that the constraint set is unifiable. Substitutions are therefore never performed, and mgus are not calculated. In this regard, our system is similar to Buet's higher-order constrained resolution system [16].

A major difficulty with using problem-reduction in predicate calculus is that the subproblems are usually not independent. In solving a particular subproblem, we may destroy our chances of finding a solution to another subproblem. To take advantage of the problem-reduction method, therefore, we must process the subproblems in
Shostak [38] builds "clause graphs" in which the vertices are clauses and each vertex is divided into "cells", one for each literal in the clause. The edges are undirected, each edge connecting two sets of cells in the graph such that two cells at opposite ends of an edge are literals with opposite sign and the same atom. Clause graphs are not true graphs since edges do not connect vertices. In order to refute a set of clauses $\mathcal{F}$, it is necessary to build a clause graph for $\mathcal{F}$ having no "loops", where a loop in a clause graph is approximately equivalent to a cycle in an undirected graph. If such a graph can be built it is called a \textit{reutation graph} for the set of clauses.

The \textit{resolution graphs} of Yates, Raphael and Bart [43] are quite similar to Shostak's graphs.

In [39] Sickel describes \textit{clause interconnectivity graphs} which are identical to Kowalski's \textit{connection graphs} [22]. The vertices of these graphs are the literals of the clauses in the set $\mathcal{F}$ of clauses being considered, and there is an undirected edge connecting each pair of literals with opposite sign and unifiable atoms. Each edge is labelled with the appropriate mgu. To extract a refutation from such a graph, Sickel marks all the nodes corresponding to some clause, then walks each marker through the graph, checking the consistency of the substitutions being accumulated on these walks. Traversing an edge $(u,v)$ with a marker
corresponds to resolving away the literal $u$, so the marker
is then removed for the literal $v$ and copies of it are
placed on all other literals in the clause containing $v$. If
all markers can be completely eliminated by this process,
then the set of clauses is unsatisfiable. In Kowalski's
system, an edge in the connection graph is selected, and the
clause obtained by resolving the connected literals is added
to the graph together with the appropriate new edges. The
edge which generated the resolvent is then deleted. If a
vertex has no edges attached to it, the clause in which it
occurs and all associated edges are deleted. Also, if a
clause is a tautology, it is deleted together with all the
associated edges. The set of clauses used to construct the
original graph is refuted when a null graph is obtained.
REFERENCES

[1] Aho A.V. and Ullman J.D.
The Theory of Parsing, Translation, and Compiling.
Volume I: Parsing
Prentice-Hall (1972).

Interpr{e}teur de langage de programmation P{E}LOG

An Efficient Unification Algorithm
Research Report CS-73-29, Department of Computer Science, University of Waterloo (1973).

A Practically Linear Unification Algorithm
Research Report CS-76-13, Department of Computer Science, University of Waterloo (1976).

The Complexity of Unification

Application of Mechanical Theorem Proving to Symbolic Calculus

Graph Theory with Applications

The sharing of structure in theorem-proving programs

[9] Brzozowski J.A. and Yoeli N.
Digital Networks

[10] Chang C. and Lee R.C.
Symbolic Logic and Mechanical Theorem Proving
Un système de communication homme-machine en français
Groupe d'Intelligence Artificielle, U.E.R. Luminy,
Marseilles (1972).

[12] Cox P.T. and Pietrzykowski T.
A Graphical Reduction System
Research Report CS-76-35, Department of Computer
Science, University of Waterloo (1976).

[13] van Emden M.H.
Programming with resolution logic
Research Report CS-75-30, Department of Computer
Science, University of Waterloo (1975).

[14] van Emden M.H. and Kowalski R.A.
The Semantics of Predicate Logic as a Programming
Language
J. ACM, v.23 no.4 (October 1976).

PLANNER: A language for proving theorems in robots

[16] Huei G.P.
Constrained resolution: a complete method for higher
order logic
Report 1117, Jennings Computing Center, Case Western
Reserve University (1972).

[17] Johnson D.E.
Finding all the Elementary Circuits of a Directed Graph
Technical Report 145, Computer Science Department,

[18] Kanoui H.
Application de la démonstration automatique aux
manipulations algébrique et à l'intégration formelle
sur ordinateur
Groupe d'Intelligence Artificielle, U.E.R. de Luminy,
Marseille (1973).

The Art of Computer Programming, Volume I: Fundamental
Algorithms
Addison-Wesley (1968).

Predicate logic as a programming language
Logic for problem-solving
DCL Memo 75, Department of Artificial Intelligence,

[22] Kowalski R.A.
A proof procedure using connection graphs

Linear resolution with selection function
Artificial Intelligence 2, 227-260 (1971).

[24] Loveland D.W.
Mechanical theorem-proving by model elimination

[25] Loveland D.W.
A simplified format for the model elimination
theorem-proving procedure

[26] Loveland D.W.
A linear format for resolution

[27] Loveland D.W.
A unifying view of some linear herbrand procedures

[28] Luckham D.
Refinements in resolution theory

[29] Manna Z. and Waldinger R.
The Logic of Computer Programming
to appear in Computing Surveys

Problem Solving Methods in Artificial Intelligence

[31] Nilsson N.J. and Fikes R.E.
STRIPS: A new approach to the application of theorem
proving to problem solving
Technical note 43, Artificial Intelligence Group,
[32] Paséro R.
Representation du français en logique du premier ordre en vue de dialoguer avec un ordinateur

[33] Paterson M.S. and Wegman M.N.
Linear Unification

[34] Read R.C. and Tarjan R.E.
Bounds on Backtrack Algorithms for listing cycles, paths and spanning trees
Memo ERL-M433, Electronics Research Laboratory, College of Engineering, University of California, Berkeley (1973).

[35] Roberts G.
An Implementation of PFCLOG

[36] Robinson J.A.
A machine oriented logic based on the resolution principle

[37] Rulifson J.F.
OA4 programming concepts

[38] Shostak R.E.
Refutation Graphs
Artificial Intelligence 7, 51-64 (1976).

[39] Sickel S.
A Search Technique for Clause Interconnectivity Graphs

[40] Szwarcfiter J.L. and Lauer P.E.
A New Backtracking Strategy for the Enumeration of the Elementary Cycles of a Directed Graph
Technical Report 69, Computing Laboratory, University of Newcastle upon Tyne (1975).

[41] Venturini-Zilli N.
Complexity of the unification algorithm for first-order expressions
Research report, Consiglio Nazionale delle Ricerche, Instituto per le applicazioni del calcolo (1975).
[42] Warren D.E.D.

WARPLAN: a system for generating plans
DCL Memo 76, Department of Artificial Intelligence,

[43] Yates R., Raphael E. and Hart T.

Resolution Graphs

[44] Zamov N.I. and Sharonov V.I.

On a class of strategies which can be used to establish
decidability by the resolution principle
Issled., po konstruktivnoye matematikye i
matematicheskie logikye v.3 no.16, 54–64 (1969).