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 unsolveable, 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 Naarten 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

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 Zamov 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 <u>complete</u> 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' PROLCG 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 Model Eliminaticn [24,25,27], and Kowalski and Kuehner as SL-resclution [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 unsolveable 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 regenerated. 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 Hart [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 $\langle V(G),E(G),\psi_G\rangle$, where V(G) is a nonempty set of _vertices_, E(G) is a set of _arcs_, disjoint from V(G), and ψ_G is an _incidence function_ from E(G) to V(G) X 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 ψ_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 ($1 \le i \le j \le n$), $e_i = e_j$ implies $i = j$. A directed trail is called a directed path if for all i and j ($1 \le i \le j \le n$), $v_i = v_j$ 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 $\langle V(G), E(G), I(G), \psi_G \rangle$, 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 ψ_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 ψ_G of a digraph G is injective, we can consider E(G) as being a subset of V(G) X V(G) (or V(G) X I(G) X 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,degree) where:

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

(ii) degree is a function from M ∪ P ∪ {-} to the nonnegative integers, where "-" is a special symbol not in M, P or V, and 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 ∪ P ∪ {-} are called function symbols. If s is any function symbol and degree(s) = m, we say that s *is of degree m*.

We juxtapcse 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 symbcl of degree 0.

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

An expressions cf 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 $P(p_1,...,p_n)$, where P is a predicate symbol of degree $n>0$, and $p_1,...,p_n$ 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 $\neg$L. Note that $\neg$ 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:

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

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

2.2.9: 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 $(1 \le i \le n)$, where $p = f(p_1,\ldots,p_n)$.

Note that if q is a subformula of p, then $\mathrm{ord}(p) \geq \mathrm{ord}(q)$, and $\mathrm{ord}(p) = \mathrm{ord}(q)$ 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 $(1 \le i \le n)$, $v_i \neq p_i$. $v_1, \ldots, v_n$ are called the <u>replaced variables</u> of the substitution, and each element of the substitution is called a <u>component</u>. We will denote substitutions by lower case Greek letters, except for the empty substitution, denoted by $\emptyset$.

<u>2.3.2</u>: <u>Definition</u>: If p is a formula and θ is a substitution, the <u>application of θ to p</u>, denoted $p\theta$, is the formula defined by:

$$p\theta = \begin{cases} q & \text{if } ord(p) = 0, \text{ and } (p,q) \in \theta \\ p & \text{if } ord(p) = 0 \\ & \quad \text{and } p \text{ is not a replaced variable of } \theta \\ f(p_1\theta, \ldots, p_n\theta) & \text{if } p = f(p_1, \ldots, p_n) \end{cases}$$

If E is a set of formulae, $\mathcal{E}$ is a set of sets of formulae, and θ is a substitution, we define:

$$E\theta = \{p\theta \mid p \in E\}$$

$$\mathcal{E}\theta = \{E\theta \mid E \in \mathcal{E}\}$$

called the <u>application</u> of θ to E, and the <u>application</u> of θ to $\mathcal{E}$. If X is a formula, set of formulae or set of sets of formulae, and θ is a substitution, we call $X\theta$ an <u>instance</u> of X, and call X a <u>generalisation</u> of $X\theta$. Note that since $X\emptyset = X$, X is both an instance and a generalisation of itself.

2.3.3: *Definition*: If θ and γ are substitutions, the *composition of θ with γ*, denoted $\theta \bullet \gamma$, is the substitution:

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

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

2.3.4: *Definition*: A substitution θ *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 θ is a *unifier* for E. θ is called a *most general unifier* (*mgu*) for E if and only if for every unifier γ for E, there is a substitution β such that $\gamma = \theta \bullet \beta$.

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

2.3.5: *Definition*: If $\mathcal{E}$ is a set of sets of formulae and θ is a substitution, θ *unifies* $\mathcal{E}$ if and only if θ unifies E for each $E \in \mathcal{E}$. 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 mguE, some mgu of E. We use this notation also for sets of sets of formulae. Note that if E is a set of formulae, then mguE = mgu$\{E\}$.

<u>2.3.6</u>: <u>Lemma</u>: 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_2mguX_1 is unifiable.

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

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

<u>Proof</u>:

(i) (a) If $X_1 \cup X_2$ is unifiable, let θ be any unifier for $X_1 \cup X_2$. θ unifies X_1, so that by the definition of mgu, θ = mgu$X_1 \bullet \beta$ for some substitution β. But θ unifies X_2, so that β unifies X_2mguX_1. Therefore X_1 and X_2mguX_1 are unifiable.

(b) Suppose X_1 and X_2mguX_1 are unifiable, and let θ be a unifier for X_2mguX_1. Now mguX_1 unifies X_1, so that mgu$X_1 \bullet \theta$ unifies X_1; also θ unifies X_2mguX_1 so that mgu$X_1 \bullet \theta$ unifies X_2. Hence mgu$X_1 \bullet \theta$ 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), mgu(X_2mguX_1) exists. Now mguX_1 unifies X_1, so that mgu$X_1 \bullet$(X_2mguX_1) unifies X_1. Also:

$$X_2\text{mgu}X_1 \bullet \text{mgu}(X_2\text{mgu}X_1) = (X_2\text{mgu}X_1)\text{mgu}(X_2\text{mgu}X_1)$$

So $mguX_1 \bullet mgu(X_2 mguX_1)$ clearly unifies X_2. Therefore $mguX_1 \bullet mgu(X_2 mguX_1)$ is a unifier for $X_1 \cup X_2$.

Suppose θ is a unifier for $X_1 \cup X_2$, then since θ unifies X_1:

$$\theta = mguX_1 \bullet \beta \text{ for some substitution } \beta$$

But θ unifies X_2, so that β unifies $X_2 mguX_1$.

$\therefore\ \beta = mgu(X_2 mguX_1) \bullet \alpha$ for some substitution α

$\therefore\ \theta = mguX_1 \bullet mgu(X_2 mguX_1) \bullet \alpha$

Therefore $mguX_1 \bullet mgu(X_2 mguX_1)$ is an mgu for $X_1 \cup X_2$.

$\square$

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 γ is a renaming, then γ^{-1} is the renaming $\{(u,v)\,|\,(v,u) \in \gamma\}$. Clearly, if γ is a renaming, then $\gamma \bullet \gamma = \gamma$, $(\gamma^{-1})^{-1} = \gamma$, and $\gamma \bullet \gamma^{-1} = \gamma^{-1}$.

2.3.8: Definition: If p is a formula and γ 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 $(X\gamma)\gamma^{-1} = X\gamma\bullet\gamma^{-1} = X\gamma^{-1} = X$: X is therefore a variant of $X\gamma$.

2.4: Predicate calculus

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

2.4.1: Definition: A clause is a finite set of literals. The empty clause is denoted by $\square$.

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

$$\Sigma(L) = T \text{ iff } \Sigma(\neg L) = F$$

We extend the domain of every valuation Σ to the set of all clauses, as follows:

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

$$\Sigma(\mathcal{C}) = T \text{ iff } \Sigma(L) = T \text{ for some } L \in \mathcal{C}$$

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

$$\Sigma(\mathcal{C}) = T \text{ iff } \Sigma(\mathcal{C}\theta) = T \text{ for all variable-free instances } \mathcal{C}\theta \text{ of } \mathcal{C}$$

__2.4.3__: __Definition__: A valuation Σ is said to __satisfy__ a clause $\mathcal{C}$ if and only if $\Sigma(\mathcal{C}) = T$. We also say that Σ is a __model__ for $\mathcal{C}$ Similarly, a valuation Σ is said to __satisfy__, or to be a __model__ for a set of clauses $\mathcal{S}$ if and only if $\Sigma(\mathcal{C}) = T$ for all $\mathcal{C} \in \mathcal{S}$

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:

$\mathcal{S} = \{$ $\{P(x,y),\ P(x,f(x)),\ -Q(x)\}$,

$\{Q(x),\ R(x,a),\ -S(x)\}$,

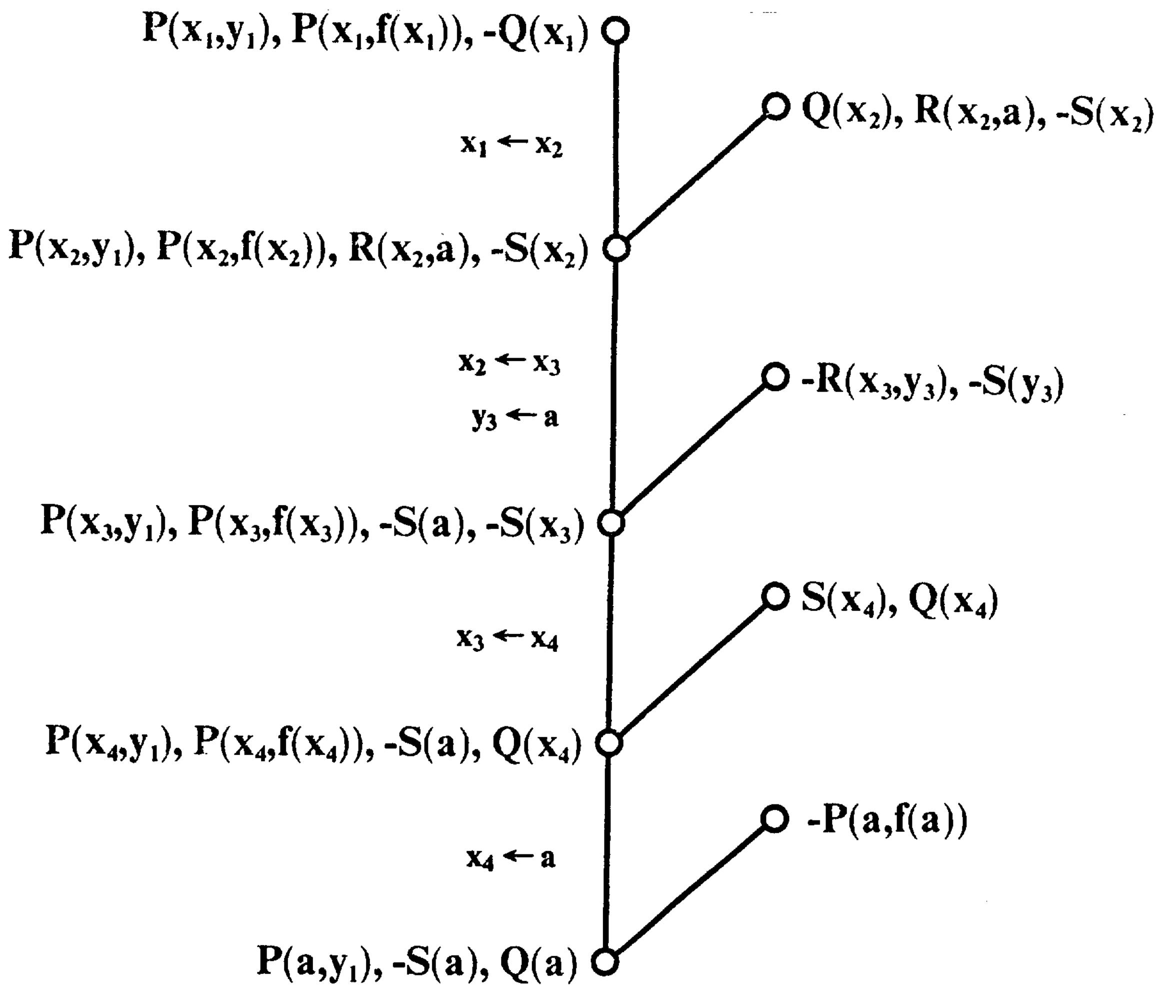
$\{S(x),\ Q(x)\}$,

$\{-R(x,y),\ -S(y)\}$,

$\{-P(a,f(a))\}$ $\}$

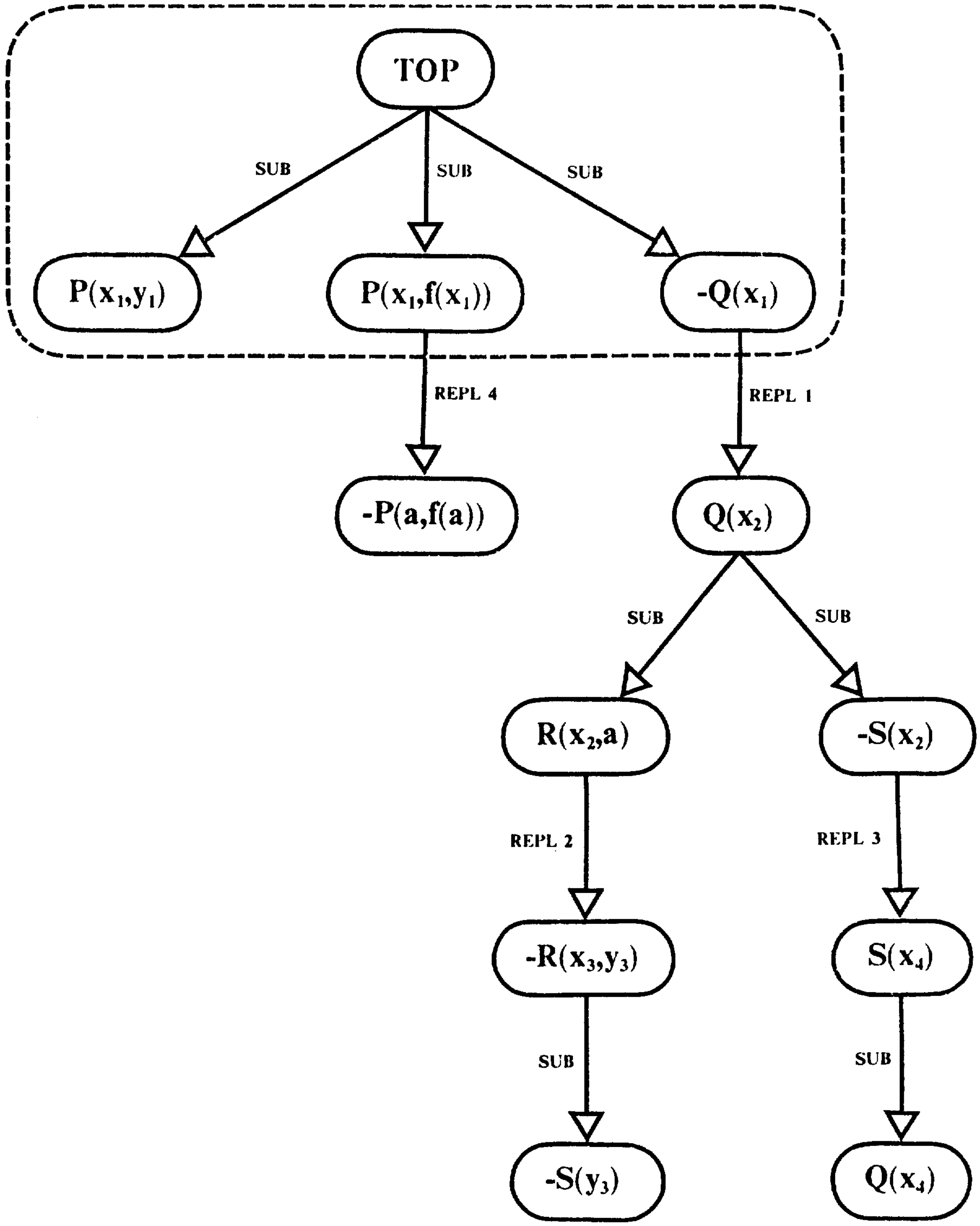
where a is a constant.

Figure 3.1 is a linear binary resolution deduction from $\mathcal{S}$, 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 $\mathcal{C}$ of some clause from $\mathcal{S}$, and have performed a binary resolution on the subproblem at the tail of the arc, using the literal of $\mathcal{C}$ which appears as the head of the arc. The remaining literals of $\mathcal{C}$ 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


$P(x_1,y_1), P(x_1,f(x_1)), -Q(x_1)$
$Q(x_2), R(x_2,a), -S(x_2)$
$x_1 \leftarrow x_2$
$P(x_2,y_1), P(x_2,f(x_2)), R(x_2,a), -S(x_2)$
$x_2 \leftarrow x_3$
$y_3 \leftarrow a$
$-R(x_3,y_3), -S(y_3)$
$P(x_3,y_1), P(x_3,f(x_3)), -S(a), -S(x_3)$
$S(x_4), Q(x_4)$
$x_3 \leftarrow x_4$
$P(x_4,y_1), P(x_4,f(x_4)), -S(a), Q(x_4)$
$-P(a,f(a))$
$x_4 \leftarrow a$
$P(a,y_1), -S(a), Q(a)$

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.

<u>Figure 3.1</u>



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

<u>Figure 3.2</u>

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.

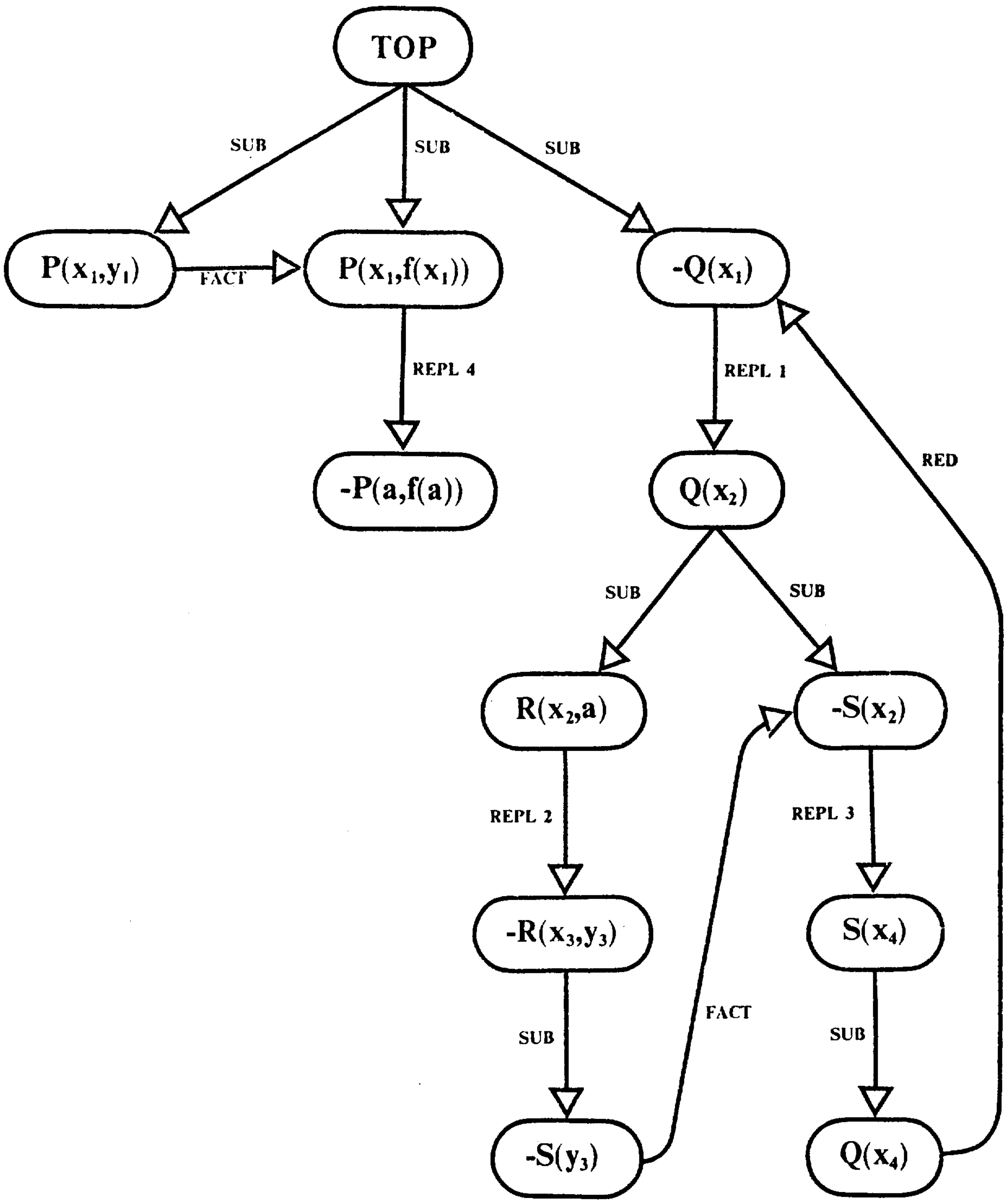
At each stage during the construction of this tree, the set of open subproblems corresponds to the clause deduced by the equivalent linear resolution. If at some stage there were no open subproblems left, then in the corresponding linear resolution, the empty clause would have been obtained, showing the set $\mathcal{S}$ 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 <u>one</u> 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 $\mathcal{S}$. 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 $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\mathcal{S}$ is a set of clauses, a *deduction plan for $\mathcal{S}$* is a digraph G, where:

(a) $E(G)$ is divided into four mutually disjoint sets REPL(G), SUB(G), RED(G) and FACT(G),

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

(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 $\mathcal{S}$ as "plans for $\mathcal{S}$" , or when the context ensures that no ambiguity is likely, simply as "plans".

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

<u>3.1.2</u>: If G is a plan, and H is any subgraph of G, we can meaningfully refer to SUB(H), FACT(H), RED(H) and REPL(H). Henceforth, in any subgraph H of a plan G, SOL(H) will be used as an abbreviation for FACT(H) $\cup$ RED(H) $\cup$ REPL(H).

__3.1.3__: __Definition__: If G is a plan for $\mathcal{S}$, and H is a subgraph of G, then H is a __subplan of G for $\mathcal{S}$__ if and only if for every $x \in V(H)$:

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

(ii) $(y,x) \in REPL(G) \Rightarrow (y,x) \in E(H)$

(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 $\mathcal{S}$__ if there exists a plan G for $\mathcal{S}$, and H is a subplan of G for $\mathcal{S}$. 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 $\mathcal{S}$ but not a plan for $\mathcal{S}$, it can be shown that there exists a set $\mathcal{S}_1$ of clauses for which H is a plan, such that $\mathcal{S}_1$ is satisfiable if and only if $\mathcal{S}$ is satisfiable.

__3.1.4__: __Definition__: If H is a subplan, the __top clause__ of H is the set $\{ v \mid (TOP,v) \in E(H) \}$.

__3.1.5__: __Definition__: If H is a subplan, the set:

$$\{ v \mid \exists x \in V(H) \text{ such that } (x,v) \in SUB(H) \}$$

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.

3.1.6.0: Basis

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

$$V(G) = \{ \text{TCP} \} \cup \mathcal{C}$$

$$SUB(G) = \{ (\text{TCP},l) \mid l \in \mathcal{C} \}$$

$$SCL(G) = \emptyset$$

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

3.1.6.1: Induction

Rule (1): Replacement

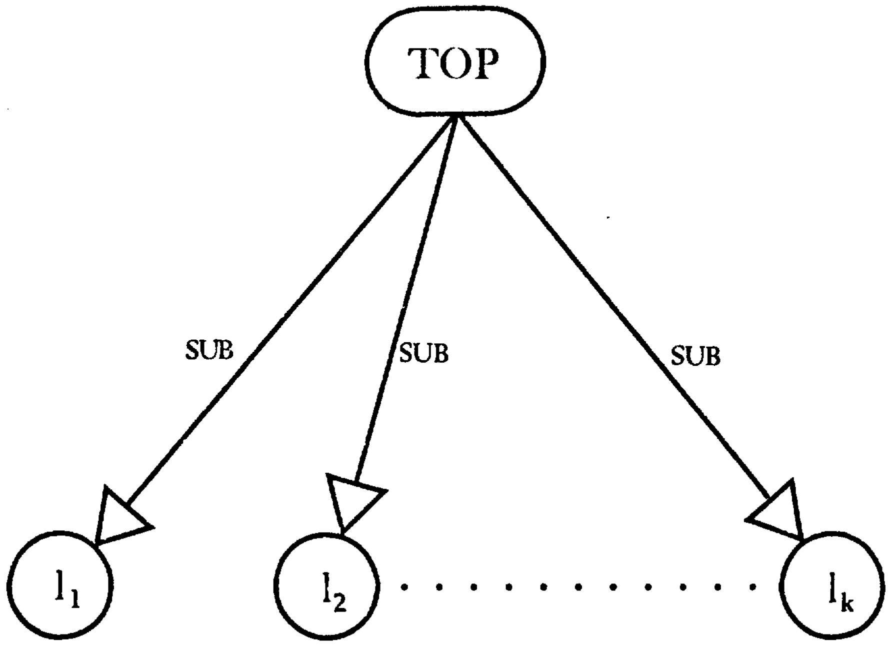
(A) Simple replacement

If G is a plan and:

(a) v $\in$ os(G),

(b) l $\in$ $\mathcal{C}$,

(c) $\mathcal{C}$ is a variant of some clause in $\mathcal{S}$ and contains no variables which occur in any vertices of G;



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:

$$V(G') = V(G) \cup \mathcal{C}$$

$$REPL(G') = REPL(G) \cup \{(v,l)\}$$

$$SUB(G') = SUB(G) \cup \{ (l,m) \mid m \in \mathcal{C}-\{l\} \}$$

$$FACT(G') = FACT(G)$$

$$RED(G') = RED(G)$$

(E) Ancestor replacement

If G is a plan, and:

(a) $v \in os(G)$,

(b) $l \in \mathcal{C}$,

(c) $\mathcal{C} = os(H)\gamma$, where H is a subplan of G, and γ 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 <u>replaced through</u> <u>l</u>, and that v is <u>replaced by</u> <u>$\mathcal{C} - \{l\}$</u>. 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 <u>literal</u> <u>occurrences</u> rather than as a set of literals: the replacement then introduces a new vertex corresponding to the new occurrence of m.

<u>Rule</u> (<u>2</u>): Reduction

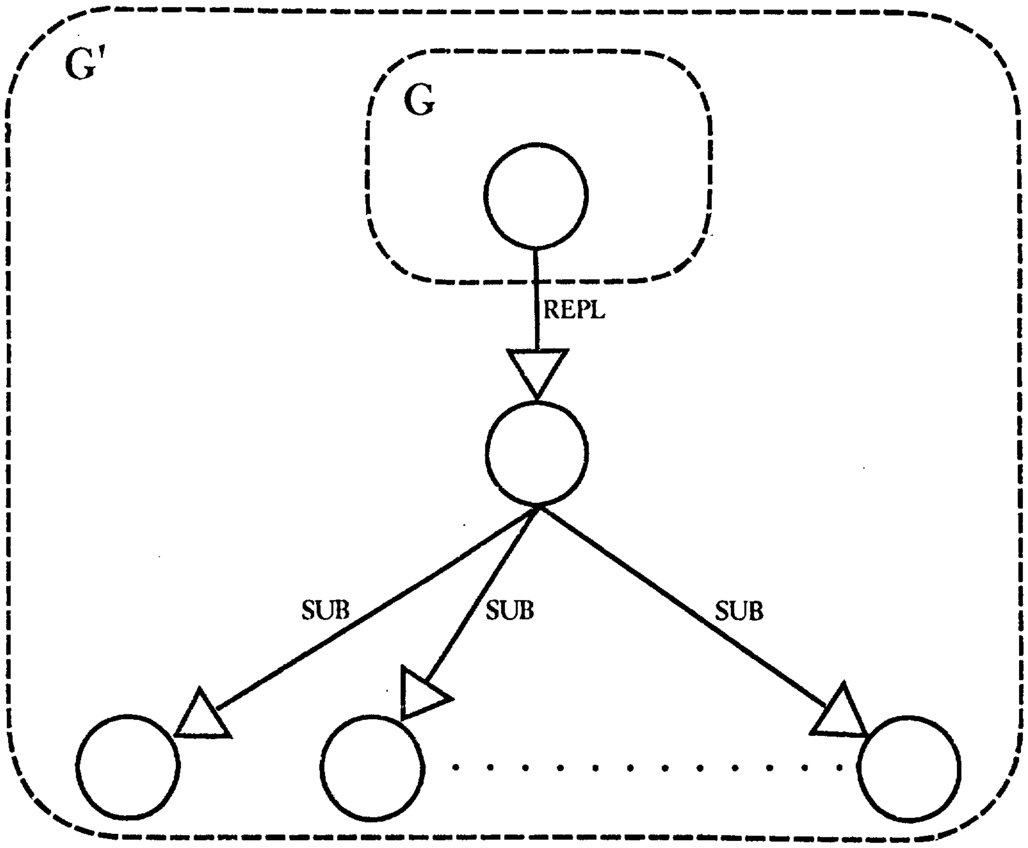
If G is a plan and:

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

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

(c) if $(x,y) \in FACT(G)$, and either $y = u$ or there is a walk from y to u which does not pass through v and contains no arcs in $RED(G)$, 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 _u is reduced to y_. Reduction may be represented pictorially as in figure 3.6.

Rule (_3_): Factoring

Factoring, like replacement, divides into two cases.

(A) Simple factoring

If G is a plan and:

(a) $x \in os(G)$,

(b) $y \in os(G) - \{x\}$;

then G' is a plan, where G' is defined by:

$$V(G') = V(G)$$

$$FACT(G') = FACT(G) \cup \{(x,y)\}$$

$$RED(G') = RED(G)$$

$$REPL(G') = REPL(G)$$

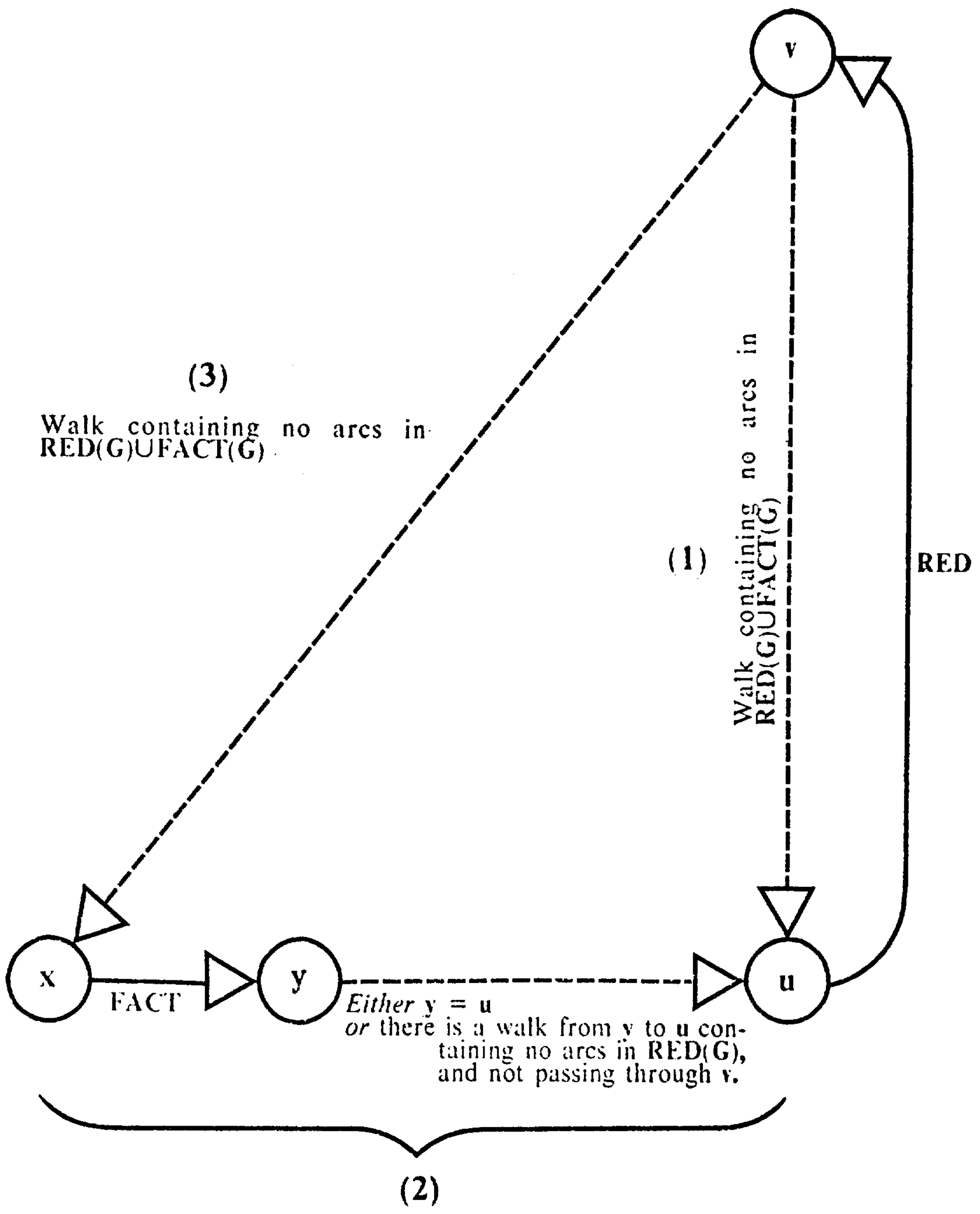
$$SUB(G') = SUB(G)$$

(B) Back factoring

If G is a plan and:

(a) $x \in os(G)$,

(b) $y \in cs(G)$,


(3)
Walk containing no arcs in RED(G)UFACT(G)
(1)
Walk containing no arcs in RED(G)UFACT(G)
RED
v
x
y
u
FACT
Either y = u or there is a walk from y to u containing no arcs in RED(G), and not passing through v.
(2)

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);

<u>and</u> if u,v ∈ 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 ∈ 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

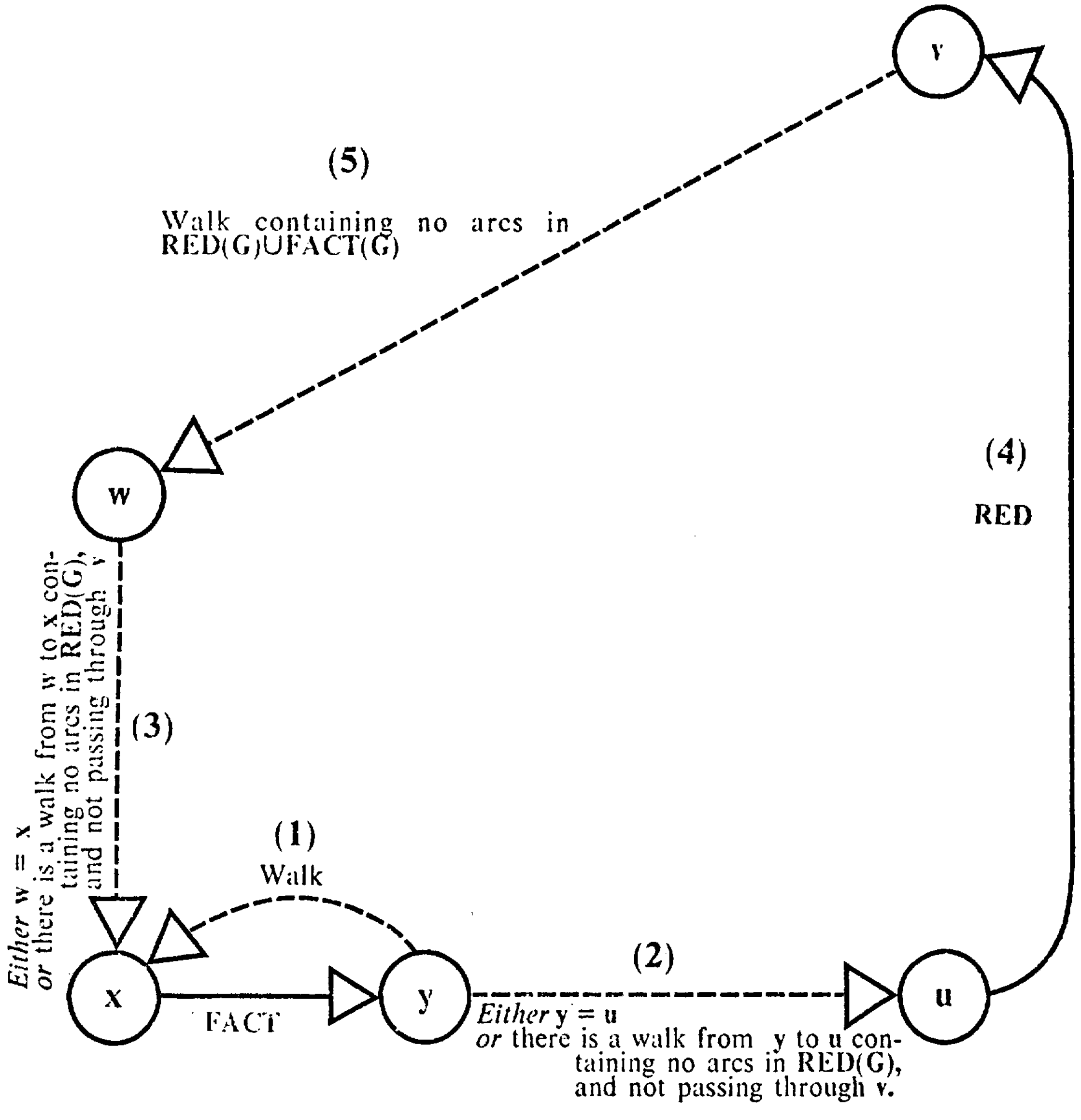
then G' is a plan, where G' is defined as for simple factoring. In both types of factoring, we say that <u>x is factored to y</u>. A pictorial representation of backfactoring is shown in figure 3.7

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

<u>3.1.7</u>: <u>Example</u>: Let $\mathcal{S}$ 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 $\mathcal{S}$. Each arc of SCL(G) is labelled with the name of the


v
(5)
Walk containing no arcs in RED(G)UFACT(G)
w
(4)
RED
Either w = x or there is a walk from w to x containing no arcs in RED(G), and not passing through v
(3)
(1)
Walk
x
FACT
y
(2)
u
Either y = u or there is a walk from y to u containing no arcs in RED(G), and not passing through v.

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).

<u>Figure 3.7</u>

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 REPL 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.

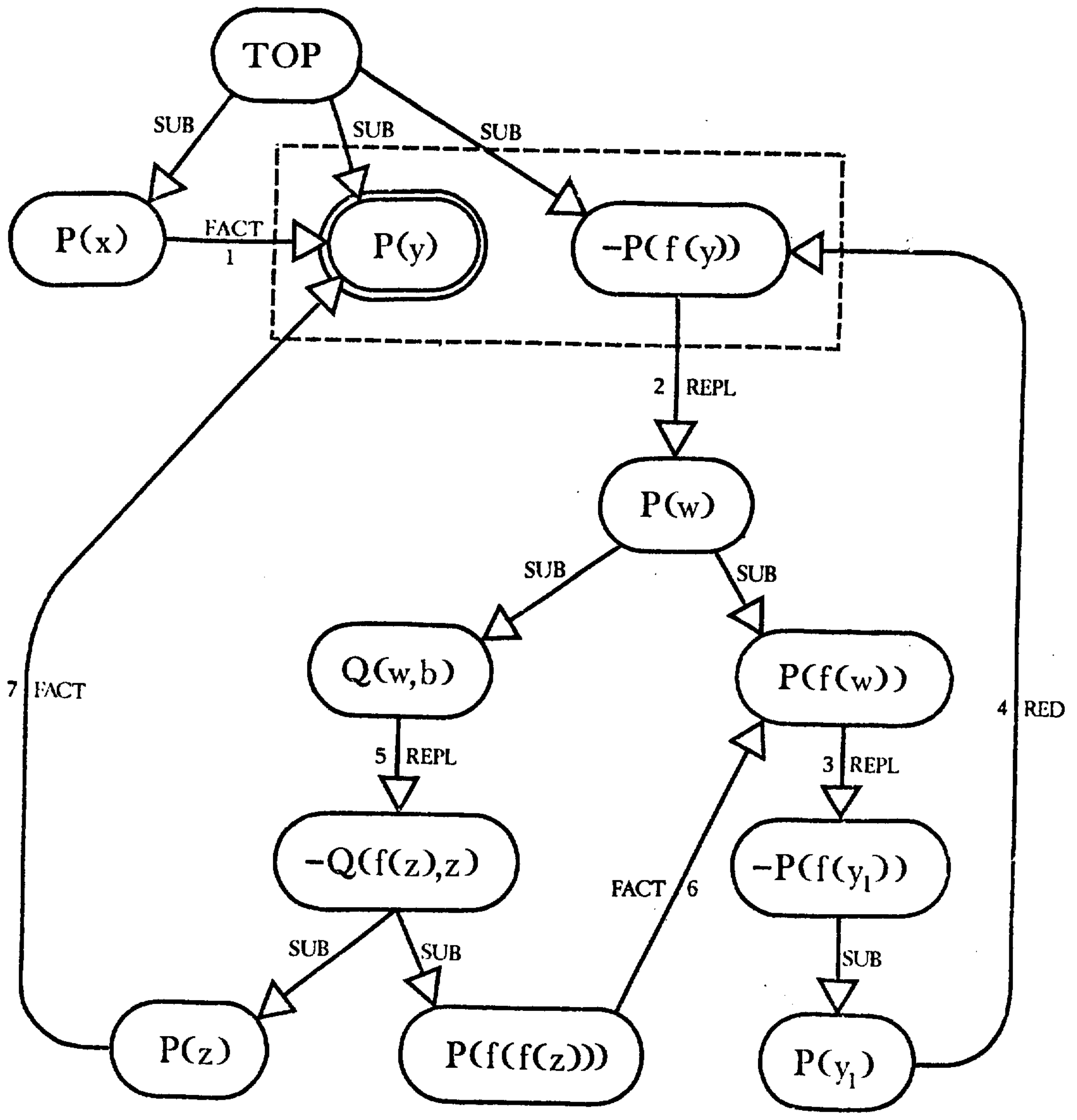
3.2: Comments, notation and some preliminary results

Every vertex of a subplan, except TOP, has indegree at least 1; TOP 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: TOP 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 $i \le n$, 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 SOL(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 $\mathcal{S}$ 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 $SOL(G)$. 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:

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

$$\text{or}\quad SOL(G_2) = SOL(G_1) \cup \{(x,y)\}$$

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

...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.

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

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

<u>Basis</u>: G_0 is a basic plan, and therefore has no closed walks.

<u>Induction</u>: 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:

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

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 $y \in os(G_i)$ 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.

$\square$

3.2.3: Lemma: If w is any closed subproblem of a plan G, then G' is a subplan of G, where G' is defined by:

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

$$E(G') = E(G) - \{(u,v) \mid \begin{array}{l} (u,v) \in E(G) \\ \text{and either } u = w \\ \quad \text{or } u \text{ is a direct} \\ \quad\quad \text{descendant of } w \\ \quad \text{or } v \text{ is a direct} \\ \quad\quad \text{descendant of } w \end{array} \}$$

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

<u>either</u> x = w, which is impossible since w is a subproblem and x is not.

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

<u>or</u> 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 $(y,x) \notin REPL(G')$

<u>either</u> y = w, so that x is a direct descendant of w

<u>or</u> y is a direct descendant of w, so that x is also a direct descendant of w

<u>or</u> 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.

$\square$

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 $0<k\le n$:

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

Suppose x is replaced through y by $y_1,\ldots,y_m$; then y_i lies on the walk for some i such that $1\le i\le m$, 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 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 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 RED(G).

(iii) Suppose either z = u or there is a walk from z to u in G containing no arcs of RED(G). Then in either case there is a walk from w to u, containing no arcs of 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 $0<k\le n$:

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

and the walk from w to u containing no arcs of 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_i lies on the walk from w to u, where $1\leq i\leq m$, and y_i 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, $(x,y) \in FACT(G)$, $(u,v) \in RED(G)$, and $x,u \in os(G_k)$.

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

$$SOL(G_{k+1}) = SOL(G_k) \cup \{(x,y)\}$$

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

(ii) If G'_{k+1} is the subplan of G defined by:

$$SOL(G'_{k+1}) = SOL(G_k) \cup \{(u,v)\}$$

then G'_{k+1} 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+1} 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 $os(G_m) \cap cs(G)$ 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+1} by:

$$cs(G_{m+1}) = cs(G_m) \cup \{x_1\}$$

It remains to prove that G_{m+1} 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+1} is a plan.

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

(d) Suppose x_1 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, $\exists x_i \in os(G_m)$ 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_1 in any derivation of G. This contradicts the ordering imposed on $os(G_m) \cap cs(G)$.

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

If $e \in SUB(G)$, then $z \in V(H) - V(G_m)$, which we have already shown to be impossible.

If $e \in SOL(G)$, 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_1 in any derivation of G; this contradicts the ordering imposed on $os(G_m) \cap cs(G)$.

H is therefore a subplan of G_m, so the conditions for closing x_1 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.

$\square$

3.2.7: <u>Corollary</u>: 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: <u>Lemma</u>: 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.

<u>Proof</u>: 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.

<u>Basis</u>: $n=1$. Fact is not empty, so there exists $(x_1,x_2) \in$ FACT(G). The walk from x_1 to x_2 consisting of this single arc has the required properties.

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

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

<u>or</u> 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 REPL(G) $\cup$ SUB(G), and a walk from w to z consisting of the single arc $(w,z) \in$ FACT(G). Appending these two walks to the end of the walk $(x_1,x_2),\dots,(x_{n-1},x_n)$, 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 RED(G). Such a walk must encounter some vertex more than once: hence there is a closed walk in G containing no arcs of RED(G). By lemma 3.2.2, some arc (x,y) on this walk is in FACT(G). Consequently, there exists $(x,y) \in$ FACT(G), and a walk from y to x with no arcs in RED(G). This contradicts lemma 3.2.3, thereby disproving hypothesis (A), and establishing the result.

$\Box$

3.2.8

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

3.2.9: <u>Example</u>: Consider the plan G_2 for the set of clauses $\{ \{v_1,v_2\}, \{v_3,v_4\} \}$, where G_2 has the derivation (G_0,G_1,G_2) defined as follows:

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

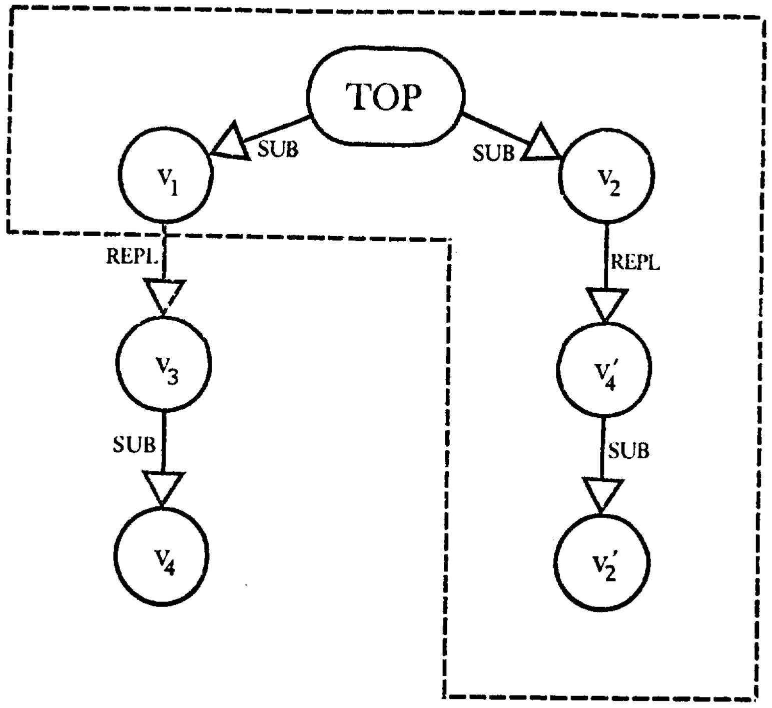
$$V(G_1) = V(G_0) \cup \{v_3,v_4\}$$
$$SUB(G_1) = SUB(G_0) \cup \{(v_3,v_4)\}$$
$$REPL(G_1) = REPL(G_0) \cup \{(v_1,v_3)\}$$

$$V(G_2) = V(G_1) \cup \{v_2',v_4'\}$$
$$SUB(G_2) = SUB(G_1) \cup \{(v_4',v_2')\}$$
$$REPL(G_2) = REPL(G_1) \cup \{(v_2,v_4')\}$$

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

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

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

Case(b): Suppose $RED(G_m) \cup FACT(G_m)$ 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 $\mathcal{C}-\{y\}$, where $\mathcal{C}$ is a variant of some clause in $\mathcal{S}$. Now since G_m is a subplan, $\mathcal{C} \subseteq V(G_m)$. Also $\mathcal{C}-\{y\} \subseteq os(G_m)$, since otherwise $x_i \in \mathcal{C}-\{y\}$ for some $i < k$ since x_i is closed before x_k in the derivation D. Hence we can define G_{m-1} by:

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

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

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 SOL(G). The only such subplan is G_0, the basic plan of G. Hence G_m is a plan.

$\square$

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.

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

<u>Proof</u>: We will show that there exists a derivation $(G_0,\ldots,G_n)$ of G such that for some $m \le n$, 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 $|FACT(G_{m-1})| = |FACT(G_m)|-1$, a finite number of applications of this process will yield a derivation of G in which all factorings are simple.

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

<u>Case(a)</u>: Suppose for some $(x,y) \in FACT(G)$, 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 $(x,y) \in \mathrm{FACT}(G)$, y is closed.

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

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 $1\le i<k$:

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

(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 $(x,y) \in \mathrm{FACT}(G)$, y is closed, lemma 3.2.8 ensures that $\exists(x,y) \in \mathrm{FACT}(G)$ 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_1 and y_1 obviously satisfy the conditions (i) to (iv).

<u>Induction</u>: 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) $(x_k, y_k) \in FACT(G)$

(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 REPL(G) $\cup$ SUB(G), contrary to lemma 3.2.2. Thus hypothesis (A) is disproved, and there exists (x,y) $\in$ 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) $\in$ 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 $|FACT(G_{m-1})| = |FACT(G_m)| - 1 = |FACT(G)| - 1$. This completes the proof.

$\square$

<u>3.2.12</u>: <u>Corollary</u>: 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.

<u>Proof</u>: We define a subgraph H of G by:

$V(H) = V(G)$

$E(H) = E(G) - RED(G)$

One application of lemma 3.2.3 and one application of lemma 3.2.10 for each arc of RED(G) removed, proves that 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.

$\square$

3.2.13: Definition: A plan H is said to be *closed* if os(H) = $\emptyset$.

3.2.14: Definition: Denote by 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:

$$P = \{ \{\tilde{l}_1',\tilde{l}_2'\} \mid \exists \mathcal{C}_1,\mathcal{C}_2 \in \mathcal{S} \text{ such that } l_1 \in \mathcal{C}_1,\ l_2 \in \mathcal{C}_2,\ l_1' \text{ is a variant of } l_1,\ l_2' \text{ is a variant of } l_2 \}$$

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

$$C(e) = \begin{cases} \emptyset & \text{if } e \in SUB(H) \\ \{\{x,y\}\} & \text{if } e = (x,y) \in FACT(H) \\ \{\{x,\neg y\}\} & \text{if } e = (x,y) \in RED(H) \\ & \text{or } e = (x,y) \in REPL(H) \text{ where} \\ & \quad (x,y) \text{ is a simple replacement} \\ \{\{x,\neg y\}\} \cup C(K)\gamma & \\ & \text{if } e = (x,y) \in REPL(H) \text{ where} \\ & \quad (x,y) \text{ is an ancestor replacement} \\ & \quad \text{using the clause } os(K)\gamma\text{, and } K \\ & \quad \text{is a subplan of } G \end{cases}$$

The set $\bigcup_{e \in E(H)} C(e)$ will be called the <u>constraint set</u> 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.

<u>3.2.15</u>: <u>Definition</u>: A subplan H is said to be <u>correct</u> if C(H) is unifiable, in which case we denote the most general unifier of C(H) by θ(H), and call the clause os(H)θ(H) the <u>clause deduced by H</u>. Note that every subplan of a correct plan is correct, and that the clause deduced by a closed plan is the empty clause □.

3.2.16: Example: If $\mathcal{S}$ contains the empty clause □, we may use □ as the top clause for constructing the basic plan G = <{TOP},∅>; then G is a closed plan, and os(G) = ∅.

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:

$$\theta(G) = \{\ (x,a),\ (y,a),\ (z,a),\ (w,f(a)),\ (y_1,f(a))\ \}$$

---

1 {P(x), P(y)}

2 {-P(f(y)), -P(w)}

3 {P(f(w)), P(f(y_1))}

3 {P(x_1), P(y_1)}

4 {P(y_1), 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)}θ(G) = {P(a)}.

3.2.18: Lemma: If G is a correct plan for $\mathcal{S}$ generated by rules (1)A and (2) only, then there exists a correct plan G' for $\mathcal{S}$ generated by rules (1) and (3)A only, such that:

$$os(G)\theta(G) = os(G')\theta(G')\bullet\alpha$$

for some substitution α.

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 $(G_0,\ldots,G_n)$ 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 α_j such that:

$$z\theta(G_j) = z\theta(G'_j)\bullet\alpha_j \text{ for all } z \in s(G_j)$$

This sequence is constructed as follows:

(A) $G'_m = G_m$

(B) Suppose the construction is complete up to G_j'. G_{j+1} is generated from G_j by reducing some $x \in os(G_j)$ to some direct ancestor y. Let H be defined by:

$V(H) = V(G_j') - \{z \mid z \text{ is a direct descendant of } y\}$

$E(H) = E(G_j') - \{(w,z) \mid w \text{ or } z \text{ is a direct descendant of } y\}$

By lemma 3.2.3, H is a subplan of G_j'. Also $os(H) = \{y, x_1, \ldots, x_k\}$, where $\{x_1, \ldots, x_k\} \subseteq os(G_j')$. 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' by replacing $x \in os(G_j')$ through $y\gamma$ by $\{x_1\gamma, \ldots, x_k\gamma\}$.

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

$$FACT(G_j^i) = FACT(G_j^{i-1}) \cup \{(x_i\gamma, x_i)\}$$

Let $G_{j+1}' = G_j$; it remains to show that G_{j+1}' satisfies the required conditions.

(i) G_{j+1}' is clearly a plan, so we need only show that $C(G_{j+1}')$ is unifiable.

First we show that $\{x\theta(G_j'), \neg y\theta(G_j')\}$ is unifiable. Since $C(G_{j+1}) = C(G_j) \cup \{\{x, \neg y\}\}$ is unifiable, by lemma 2.3.6, $\{x\theta(G_j), \neg y\theta(G_j)\}$ is unifiable, so by the induction hypothesis condition (iv), $\{x\theta(G_j')\bullet\alpha_j, \neg y\theta(G_j')\bullet\alpha_j\}$ is unifiable and therefore $\{x\theta(G_j'), \neg y\theta(G_j')\}$ has a unifier $\alpha_j \bullet mgu\{x\theta(G_j), \neg y\theta(G_j)\}$. Therefore there is a substitution β such that:

(1)...$\alpha_j \bullet mgu\{x\theta(G_j), \neg y\theta(G_j)\} = mgu\{x\theta(G_j'), \neg y\theta(G_j')\}\bullet\beta$

We now proceed to show that $C(G'_{j+1})$ is unifiable.

(2)...$C(G'_{j+1}) = C(G'_j) \cup C(H)\gamma \cup \{\{x,\neg y\gamma\}\}$

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

Let $\delta = \gamma^{-1}\bullet\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\}$

We show that δ unifies each set in the above expression (2) for $C(G'_{j+1})$.

$C(G'_j)\delta = (C(G'_j)\gamma^{-1})(\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\})$

$= (C(G'_j)\theta(G'_j))mgu\{x\theta(G'_j),\neg y\theta(G'_j)\}$

since none of the replaced variables of γ^{-1} occur in $C(G'_j)$

So since $\theta(G'_j)$ unifies $C(G'_j)$, δ unifies $C(G'_j)$.

$(C(H)\gamma)\delta = (C(H)(\gamma\bullet\gamma^{-1}))(\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\})$

$= (C(H)\gamma^{-1})(\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\})$

by 2.3.7

$=(C(H)\theta(G'_j))\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\}$

since none of the replaced variables of γ^{-1} occur in $C(H)$

So since $C(H) \subseteq C(G'_j)$, $\theta(G'_j)$ unifies $C(H)$, so that δ unifies $C(H)$.

$\{\{x,\neg y\gamma\}\}\delta = \{\{x\gamma^{-1},\neg y\gamma^{-1}\}\}(\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\})$

by 2.3.7

$\{\{x,\neg y\}\}(\theta(G'_j)\bullet mgu\{x\theta(G'_j),\neg y\theta(G'_j)\})$

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

$= \{\{x\theta(G'_j),\neg y\theta(G'_j)\}\}mgu\{x\theta(G'_j),\neg y\theta(G'_j)\}$

Therefore δ unifies $\{\{x,\neg y\gamma\}\}$.

Finally, for each $i \in \{1,\ldots,k\}$:

$$\{x_i \gamma, x_i\}\alpha = \{x_i \gamma^{-1}, x_i \gamma^{-1}\}(\theta(G_j')\bullet mgu\{x\theta(G_j'), \neg y\theta(G_j')\})$$

by 2.3.7

Therefore δ unifies $\{x_i \gamma, x_i\}$.

Hence $C(G_j'+1)$ is unifiable so that $G_j'+1$ is correct.

(ii) $$os(G_j'+1) = os(G_j') - \{x\}$$
$$= os(G_j) - \{x\}$$
$$= os(G_j+1)$$

(iii) Since $\{x_1,\ldots,x_k\} \subseteq os(G_j')$, all the factorings performed in deriving $G_j'+1$ from G_j' are simple.

(iv) In the proof of (i) above, we have shown that $\gamma^{-1}\bullet\theta(G_j')\bullet mgu\{\{x\theta(G_j'), \neg y\theta(G_j')\}\}$ unifies $C(G_j'+1)$, so for some substitution τ:

(2)...$\theta(G_j'+1)\bullet\tau = \gamma^{-1}\bullet\theta(G_j')\bullet mgu\{\{x\theta(G_j'), \neg y\theta(G_j')\}\}$

Let β be the substitution defined in equation (1) in part (i) above, and let $\alpha_{j+1} = \tau\bullet\beta$, then if $z \in s(G_j+1)$:

$$z\theta(G_j'+1)\bullet\alpha_{j+1} = z(\theta(G_j'+1)\bullet\tau)\bullet\beta$$
$$= z\gamma^{-1}\bullet\theta(G_j')\bullet mgu\{\{x\theta(G_j'), \neg y\theta(G_j')\}\}\bullet\beta$$

from (2) above

$$= z\theta(G_j')\bullet mgu\{\{x\theta(G_j'), \neg y\theta(G_j')\}\}\bullet\beta$$

since none of the replaced variables of γ^{-1} occur in z

$$= z\theta(G_j')\bullet\alpha_j\bullet mgu\{\{x\theta(G_j), \neg y\theta(G_j)\}\}$$

by (1) in (i) above

$= z\theta(G_j)\bullet mgu\{\{x\theta(G_j), \neg y\theta(G_j)\}\}$

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:

$$\theta(G_{j+1}) = \theta(G_j)\bullet mgu\{\{x\theta(G_j), \neg y\theta(G_j)\}\}$$

$$\therefore z\theta(G_{j+1}) = z\theta(G'_{j+1})\bullet\alpha_{j+1}$$

The sequence having been constructed, let $G' = G'_n$ and $\alpha = \alpha_n$, then:

$$os(G) = os(G')$$

$$\text{and } z\theta(G) = z\theta(G')\bullet\alpha \quad \text{for all } z \in s(G)$$

$$\therefore os(G)\theta(G) = os(G')\theta(G')\bullet\alpha$$

□

<u>3.2.19</u>: <u>Lemma</u>: If G is a closed, correct plan for $\mathcal{S}$ generated by rules (1)A and (2) only, then there exists a closed, correct plan G′ for $\mathcal{S}$ generated by rules (1) and (3)B only.

<u>Proof</u>: If RED(G) = ∅ 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 and for $i > m$, G_i is derived by reduction from G_{i-1}. Let H be defined by:

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

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

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

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

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 back factoring only.

(iv) there is a substitution α_j such that:

$$z\theta(G_j) = z\theta(G'_j)\bullet\alpha_j \text{ for all } z \in s(G_j)$$

(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 $x \in os(G_j)$ to some direct ancestor y. Now $y \in os(H)$, 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 $(G_j^0, \ldots, G_j^k)$ as follows:

(a) Let $C(H)\gamma$ be a variant of C(H). G_j^0 is the plan obtained from G_j' by replacing $x \in$ os(G_j') through $y\gamma$ by $\{x_1\gamma, \ldots, x_k\gamma\}$.

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

$$FACT(G_j^i) = FACT(G_j^{i-1}) \cup \{(x_i\gamma, x_i)\}$$

Let $G_{j+1}' = G_j^k$; it remains to show that G_{j+1}' 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_j'). Hence all the factorings performed in deriving G_{j+1}' from G_j' are back factorings.

(v) G_m is a subplan of G_j' and hence of G_{j+1}'.

The sequence having been constructed, let $G' = G_n'$, then os(G') = os(G) = $\mathcal{C}$, and G' is correct.

$\square$

3.2.19

3.3: Soundness and Completeness

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

3.3.1: Definition: If G is a subplan, and γ is any substitution, we define:

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

Also, we denote the graph $\langle V(G)\gamma, E(G)\gamma\rangle$ 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 $\mathscr{S}$ of clauses, then clearly so is every variant of G.

3.3.2: Lemma: Let G be a correct plan for a set $\mathscr{S}$ of clauses, H a subplan of G, Σ a model for $\mathscr{S}$, and $H\Theta(H)\bullet\gamma$ 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 $x_0 = TOP$, $x_n \in os(H)$, and $\Sigma(x_i\Theta(H)\bullet\gamma) = T$ for every $x_i \in s(H)$, $0 < i \leq n$.

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 $\{x \mid (TOP,x) \in SUB(H)\}$ of H is a variant of some clause in $\mathcal{S}$, and therefore contains some literal x_1 such that $\Sigma(x_1\theta(H)\bullet\gamma) = T$. (TOP,x_1) 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 $(x_i,y) \in RED(H)$.

<u>either</u> the walk from TOP to x_i contains no arcs of 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.

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

<u>either</u> that part of the walk from x_{j+1} to x_i passes through y,

<u>or</u> 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.

$$\therefore \Sigma(y\theta(H)\bullet\gamma) = T$$

$$\begin{aligned}\text{But } \Sigma(y\theta(H)\bullet\gamma) &= \Sigma(\neg x_i\,\theta(H)\bullet\gamma)\\ &= F\end{aligned}$$

which is a contradiction. Therefore x_i is not closed by reduction in H. Hence we have three cases to consider:

(a) $x_i \in os(H)$, in which case $n = i$, and we have the required walk.

(b) x_i is closed by factoring in H, say $(x_i, z) \in FACT(H)$, in which case:

$$z\theta(H) = x_i\theta(H)$$

$$\therefore \Sigma(z\theta(H)\bullet\gamma) = \Sigma(x_i\theta(H)\bullet\gamma)$$

$$= T$$

Let $x_{i+1} = z$; (x_i, x_{i+1}) is then the next arc in the walk.

(c) x_i is closed by simple replacement through y in H; then we define $x_{i+1} = y$, so that $(x_i, x_{i+1}) \in REPL(H)$. In this case $\{x_{i+1}\} \cup \{x \mid (x_{i+1}, x) \in SUB(H)\}$ is a variant of a clause in $\mathcal{S}$, and therefore:

$$\Sigma([\{x_{i+1}\} \cup \{x \mid (x_{i+1}, x) \in SUB(H)\}]\theta(H)\bullet\gamma) = T$$

$$\text{But } x_{i+1}\theta(H) = \neg x_i\theta(H)$$

$$\therefore \Sigma(x_{i+1}\theta(H)\bullet\gamma) = \Sigma(\neg x_i\theta(H)\bullet\gamma)$$

$$= F$$

Hence $\exists z \in \{x \mid (x_{i+1}, x) \in SUB(H)\}$ such that:

$$\Sigma(z\theta(H)\bullet\gamma) = T.$$

We define $x_{i+2} = z$. Then $(x_i, x_{i+1}), (x_{i+1}, x_{i+2})$ 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 RED(H). By lemma 3.3.2, however, no closed walk can consist entirely of arcs in REPL(H) $\cup$ SUB(H). Therefore some arc (x_j, x_{j+1}) on this closed walk must belong to FACT(H): then there is a walk from x_{j+1} to x_j containing no arcs in RED(H), contrary to lemma 3.2.4. Hence the construction must terminate as in case (a).

<u>Induction</u>: 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 $K\alpha$ of some subplan K of G. Suppose x_i is replaced through $y\alpha$. Let $x_{i+1} = y\alpha$, then:

$$os(K) = \{y\} \cup \{w \mid (x_{i+1}, w\alpha) \in SUB(H)\}$$

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

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

G_{k-1} is a correct plan for $\mathcal{S}$, 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 $K\theta(K)\bullet\tau$ is a variable-free instance of $K\theta(K)$, then $\exists z \in os(K)$ such that:

$$\Sigma(z\theta(K)\bullet\tau) = T$$

Also we have:

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

$$\therefore \theta(H) \text{ unifies } C(K)\alpha$$

$$\therefore \alpha\bullet\theta(H) \text{ unifies } C(K)$$

Therefore:

(2)............$\alpha\bullet\theta(H) = \theta(K)\bullet\beta$

for some substitution β

$H\theta(H)\bullet\gamma$ is a variable-free instance of $H\theta(H)$. Suppose $(K\alpha)\theta(H)\bullet\gamma$ is not variable-free; this is possible since K is not necessarily a subplan of H. In this case, let γ_1 be some substitution such that $(K\alpha)\theta(H)\bullet\gamma\bullet\gamma_1$ is variable-free, then:

(3)........$x\theta(H)\bullet\gamma\bullet\gamma_1 = x\theta\gamma$ if $x \in V(H)$

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

$$\Sigma(z\theta(K)\bullet(\beta\bullet\gamma\bullet\gamma_1)) = T \quad \text{for some } z \in os(K)$$

Now $\Sigma(y\theta(K)\bullet(\beta\bullet\gamma\bullet\gamma_1)) = \Sigma(x_{i+1}\theta(H)\bullet(\gamma\bullet\gamma_1))$

by applying (2)

$= \Sigma(\neg x_i\theta(H)\bullet(\gamma\bullet\gamma_1))$

$= \Sigma(\neg x_i\theta(H)\bullet\gamma)$

by (3)

$$= F$$

$$\therefore z \neq y$$

$$\therefore z \in \{w \mid (x_{i+1}, w\alpha) \in SUB(H)\}$$

$$\text{Let } x_{i+2} = z\alpha$$

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

$$\text{and } \Sigma(x_{i+2}\theta(H)\bullet\gamma) = \Sigma((z\alpha)\theta(H)\bullet\gamma\bullet\gamma_1)$$

$$= \Sigma(z\theta(K)\bullet(\beta\bullet\gamma\bullet\gamma_1))$$

$$= T$$

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

□

3.3.3: <u>Theorem</u>: <u>The Soundness of Plans</u>

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

<u>Proof</u>: 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 TOP to y in G such that y ∈ 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) = ∅, 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.

<u>3.3.4.4</u>

(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 lit(l) and 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 lit(L).

(iv) For any substitution γ, 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:

<u>3.3.4.5</u>: <u>Definition</u>: If K is a chain, and $l,k \in K$, then we say that l <u>is to the left of k in K</u> if pos(l) < 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.

<u>3.3.4.6</u>: <u>Definition</u>: A chain K is said to be <u>preadmissible</u> if and only if:

(i) any two B-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 M is a set of elementary chains, a finite sequence $(K_0,\ldots,K_n)$ of chains is called an ME-deduction of K from M if K_0 is a variant of some chain in M, 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 M, 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 $\gamma = \text{mgu}\{k_m,\neg l_1\}$

(ii) Reduction

If K_{i-1} is admissible:

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

and $A(K_i) = A(K_{i-1})\gamma$

where $\gamma = \text{mgu}\{k_m,\neg k_j\}$

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

for some k_j to the left of k_m.

(iii) Contraction

If K_{i-1} is preadmissible, and the last occurrence of K_{i-1} is an A-occurrence, then:

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

$$\text{and } A(K_i) = A(K_{i-1}) - \{(k_m, m)\}$$

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

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

(a) If $k \in \mathcal{C}$

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 $\mathcal{C}$

then $L_1 = L_2$

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

3.3.4.10: Definition: A clause $\mathcal{C}$ is said to be *ME-deducible from* $\mathcal{S}$, a set of clauses, if there exists an ME-deduction $(K_0, \ldots, K_n)$ from some matrix set M for $\mathcal{S}$, such that $\mathcal{C} = \text{lit}(B(K_n))$.

__3.3.4.11__: __Theorem__:(__Loveland [25]__)

$\mathcal{S}$ is unsatisfiable if and only if the empty clause $\square$ is ME-deducible from $\mathcal{S}$.

__3.3.4.12__: __Lemma__: If a clause $\mathcal{C}$ is ME-deducible from a set $\mathcal{S}$ of clauses, then there exists a correct plan G for $\mathcal{S}$, such that G is constructed using rules (1)A and (2) only, and $os(G)\theta(G) = \mathcal{C}$.

__Proof__: Let $(K_0,\dots,K_n)$ be an ME-deduction from some matrix set M for $\mathcal{S}$ such that $\mathcal{C} = lit(B(K_n))$. We show now that for each $i \in \{0,\dots,n\}$, there is a correct plan G_i and a function $g_i : K_i \dashrightarrow s(G_i)$ such that:

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

(ii) g_i is injective,

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

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

(iv) $lit(l) = g_i(l)\theta(G_i)$ for all $l \in K_i$,

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

The proof is by induction on i.

__Basis__: K_0 is a variant of a chain in M, and so $\{lit(l) \mid l \in K_0\}$ is a variant of a clause in $\mathcal{S}$. Let G_0 be the basic plan with this top clause, and define:

$$g_0(l) = \mathrm{lit}(l) \text{ for all } l \in K_0$$

Then G_0 and g_0 have the required properties.

<u>Induction</u>: Assume G_{i-1} and g_{i-1} have been constructed. We have three cases to consider.

<u>Case(a)</u>: K_i is derived by contraction from K_{i-1}. We define:

$$G_i = G_{i-1}$$

$$\text{and } g_i = g_{i-1} | K_i$$

Again G_i and g_i have the required properties.

<u>Case(b)</u>: K_i is derived by reduction from K_{i-1}.

$$\text{Let } K_{i-1} = (k_1, \ldots, k_m)$$

$$\text{then } K_i = (k_1, \ldots, k_{m-1})\gamma$$

$$\text{and } B(K_i) = (B(K_{i-1}) - \{(k_m, m)\})\gamma$$

$$\text{where } \gamma = \mathrm{mgu}\{k_m, \neg k_j\}$$

$$\text{and } (k_j, j) \in A(K_{i-1})$$

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

We define g_i by:

$$g_i(k\gamma) = g_{i-1}(k) \quad \text{for all } k\gamma \in K_i$$

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

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

$\{\{g_{i-1}((k_m,m))\Theta(G_{i-1}), \neg g_{i-1}((k_j,j))\Theta(G_{i-1})\}\}$ is unifiable. But by the induction hypothesis, from condition (iv) we have:

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

$$\text{and } g_{i-1}((k_j,j))\Theta(G_{i-1}) = k_j$$

So $C(G_i)$ is unifiable if $\{k_m, \neg k_j\}$ is unifiable; therefore, since $\mathrm{mgu}\{k_m, \neg k_j\} = \gamma$, $C(G_i)$ is unifiable. Therefore G_i is correct and:

$$\Theta(G_i) = \Theta(G_{i-1})\bullet\gamma \quad \text{by lemma 2.3.6}$$

(ii) g_i is clearly injective, since g_{i-1} is injective, by the induction hypothesis.

(iii) $g_i(B(K_i)) = g_i((B(K_{i-1})-\{(k_m,m)\})\gamma)$

$= g_i(B(K_{i-1})\gamma-\{(k_m,m)\gamma\})$

$= g_i(B(K_{i-1})\gamma)-\{g_i((k_m,m)\gamma)\}$

since g_i is injective

$= g_{i-1}(B(K_{i-1}))-\{g_{i-1}((k_m,m))\}$

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

$= os(G_i)$

(iv) Suppose $k\gamma \in K_i$, then $k \in K_{i-1}$ and:

$$\mathrm{lit}(k\gamma) = \mathrm{lit}(k)\gamma$$

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

by condition (iv)

$= g_{i-1}(k)(\theta(G_{i-1})\cdot\gamma)$

$= g_{i-1}(k)\theta(G_i)$

as shown in (i) above.

(v) If $q\gamma \in A(K_i)$ and $q\gamma <(K_i)\, p\gamma$, then $q \in A(K_{i-1})$ and $q <(K_{i-1})\, p$, 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 $g_i(p\gamma)$.

<u>Case</u>(<u>c</u>): K_i is derived by extension from K_{i-1}.

$$
\begin{aligned}
\text{Let } K_{i-1} &= (k_1,\ldots,k_m) \\
\text{then } K_i &= (k_1,\ldots,k_m,l_2,\ldots,l_r)\gamma \\
\text{and } A(K_i) &= (A(K_{i-1}) \cup \{(k_m,m)\})\gamma \\
\text{where } \gamma &= \text{mgu}\{k_m,\neg l_1\} \\
\text{and } K &= (l_1,\ldots,l_r) \text{ is a variant of} \\
&\quad \text{some chain in } M.
\end{aligned}
$$

$(k_m,m) \in B(K_{i-1})$, so by condition (iii) on G_{i-1}, $g_{i-1}((k_m,m)) \in os(G_{i-1})$. Now $\{l_1,\ldots,l_r\}$ is a variant of a clause in $\mathcal{S}$, so we can replace $g_{i-1}((k_m,m))$ through l_1 by $\{l_1,\ldots,l_r\}$ to obtain G_i.

We define g_i is follows:

$$
g_i(k\gamma) = \begin{cases} g_{i-1}(k) & \text{for all } k \in K_{i-1} \\ \text{lit}(k) \in V(G_i)-V(G_{i-1}) & \text{otherwise} \end{cases}
$$

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 $k \in \{(l_2,m+1),\ldots,(l_r,m+r-1)\}$, $g_i(k\gamma)$ 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.

$$C(G_i) = C(G_{i-1}) \cup \{\{g_{i-1}((k_m,m)),\neg l_1\}\}$$

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 $\{g_{i-1}((k_m,m))\theta(G_{i-1}),\neg l_1\theta(G_{i-1})\}$ 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_1 contains only variables which do not occur in G_{i-1}:

$$\neg l_1\theta(G_{i-1}) = \neg l_1$$

Hence $C(G_i)$ is unifiable if $\{k_m,\neg l_1\}$ is unifiable; so since $\mathrm{mgu}\{k_m,\neg l_1\} = \gamma$, $C(G_i)$ is unifiable. Therefore G_i is correct and:

$$\theta(G_i) = \theta(G_{i-1})\cdot\gamma \quad \text{by lemma 2.3.6}$$

(ii) We establish that g_i is injective by making the following observations:

(1) $g_i \mid K_{i-1}\gamma$ is injective since, by the induction hypothesis, g_{i-1} is injective.

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

(3) If $k_1 \in K_{i-1}$ and $k_2 \in \{(l_2,m+1),\ldots,(l_r,m+r-1)\}$

then $g_i(k_1\gamma) \in V(G_{i-1})$

and $g_i(k_2\gamma) \in V(G_i) - V(G_{i-1})$

so $g_i(k_1\gamma) \neq g_i(k_2\gamma)$

(iii) $B(K_i) = B(K_{i-1}) - \{(k_m,m)\}\gamma$

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

$\therefore g_i(B(K_i)) = g_i(B(K_{i-1})\gamma) - g_i(\{(k_m,m)\}\gamma)$

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

since g_i is injective

$= g_{i-1}(B(K_{i-1})) - \{g_{i-1}((k_m,m))\}$

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

$= g_{i-1}(B(K_{i-1})) - \{g_{i-1}((k_m,m))\}$

by the induction hypothesis

$= os(G_i)$

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

In the first case:

$lit(l\gamma) = lit(l)\gamma$

$=(g_{i-1}(l)\theta(G_{i-1}))\gamma$

$= g_{i-1}(l)\theta(G_i)$ as shown above

$= g_i(l\gamma)\theta(G_i)$

In the second case:

$lit(l\gamma) = lit(l)\gamma$

$= (lit(l)\theta(G_{i-1}))\gamma$

since lit(l) contains only variables not in G_{i-1}

$= \text{lit}(l)\theta(G_i)$

$= g_i(l\gamma)\theta(G_i)$

(v) Suppose $q\gamma \in A(K_i)$ and $q\gamma <(K_i)\ p\gamma$. There are two cases to consider.

(a) Suppose $p \in K_{i-1}$, then $q \neq (k_m,m)$, so that $q \in A(K_{i-1})$. Also, $q <(K_{i-1})\ p$ so $g_{i-1}(q)$ is a direct ancestor of $g_{i-1}(p)$. Consequently, $g_i(q\gamma)$ is a direct ancestor of $g_i(p\gamma)$.

(b) Suppose $p \notin K_{i-1}$, then for some s, where $2 \leq s \leq r$, $g_i(p\gamma) = l_s \in os(G_i)-os(G_{i-1})$; 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 $q \in A(K_{i-1})$ 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 $\mathcal{C} = \text{lit}(B(K_n))$

$= \{\text{lit}(l) \mid l \in B(K_n)\}$

$= \{g_n(l)\theta(G_n) \mid l \in B(K_n)\}$

by condition (iv)

$$= \sigma_n(B(K_n))\theta(G_n)$$

$$= os(G_n)\theta(G_n)$$

by condition (iii)

Therefore $G = G_n$ is the required plan.

$\square$

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

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

$\square$

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

<u>Proof</u>: This follows from corollary 3.3.4.13 and lemmas 3.2.18 and 3.2.19.

$\square$

Either of the above corollaries implies the completeness of general plans as follows.

3.3.5: Theorem: Completeness of plans

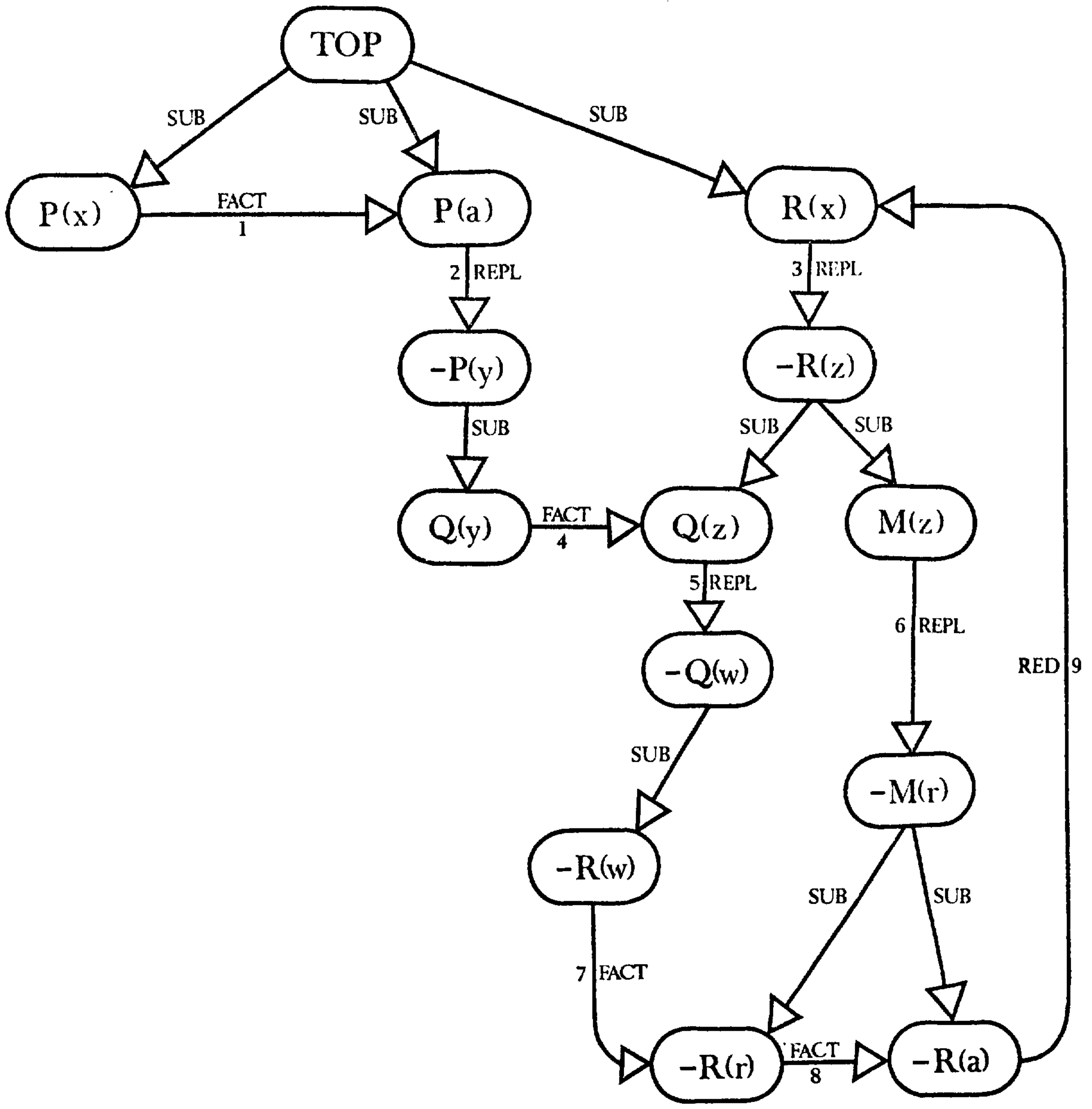
If $\mathcal{S}$ is unsatisfiable, there exists a closed, correct plan for $\mathcal{S}$.

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:

$$\{\ \{P(x),\ P(a),\ R(x)\},$$
$$\{-P(y),\ Q(y)\},$$
$$\{-R(z),\ Q(z),\ M(z)\},$$
$$\{-Q(w),\ -R(w)\},$$
$$\{-M(r),\ -R(r),\ -R(a)\}\ \}$$

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.

<u>Figure 3.11</u>

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 S, S is unsatisfiable if (only if) there exists a closed, correct plan for S 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 {(1)A, (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) -P(x),Q(x)

(3) -P(x),H(x)

(4) -Q(x),K(x)

(5) -Q(x),N(x)

(6) -H(x),K(x)

(7) -H(x),N(x)

(8) -K(x),M(x)

(9) -K(x),S(x)

(10) -N(x),M(x)

(11) -N(x),S(x)

(12) -M(x),-B(x)

(13) -S(x),-B(x)

(14) -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-eliminaticn. The following search is performed, in which A-literals are framed:

(1)	P(x), R(b), R(x)	
(16)	P(b), R(b)	Factoring
(17)	P(b),$\boxed{R(b)}$	Extension with (14)
(18)	P(b)	Contraction
(19)	$\boxed{P(b)}$, Q(b)	Extension with (2)
(20)	$\boxed{P(b)}$,$\boxed{Q(b)}$, K(b)	Extension with (4)
(21)	$\boxed{P(b)}$,$\boxed{Q(b)}$,$\boxed{K(b)}$, M(b)	Extension with (8)
(22)	$\boxed{P(b)}$,$\boxed{Q(b)}$,$\boxed{K(b)}$,$\boxed{M(b)}$, -B(b)	Extension with (12)
	Backtrack to (20)	
(23)	$\boxed{P(b)}$,$\boxed{Q(b)}$,$\boxed{K(b)}$, S(b)	Extension with (9)
.		
.	three backtrackings occur here	
.		
(39)	$\boxed{P(b)}$,$\boxed{H(b)}$,$\boxed{N(b)}$,$\boxed{S(b)}$, -B(b)	Extension with (13)
	Backtrack to (1)	
(40)	P(x), R(b),$\boxed{R(x)}$	Extensicn with (14)
(41)	P(x), R(b)	Contraction
(42)	P(x),$\boxed{R(b)}$	Extension with (14)
(43)	P(x)	Contraction
(44)	$\boxed{P(x)}$, Q(x)	Extension with (2)
(45)	$\boxed{P(x)}$,$\boxed{Q(x)}$, K(x)	Extension with (4)
(46)	$\boxed{P(x)}$,$\boxed{Q(x)}$,$\boxed{K(x)}$, M(x)	Extension with (8)
(47)	$\boxed{P(x)}$,$\boxed{Q(x)}$,$\boxed{K(x)}$,$\boxed{M(x)}$, -B(x)	Extension with (12)

(48) $\boxed{P(a)}$,$\boxed{Q(a)}$,$\boxed{K(a)}$,$\boxed{M(a)}$,$\boxed{-B(a)}$ Extension with (15)

(49)

.

. Contractions

.

(53) □

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.out of all subformulae occurring in C. F.out has the property that for all substitutions θ, θ unifies C if and only if θ unifies F.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.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 af the algorithm is F.out, and S is finally empty.

In his description of the algorithm in [5], Baxter allows F.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 $p \equiv q \bmod F$ 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 ≠ Ø
   do  Delete a constraint {$p_1$,$p_2$} from S;
       if [$p_1$] ≠ [$p_2$]
       then  if [$p_1$] contains a term $f_1(q_{11},\dots,q_{1m})$
                and [$p_2$] contains a term $f_2(q_{21},\dots,q_{2n})$
             then  if $f_1 \neq f_2$
                   then  unification fails;
                         stop
                   else add to S the pairs:
                         {$q_{11}$,$q_{21}$},...,{$q_{1n}$,$q_{2n}$}
             Replace [$p_1$] and [$p_2$] by [$p_1$] ∪ [$p_2$] 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 $t_1 \equiv t_2$ mod F.out, where $t_1 = f(q_{11},\dots,q_{1n})$ and $t_2 = f(q_{21},\dots,q_{2n})$ then for each $i \in \{1,\dots,n\}$, $q_{1i} \equiv q_{2i}$ mod F.out.

Both these results are proved in [5].

4.1.3: Sorting stage

To determine whether or not F.out is unifiable, it is necessary to construct a certain directed graph whose vertex set is F.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.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 TRANSFORM(C) succeeds, returning F.out, then D(C) is a digraph, where $V(D(C)) = F.out$, and $E(D(C))$ is defined as follows. Suppose there are m classes of F.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.out, D(C) is unique. This follows from the fact that if a class of F.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.

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: <u>Lemma</u>: 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: <u>Theorem</u>: 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 <u>source</u> 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

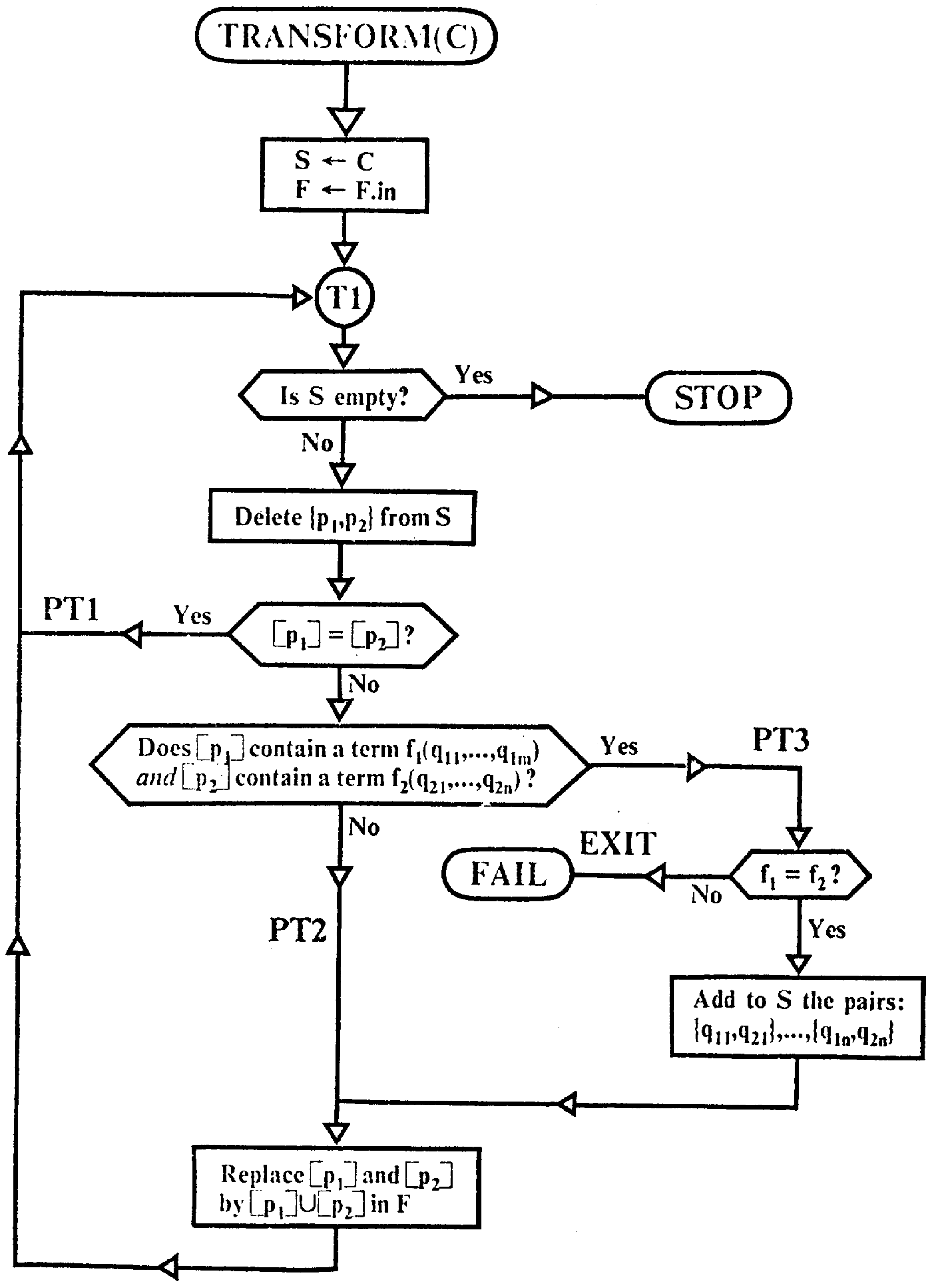
As with TRANSFORM, the input to CLASSIFY is C, the set of constraints to be unified. When the algorithm halts, it returns a partition F.out of the subformulae of C, and a partition P.out of the subformulae of C which are terms. Recall that a term is a formula that is not a variable (2.2.6). CLASSIFY 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 P or entire classes of P 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 P.

4.2.1.1: Definition: If t is a term which is a subformula of C, we denote by $\langle t\rangle_P$, the class in P containing t, and write $t_1 \equiv t_2 \bmod P$ if and only if $\langle t_1\rangle_P = \langle t_2\rangle_P$. When no ambiguity is likely we will write $\langle t\rangle$ for $\langle t\rangle_P$. 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 ≠ Ø
   do  Delete a constraint {p1,p2} from S;
       if [p1] ≠ [p2]
       then  T <-- [p1];
             while T contains a term t1 = f(q11,...,q1n)
                do  Delete from T all terms in <t1>;
                    if [p2] contains a term
                          t2 = f(q21,...,q2n)
                    then  Add to S the pairs:
                             {q11,q21},...,{q1n,q2n};
                          Replace <t1> and <t2> by
                             <t1>U <t2> in P
             Replace [p1] and [p2] by [p1]U [p2] in F
stop
```

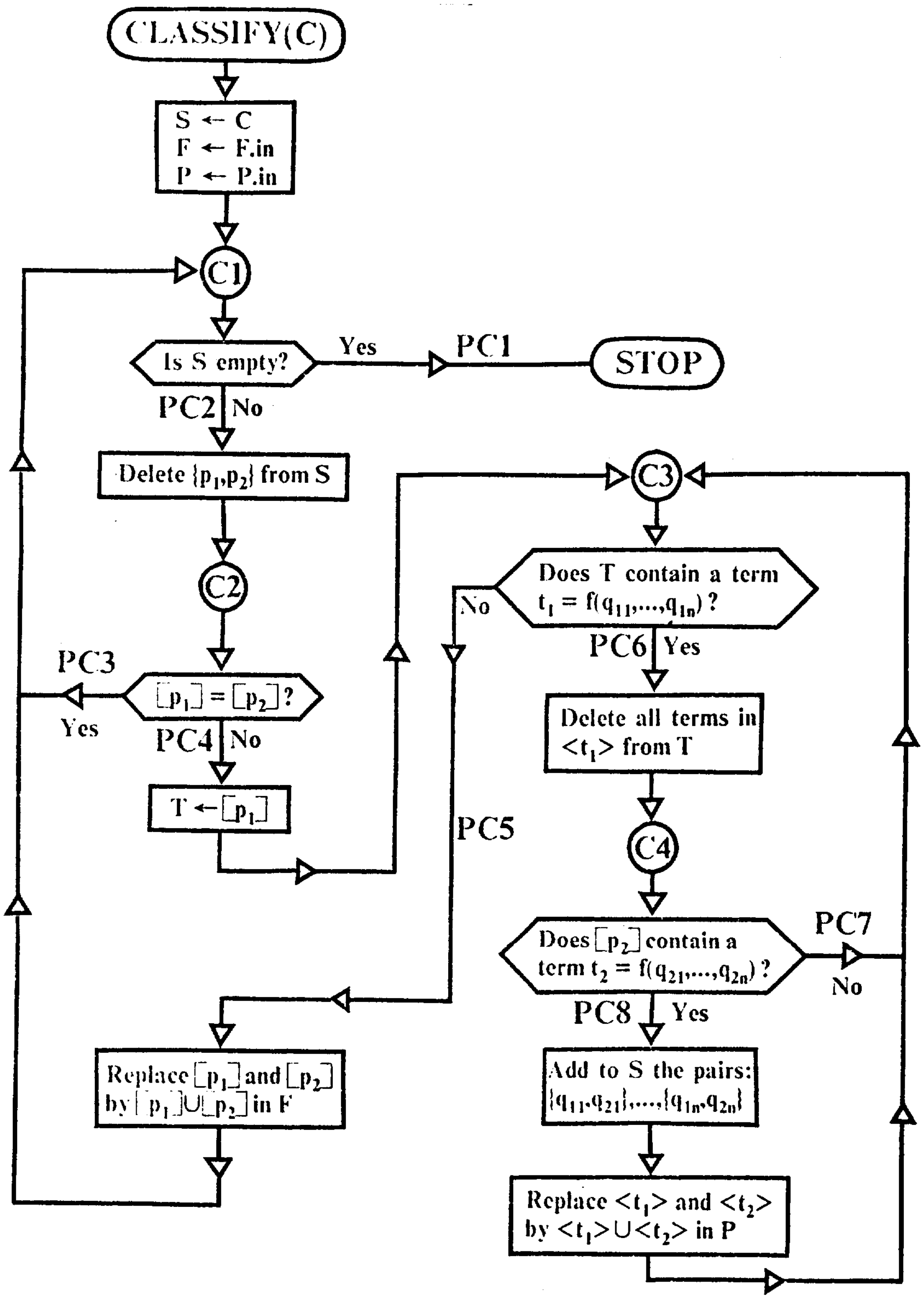
In the remainder of section 4.2.1, we prove several properties of the algorithm CLASSIFY, and in particular, we show precisely how TRANSFORM and CLASSIFY are related. These proofs refer to the flowcharts of the algorithms, shown in figures 4.1 and 4.2. In the flowchart for CLASSIFY, each decision box 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.



TRANSFORM(C)
$S \leftarrow C$
$F \leftarrow F.in$
T1
Is S empty?
Yes
STOP
No
Delete $\{p_1,p_2\}$ from S
PT1
Yes
$[p_1] = [p_2]$?
No
Does $[p_1]$ contain a term $f_1(q_{11},\ldots,q_{1m})$ *and* $[p_2]$ contain a term $f_2(q_{21},\ldots,q_{2n})$?
Yes
PT3
No
EXIT
FAIL
No
$f_1 = f_2$?
Yes
PT2
Add to S the pairs: $\{q_{11},q_{21}\},\ldots,\{q_{1n},q_{2n}\}$
Replace $[p_1]$ and $[p_2]$ by $[p_1] \cup [p_2]$ in F

The flowchart for the algorithm TRANSFORM.

Figure 4.1



CLASSIFY(C)
S ← C
F ← F.in
P ← P.in
C1
Is S empty?
Yes
PC1
STOP
PC2
No
Delete {p_1,p_2} from S
C2
PC3
[p_1] = [p_2]?
Yes
PC4
No
T ← [p_1]
C3
Does T contain a term $t_1 = f(q_{11},...,q_{1n})$?
No
PC6
Yes
Delete all terms in <t_1> from T
PC5
C4
Does [p_2] contain a term $t_2 = f(q_{21},...,q_{2n})$?
PC7
No
PC8
Yes
Replace [p_1] and [p_2] by [p_1]∪[p_2] in F
Add to S the pairs: {q_{11},q_{21}},...,{q_{1n},q_{2n}}
Replace <t_1> and <t_2> by <t_1>∪<t_2> in P

The flowchart for the algorithm CLASSIFY.

<u>Figure 4.2</u>

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 C1 in the flowchart. This involves investigating the two paths PC3 and PC5 from C1 to C1, showing that if the assertion holds at C1 at the beginning of such a loop, it again holds on returning to C1. To complete the induction, H must be shown to hold at the first encounter with C1 in the execution of CLASSIFY. The path PC5 includes a loop from C3 to C3, so in general, we must establish the invariance of some other assertion at C3 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 CLASSIFY(C), where C is the set of constraints:

$$\begin{aligned}\{\quad &\{F(x,x),\ v\},\\ &\{v,\ F(f(a),h(y))\},\\ &\{F(u,f(y)),\ F(h(g(b)),u)\}\\ &\{h(u),\ h(f(a))\}\quad\}\end{aligned}$$

and a and b are constants. The set of subformulae of C is partitioned into six classes:

$$F.out = \{ \{F(x,x), F(f(a),h(y)), v\}, \{F(u,f(y)), F(h(g(b)),u)\}, \{h(u), h(f(a))\}, \{x, u, f(a), f(y), h(y), h(g(b))\}, \{a, y, g(b)\}, \{b\} \}$$

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

$$P.out = \{ \{F(x,x), F(f(a),h(y))\}, \{F(u,f(y)), F(h(g(b)),u)\}, \{h(u), h(f(a))\}, \{f(a), f(y)\}, \{h(y), h(g(b))\}, \{a\}, \{g(b)\}, \{b\} \}$$

Introduced	Removed	S	F.in=F(C1,1) P.in=P(C1,1)	F(C1,2) P(C1,2)	F(C1,3) P(C1,3)	F(C1,4) P(C1,4)	F(C1,5) F(C1,5)
			F(x,x)	*F(x,x)	F(x,x)	F(x,x)	F(x,x)
			x	v	F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))
			v	x	v	v	v
			F(f(a),h(y))	*F(f(a),h(y))	x	F(u,f(y))	F(u,f(y))
1	2	{F(x,x), v}	f(a)	f(a)	f(a)	F(h(g(b)),u)	F(h(g(b)),u)
1	3	{v, F(f(a),h(y))}	a	a	a	f(a)	h(u)
1	4	{F(u,f(y)), F(h(g(b),u))}	h(y)	h(y)	h(y)	x	h(f(a))
1	5	{h(u), h(f(a))}	y	y	y	a	x
3	6	{x, f(a)}	F(u,f(y))	F(u,f(y))	*F(u,f(y))	h(y)	f(a)
3	7	{x, h(y)}	u	u	u	y	a
4	8	{u, h(g(b))}	f(y)	f(y)	f(y)	u	h(y)
4	9	{f(y), u}	F(h(g(b)),u)	F(h(g(b)),u)	*F(h(g(b)),u)	f(y)	y
5	10	{u, f(a)}	h(g(b))	h(g(b))	h(g(b))	h(g(b))	u
10	11	{y, a}	g(b)	g(b)	g(b)	g(b)	f(y)
10	12	{y, g(b)}	b	b	b	b	h(g(b))
			h(u)	h(u)	h(u)	*h(u)	g(b)
			h(f(a))	h(f(a))	h(f(a))	*h(f(a))	b

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 C1 in the flowchart at which the corresponding constraint first appears: similarly, the "Removed" column indicates the encounter with C1 at which the constraint is first absent. (Continued on page 108.)

<u>Figure 4.3</u>

F(C1,6) P(C1,6)	F(C1,7) P(C1,7)	F(C1,8) P(C1,8)	F(C1,9) P(C1,9)	F(C1,10) P(C1,10)	F(C1,11) P(C1,11)	F.out=F(C1,12) P.out=P(C1,12)
F(x,x)	F(x,x)	F(x,x)	F(x,x)	F(x,x)	F(x,x)	F(x,x)
F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))	F(f(a),h(y))
v	v	v	v	v	v	v
F(u,f(y))	F(u,f(y))	F(u,f(y))	F(u,f(y))	F(u,f(y))	F(u,f(y))	F(u,f(y))
F(h(g(b)),u)	F(h(g(b)),u)	F(h(g(b)),u)	F(h(g(b)),u)	F(h(g(b)),u)	F(h(g(b)),u)	F(h(g(b)),u)
h(u)	h(u)	h(u)	h(u)	h(u)	h(u)	h(u)
h(f(a))	h(f(a))	h(f(a))	h(f(a))	h(f(a))	h(f(a))	h(f(a))
x	x	x	x	x	x	x
f(a)	f(a)	f(a)	*f(a)	u	u	u
a	h(y)	h(y)	*h(y)	f(a)	f(a)	f(a)
h(y)	a	a	a	f(y)	f(y)	f(y)
y	y	y	y	h(y)	h(y)	h(y)
u	u	u	u	h(g(b))	h(g(b))	h(g(b))
f(y)	f(y)	h(g(b))	*h(g(b))	a	a	a
h(g(b))	h(g(b))	f(y)	*f(y)	y	y	y
g(b)	g(b)	g(b)	g(b)	g(b)	g(b)	g(b)
b	b	b	b	b	b	b

The narrow columns represent the partitions F and P at successive encounters with C1. 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 (contd.)

<u>4.2.1.3</u>: <u>Definition</u>: 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.

<u>4.2.1.4</u>: <u>Lemma</u>: CLASSIFY(C) halts for every set of constraints C.

<u>Proof</u>: We refer to the flowchart for CLASSIFY in figure 4.2.

First we show that S is always finite at C1, and that every path from C1 to C1 is executed in finite time. Since S is initially finite at C1, having been initialised to C, we need only investigate each path from C1 to C1, 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 C1 via PC3.

$$S(C1,2) = S(C1,1) - \{\{p_1,p_2\}\},$$

so S(C1,2) is finite if S(C1,1) is finite.

(2) Now consider the path returning to C1 via PC5. We observe that for each i, if S(C3,i) is finite then S(C3,i+1) is also finite, and that T(C3,i+1) contains less terms than T(C3,i). But T(C3,1) = [p_1], which contains a finite number of terms, and S(C3,1) is finite. Therefore, for some integer n, S(C3,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.

$\square$

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: $\langle s_1 \rangle = \langle s_2 \rangle$ if and only if s_1 and s_2 begin with the same function symbol and $[s_1] = [s_2]$.

Proof: Let H(P,F) be the assertion:

"$\langle s_1 \rangle = \langle s_2 \rangle$ iff s_1 and s_2 begin with the same function symbol, and $[s_1] = [s_2]$"

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

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

$$P(C1,2) = P(C1,1)$$

$$F(C1,2) = F(C1,1)$$

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:

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

<u>then</u> $\langle s_1\rangle = \langle s_2\rangle$"

B(P) asserts:

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

<u>then</u> either $s_1 \in T$ or $\langle s_1\rangle = \langle s_2\rangle$"

D(P) asserts:

"<u>if</u> $\langle s_1\rangle = \langle s_2\rangle$

<u>then</u> s_1 and s_2 begin with the same function symbol

and either $[s_1] = [s_2]$

or $([s_1] = [p_1]$ and $[s_2] = [p_2])$

or $([s_1] = [p_2]$ 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 $[s_1] = [s_2]$ and that s_1 and s_2 begin with the same function symbol, then:

$$s_1 \equiv s_2 \bmod P(C3,1) \text{ by } A(P(C3,1))$$

$$\text{But } P(C3,1) \leq P(C3,2)$$

$$\therefore s_1 \equiv s_2 \bmod P(C3,2)$$

$\therefore$ 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.

<u>case</u> (<u>i</u>) If $s_1 \in$ T(C3,1), then $s_1 \equiv t_1$ mod P(C3,1), so by D(P(C3,1)), s_1 and t_1 begin with the same function symbol. Also $[s_2] = [p_2]$, and s_2 begins with the same function symbol as s_1 (and hence t_1), 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).

<u>case</u> (<u>ii</u>) 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.

<u>case</u> (<u>i</u>) $s_1 \equiv s_2$ 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 ($[s_1] = [p_1]$ and $[s_2] = [p_2]$)

or ($[s_1] = [p_2]$ and $[s_2] = [p_1]$)

<u>case</u> (<u>ii</u>) $s_1 \equiv t_1 \bmod P(C3,1)$

and $s_2 \equiv t_2 \bmod P(C3,1)$

So by D(P(C3,1)), s_1 and t_1 begin with the same function symbol. But t_1 and t_2 begin with the same function symbol, so s_1 and s_2 begin with the same function symbol. Also by D(P(C3,1)):

$$\left[\begin{array}{l} \quad [s_1] = [t_1] \\ \text{or } ([s_1] = [p_1] \text{ and } [t_1] = [p_2]) \\ \text{or } ([s_1] = [p_2] \text{ and } [t_1] = [p_1]) \end{array}\right]$$

and

$$\left[\begin{array}{l} \quad [s_2] = [t_2] \\ \text{or } ([s_2] = [p_1] \text{ and } [t_2] = [p_2]) \\ \text{or } ([s_2] = [p_2] \text{ and } [t_2] = [p_1]) \end{array}\right]$$

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

$$\left[\begin{array}{l} \quad [s_1] = [p_1] \\ \text{or } ([s_1] = [p_2] \text{ and } [t_1] = [p_2]) \end{array}\right]$$

and

$$\left[\begin{array}{l} \quad [s_2] = [p_2] \\ \text{or } ([s_2] = [p_1] \text{ and } [t_2] = [p_2]) \end{array}\right]$$

From this we can deduce:

$[s_1] = [s_2]$

or ($[s_1] = [p_1]$ and $[s_2] = [p_2]$)

or ($[s_1] = [p_2]$ and $[s_2] = [p_1]$)

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 m times on this path. We assume H(P(C1,1),F(C1,1)) holds, so that $s_1 \equiv s_2$ mod P(C1,1) if and only if both $s_1 \equiv s_2$ mod F(C1,1) and s_1 and s_2 begin with the same function symbol.

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

P(C3,1) = P(C1,1)

and T(C3,1) = $[p_1]$

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) − $\{[p_1],[p_2]\} \cup \{[p_1] \cup [p_2]\}$

and P(C1,2) = P(C3,m)

(A) Suppose that $s_1 \equiv s_2$ mod F(C1,2) and that s_1 and s_2 begin with the same function symbol, then:

<u>either</u> $s_1 \equiv s_2$ mod F(C1,1)

so by A(P(C3,m)) we have:

$s_1 \equiv s_2$ mod P(C3,m)

$\therefore$ $s_1 \equiv s_2$ mod P(C1,2)

<u>or</u> $s_1 \equiv p_1$ mod F(C1,1)

and $s_2 \equiv p_2$ mod F(C1,1)

but by B(P(C3,m) we have:

$s_1 \in$ T(C3,m)

or $s_1 \equiv p_2 \bmod P(C3,m)$

However, since C3 is encountered only m times, $T(C3,m)$ contains no terms.

$\therefore s_1 \equiv s_2 \bmod P(C3,m)$

$\therefore s_1 \equiv s_2 \bmod P(C1,2)$

(B) Suppose that $s_1 \equiv s_2 \bmod P(C1,2)$

then $s_1 \equiv s_2 \bmod P(C3,m)$

so by $D(P(C3,m))$, s_1 and s_2 begin with the same function symbol and:

<u>either</u> $s_1 \equiv s_2 \bmod F(C1,1)$

so $s_1 \equiv s_2 \bmod F(C1,2)$ since $F(C1,2) \geq F(C1,1)$

<u>or</u> $s_1 \equiv p_1 \bmod F(c1,1)$

and $s_2 \equiv p_2 \bmod F(C1,1)$

$\therefore s_1 \equiv p_1 \bmod F(C1,2)$

and $s_2 \equiv p_2 \bmod F(C1,2)$ since $F(C1,2) \geq F(C1,1)$

but $p_1 \equiv p_2 \bmod F(C1,2)$

$\therefore s_1 \equiv s_2 \bmod F(C1,2)$

<u>or</u> $s_1 \equiv p_2 \bmod F(C1,1)$

and $s_2 \equiv p_1 \bmod F(C1,1)$

so by the same reasoning as in the second alternative:

$s_1 \equiv s_2 \bmod F(C1,2)$

So for the path from C1 to C1 via PC5, $H(P(C1,2),F(C1,2))$ 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.

<u>4.2.1.6</u>: <u>Lemma</u>:

(a) If TRANSFORM(C) succeeds returning partition F.out, then there is an execution of CLASSIFY(C) returning partition F.out, where in each class of F.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.out, where some class of F.out contains terms beginning with different function symbols.

<u>Proof</u>:

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

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

<u>and</u> 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 $[p_1]$ contains no terms, then path PC5 is followed to C1, where:

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

(2) Suppose $[p_1]$ contains a term t_1, so that by (iii), $[p_2]$ contains no terms. Path PC6 is followed to C4, where:

$$T(C4,1) = T(C3,1) - \langle t_1 \rangle$$
$$= [p_1] - \langle t_1 \rangle$$

But by $H(F(T1,1),S(T1,1))$, all terms in $[p_1]$ begin with the same function symbol, so by lemma 4.2.1.5, all terms in $[p_1]$ are in $\langle t_1 \rangle$, 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:

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

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

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

$$\text{and} \quad S(C1,1) = S(T1,1)$$

So in both cases:

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

$$\text{and} \quad S(C1,2) = S(T1,2)$$

Hence $H(F(T1,2),S(T1,2))$ holds for this path.

<u>Path PT3</u>: On returning to T1, we have:

$$F(T1,2) = F(T1,1) - \{[p_1],[p_2]\} \cup \{[p_1] \cup [p_2]\}$$

$$\text{and } S(T1,2) = S(T1,1) - \{\{p_1,p_2\}\}$$
$$\cup \{\{q_{11},q_{21}\},\ldots,\{q_{1n},q_{2n}\}\}$$

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

(i) $S(T1,1) \neq \emptyset$

(ii) $[p_1] \neq [p_2]$

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

$t_1 = f_1(q_{11},\ldots,q_{1m})$, and $t_2 = f_2(q_{21},\ldots,q_{2n})$

(iv) $f_1 = f_2$

Every class of $F(T1,2)$ is either a class of $F(T1,1)$, in which case all terms in it begin with the same function symbol, by $H(F(T1,1),S(T1,1))$; or it is the union of classes $[p_1]$ and $[p_2]$ of $F(T1,1)$. 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 $H(F(T1,1),S(T1,1))$: 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 (i) 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:

$$\begin{aligned} S(C1,2) &= S(C3,2) \\ &= S(C1,1) - \{\{p_1,p_2\}\} \\ &\qquad \cup \{\{q_{11},q_{21}\},\ldots,\{q_{1n},q_{2n}\}\} \end{aligned}$$

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

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

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

$$\text{and } S(C1,1) = S(T1,1)$$

Therefore:

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

$$\text{and } S(C1,2) = S(T1,2)$$

Hence H(F(T1,2),S(T1,2)) holds for this path.

This proves the invariance of H(F,S) at T1, so since H(F,S) is initially true at T1, H(F,S) always holds at T1. Consequently if TRANSFORM(C) succeeds, returning F.out, then there exists an execution of CLASSIFY(C) which encounters C1 with values F.out and Ø for F and S respectively, and the result is proved.

(b) Now suppose TRANSFORM(C) fails. By part (a), there exists an execution of CLASSIFY(C) which encounters C1 with the same values of S and F that obtain at the last encounter with T1 in the execution of TRANSFORM(C). Since TRANSFORM(C) fails, the path EXIT must be followed to the point FAIL, so that $[p_1]$ and $[p_2]$ contain terms t_1 and t_2 respectively, which begin with different function symbols. Since $[p_1] \neq [p_2]$, CLASSIFY can follow the locp returning to C1 via PC5, and as a result:

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

Hence F(C1,2) contains a class in which two terms, namely t_1 and t_2, begin with different function symbols, and since $F.out \geq F(C1,2)$, the same can be said for F.out.

$\Box$

We now prove another property of 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:

<u>4.2.1.7</u>: <u>Lemma</u>: Suppose the output partitions of CLASSIFY are F.out and P.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.out, then for each $i \in \{1,\ldots,m\}$, $p_{1i} \equiv p_{2i}$ mod F.out.

<u>Proof</u>: Let H(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:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \text{ mod F for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S \text{ for all } j \in \{1,\ldots,k-1\}$$ "

We will ultimately show that H(F,P,S) always holds at C1; first, however, we show that if H(F,P,S $\cup \{\{p_1,p_2\}\}$) holds at C3, then it again holds at C3 after the loop PC6 has been traversed.

<u>Path PC7</u>: On this path, values of F, P, and S are unchanged so the result holds.

<u>Path PC8</u>: In this case:

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

$$P(C3,2) = P(C3,1) - \{\langle t_1\rangle,\langle t_2\rangle\} \cup \{\langle t_1\rangle \cup \langle t_2\rangle\}$$

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

Now if $s_1 \equiv s_2$ mod P(C3,2) then:

<u>either</u> $s_1 \equiv s_2$ mod P(C3,2),

so the result holds in view of the fact that $F(C3,2) = F(C3,1)$ and $S(C3,1) \subseteq S(C3,2)$

<u>or</u> $s_1 \equiv t_1 \mod P(C3,1)$

$s_2 \equiv t_2 \mod P(C3,1)$,

so $f=g$ and $n=m$, and by hypothesis, for each $i \in \{1,\dots,m\}$:

(a) there exists $k_1 \geq 1$, $r_1,\dots,r_{k_1},w_1,\dots,w_{k_1}$ such that:

$$p_{1i} = r_1$$

$$q_{1i} = w_{k_1}$$

$$r_j \equiv w_j \mod F(C3,1) \quad \text{for all } j \in \{1,\dots,k_1\}$$

$$\{w_j, r_{j+1}\} \in S(C3,1) \cup \{\{p_1,p_2\}\} \quad \text{for all } j \in \{1,\dots,k_1-1\}$$

(b) there exists $k_2 \geq 1$, $r'_1,\dots,r'_{k_2}, w'_1,\dots,w'_{k_2}$ such that:

$$p_{2i} = r'_1$$

$$q_{2i} = w'_{k_2}$$

$$r'_j \equiv w'_j \mod F(C3,1) \quad \text{for all } j \in \{1,\dots,k_2\}$$

$$\{w'_j, r'_{j+1}\} \in S(C3,1) \cup \{\{p_1,p_2\}\} \quad \text{for all } j \in \{1,\dots,k_2-1\}$$

But $\{q_{1i}, q_{2i}\} \in S(C3,2)$

$S(C3,1) \subseteq S(C3,2)$

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

Now if we let $k=k_1+k_2$, rename $r'_1,\dots,r'_{k_2}$ as $w_k,\dots,w_{k_1+1}$ and rename $w'_1,\dots,w'_{k_2}$ as $r_k,\dots,r_{k_1+1}$, then:

$\exists\, k \geq 1$, $r_1,\dots,r_k,w_1,\dots,w_k$ such that:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F(C3,2) \quad \text{for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S(C3,2) \cup \{\{p_1,p_2\}\}$$

$$\text{for all } j \in \{1,\ldots k-1\}$$

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, $P = P.in$ and $s_1 \equiv s_2 \bmod P.in$ implies $s_1 = s_2$, since each class in P.in contains only one formula. Therefore, for each i, $p_{1i} = p_{2i}$, so that $p_{1i} \equiv p_{2i} \bmod F.in$, 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:

$$F(C1,2) = F(C1,1)$$

$$P(C1,2) = P(C1,1)$$

$$\text{and } S(C1,2) = S(C1,1) - \{\{p_1,p_2\}\}$$

$$\text{where } p_1 \equiv p_2 \bmod F(C1,1)$$

Suppose $s_1 \equiv s_2 \bmod P(C1,2)$

then $s_1 \equiv s_2 \bmod P(C1,1)$

so by $H(F(C1,1),P(C1,1),S(C1,1))$, there exists an integer $k \geq 1$, and formulae $r_1,\ldots,r_k, w_1,\ldots,w_k$ such that:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F(C1,1) \text{ for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S(C1,1) \text{ for all } j \in \{1,\ldots,k-1\}$$

which is equivalent to:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F(C1,2) \text{ for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S(C1,2) \cup \{\{p_1,p_2\}\}$$
$$\text{for all } j \in \{1,\ldots,k\}$$

where $p_1 \equiv p_2 \bmod F(C1,2)$

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:

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

$$P(C1,2) = P(C3,l)$$

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

where C3 is encountered l times on the path.

Now $H(F(C1,1),P(C1,1),S(C1,1))$ holds, by hypothesis, and

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

$$P(C3,1) = P(C3,1)$$

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

so that $H(F(C3,1),P(C3,1),S(C3,1) \cup \{\{p_1,p_2\}\})$ holds.

But $H(F,P,S \cup \{\{p_1,p_2\}\})$ has been proved invariant at C3, so that $H(F(C3,l),P(C3,l),S(C3,l) \cup \{\{p_1,p_2\}\})$ holds.

Suppose $s_1 \equiv s_2 \bmod P(C1,2)$

then $s_1 \equiv s_2 \bmod P(C3,l)$

and therefore, for each i, there exists $k \geq 1$, $r_1,\ldots,r_k$, $w_1,\ldots,w_k$ such that:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F(C3,l) \text{ for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S(C3,l) \cup \{\{p_1,p_2\}\}$$

$$\text{for all } j \in \{1,\ldots,k-1\}$$

Since $F(C1,2) \geq F(C3,l)$, this is equivalent to:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F(C1,2) \text{ for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in S(C1,2) \cup \{\{p_1,p_2\}\}$$

$$\text{for all } j \in \{1,\ldots,k-1\}$$

where $p_1 \equiv p_2 \bmod F(C1,2)$

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 $r_{j_1} \equiv w_{j_1} \equiv r_{j_1+1} \equiv w_{j_2} \equiv r_{j_2+1} \equiv w_{j_2+1} \bmod F(C1,2)$. So if we let $k_1 = k-j_2+j_1-1$; rename $w_{j_2+1},\ldots,w_k$ as $w_{j_1},\ldots,w_{k_1}$; and rename $r_{j_2+2},\ldots,r_k$ as $r_{j_1+1},\ldots,r_{k_1}$, we then have $k_1 \geq 1$ since $1 \leq j_1 \leq j_2 < k$, so that:

$\exists\ k_1 \geq 1,\ r_1,\ldots,r_{k_1},\ w_1,\ldots,w_{k_1}$ such that:

$$p_{1i} = r_1$$

$$p_{2i} = w_{k_1}$$

$$r_j \equiv w_j \bmod F(C1,2) \text{ for all } j \in \{1,\ldots,k_1\}$$

$$\{w_j, r_{j+1}\} \in S(C1,2) \text{ for all } j \in \{1,\ldots,k_1-1\}$$

which is the required result.

Therefore $H(F,P,S)$ holds at C1, and in particular $H(F.out,P.out,\emptyset)$, so:

If $s_1 \equiv s_2$ mod P.out, then for each $i \in \{1,\ldots,n\}$ there exists $k\geq1$, and $r_1,\ldots,r_k,w_1,\ldots,w_k$ such that:

$$p_{1i} = r_1$$

$$p_{2i} = w_k$$

$$r_j \equiv w_j \bmod F.out \text{ for all } j \in \{1,\ldots,k\}$$

$$\{w_j, r_{j+1}\} \in \emptyset \text{ for all } j \in \{1,\ldots,k-1\}$$

In view of this last result, k=1, and therefore $p_1 \equiv p_2$ mod F.out.

□

4.2.2: The Automaton for a Constraint Set

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 M(C) the set of all function symbols occurring in C. We then define:

$$\text{degree}(C) = \max_{f \in M(C)} \text{degree}(f)$$

$$\text{and } N(C) = \{i \mid 1 \leq i \leq \text{degree}(C)\}$$

4.2.2.2: Definition: If C is a set of constraints, then A(C) is a labelled, directed graph, where:

$$V(A(C)) = \{p \mid p \text{ is a subformula of } C\}$$

$$I(A(C)) = C \cup (M(C) \times N(C))$$

$$E(A(C)) = \text{TRANS}(A(C)) \cup \text{PUSH}(A(C)) \cup \text{POP}(A(C))$$

where TRANS(A(C)), PUSH(A(C)) and POP(A(C)) are mutually disjoint sets of arcs defined by:

$$\text{TRANS}(A(C)) = \{(p_1, \phi, p_2) \mid \{p_1, p_2\} = \phi \in C\}$$

$$\text{PUSH}(A(C)) = \{(p,(f,i),t) \mid p \text{ and } t \text{ are subformulae of } C,\ t=f(p_1,\ldots,p_n),\text{ and } p=p_i \text{ for some } i \in \{1,\ldots,n\}\}$$

$$\text{POP}(A(C)) = \{(t,(f,i),p) \mid p \text{ and } t \text{ are subformulae of } C,\ t=f(p_1,\ldots,p_n),\text{ and } p=p_i \text{ for some } i \in \{1,\ldots,n\}\}$$

If $e = (p_1, \text{label}, p_2) \in E(A(C))$, we denote by e^{-1} the ordered triple (p_2, label, p_1). Then $(e^{-1})^{-1}=e$; $e \in TRANS(A(C))$ if and only if $e^{-1} \in TRANS(A(C))$; and $e \in PUSH(A(C))$ if and only if $e^{-1} \in POP(A(C))$.

Note that A(C) can be regarded as a nondeterministic, finite, pushdown automaton [1], where V(A(C)) is the set of <u>states</u>, (which we will also refer to as vertices of A(C), and subformulae 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; <u>input alphabet</u> C; and <u>pushdown alphabet</u> M(C) X N(C), which we will henceforth refer to as Z. Accordingly, we call A(C) the <u>automaton for C</u>, and make the following definitions.

<u>4.2.2.3</u>: <u>Definition</u>: If X is any finite set, <u>a word of length n over X</u>, 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.

<u>4.2.2.4</u>: <u>Definition</u>: If $a \in C^*$, $\gamma \in Z^*$, and $p \in V(A(C))$, then (p,a,γ) is called a <u>configuration of A(C)</u>, and p, a and γ are called the <u>state</u>, <u>input</u> and <u>stack</u> of the configuration, respectively.

4.2.2.5: Definition: If $e \in E(A(C))$, we define a relation $\vdash e \dashv$ on the set of configurations of A(C) as follows: let $e = (p_1, \text{label}, p_2)$, then $(q_1, a_1, \gamma_1) \vdash e \dashv (q_2, a_2, \gamma_2)$ if and only if:

(i) $q_1 = p_1$, $q_2 = p_2$

and (ii) (a) if $e \in TRANS(A(C))$, and label $= \not{c} \in C$

then $a_1 = \not{c} a_2$

$\gamma_1 = \gamma_2$

(b) if $e \in PUSH(A(C))$, and label $= (f,i) \in Z$

then $a_1 = a_2$

$\gamma_2 = (f,i)\gamma_1$

(c) if $e \in POP(A(C))$, 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, γ_1) to $(p_{n+1}, a_{n+1}, \gamma_{n+1})$, if and only if:

$(p_i, a_i, \gamma_i) \vdash e_i \dashv (p_{i+1}, a_{i+1}, \gamma_{i+1})$ for all $i \in \{1, \ldots, n\}$

4.2.2.7: Definition: For each integer $n \geq 0$, we define a relation $\vdash^n$ on the set of configurations of A(C) as follows:

(i) $(p_1, a_1, \gamma_1) \vdash^0 (p_2, a_2, \gamma_2)$ if and only if $p_1 = p_2$, $a_1 = a_2$, and $\gamma_1 = \gamma_2$

(ii) If $n>0$, $(p_1,a_1,\gamma_1) \vdash^n (p_2,a_2,\gamma_2)$ if and only if there is a chain of length n in A(C) from (p_1,a_1,γ_1) to (p_2,a_2,γ_2)

We abbreviate $\vdash^1$ as $\vdash$, and also define the relation $\vdash^*$ on the set of configurations of A(C) as follows:

$$(p_1,a_1,\gamma_1) \vdash^* (p_2,a_2,\gamma_2)$$
$$\text{iff } (p_1,a_1,\gamma_1) \vdash^n (p_2,a_2,\gamma_2) \text{ for some } n\geq 0$$

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.

4.2.2.9: Lemma: If $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is a chain in A(C) such that $a_{n+1}=\emptyset$, then there exists a chain $(p_{n+1},b_{n+1},\gamma_{n+1}),e_n^{-1},\ldots \quad ,e_1^{-1},(p_1,b_1,\gamma_1)$ where $b_1=\emptyset$ and $b_{n+1}=\hat{a}_1$. We define $\hat{a}$ as the word obtained by arranging the elements of a in reverse order.

Proof: By induction on n.

Basis: $n=1$, We have three cases to consider:

(a) $e_1 = (p_1,\emptyset,p_2) \in \text{TRANS}(A(C))$.

In this case $a_1=\not{c}$, $\gamma_1=\gamma_2$ and

$e_1^{-1} = (p_2,\not{c},p_1) \in$ TRANS(A(C))

Clearly $(p_2,\not{c},\gamma_2) \vdash e_1^{-1} \dashv (p_1,\emptyset,\gamma_2)$, so that

$(p_2,\hat{a}_1,\gamma_2)$, e_1^{-1}, $(p_1,\emptyset,\gamma_1)$ is a chain in A(C).

(b) $e_1 = (p_1,(f,i),p_2) \in$ PUSH(A(C)).

In this case $a_1=a_2=\emptyset$, $\gamma_2=(f,i)\gamma_1$ and

$e_1^{-1} = (p_2,(f,i),p_1) \in$ POP(A(C))

So therefore $(p_2,\hat{a}_1,\gamma_2) \vdash e_1^{-1} \dashv (p_1,\hat{a}_1,\gamma_1)$, and hence

$(p_2,\hat{a}_1,\gamma_2)$, e_1^{-1}, $(p_1,\emptyset,\gamma_1)$ is a chain in A(C).

(c) $e_1 = (p_1,(f,i),p_2) \in$ POP(A(C)).

In this case $a_1=a_2=\emptyset$, $\gamma_1=(f,i)\gamma_2$, and

$e_1^{-1} = (p_2,(f,i),p_1) \in$ PUSH(A(C)).

Therefore, $(p_2,\hat{a}_1,\gamma_2) \vdash e_1^{-1} \dashv (p_1,\hat{a}_1,\gamma_1)$, so that

$(p_2,\hat{a}_1,\gamma_2)$, e_1^{-1}, $(p_1,\emptyset,\gamma_1)$ is a chain in A(C)

<u>Induction</u>: Assume the result holds for chains of length $<n$.

Now $(p_2,a_2,\gamma_2),e_2, \ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is a chain of the given form of length n-1, so by the induction hypothesis, there exists a chain:

(i).....$(p_{n+1},b_{n+1},\gamma_{n+1}),e_n^{-1},\ldots,e_2^{-1},(p_2,b_2,\gamma_2)$

where $b_{n+1}=\hat{a}_2$, and $b_2=\emptyset$.

Consider the chain (p_1,a_1,γ_1), e_1, (p_2,a_2,γ_2). Now if $e_1 = (p_1,\not{c},p_2) \in$ TRANS(A(C)), then $a_2=\not{c}a_1$, $\gamma_2=\gamma_1$, and $e_1^{-1}=(p_2,\not{c},p_1)$, so that:

(ii).....$(p_2,\not{c},\gamma_2) \vdash e_1^{-1} \dashv (p_1,\emptyset,\gamma_1)$

If we let $c_j = b_j \not{c}$ for all $j \in \{2,\ldots,n+1\}$, we obtain from (i) and (ii), the chain:

$$(p_{n+1}, c_{n+1}, \gamma_{n+1}), e_n^{-1}, \ldots, e_1^{-1}, (p_1, c_1, \gamma_1)$$

where $c_1 = \emptyset$, and $c_{n+1} = b_{n+1}\not{c} = \hat{a}_2\not{c} = \hat{a}_1$.

Finally, if $e_1 \in \mathrm{PUSH}(A(C)) \cup \mathrm{POP}(A(C))$, then $a_1 = a_2$ so $(p_1, \emptyset, \gamma_1)$, e_1, $(p_2, \mathcal{E}, \gamma_2)$ is a chain of the given form. Therefore $(p_2, \mathcal{E}, \gamma_2)$, e_1^{-1}, $(p_1, \mathcal{E}, \gamma_1)$ is a chain, where $b_2 = b_1 = \mathcal{E}$. From this and (i) we get a chain:

$$(p_{n+1}, b_{n+1}, \gamma_{n+1}), e_n^{-1}, \ldots, e_1^{-1}, (p_1, b_1, \gamma_1)$$

where $b_1 = \emptyset$, and $b_{n+1} = \hat{a}_2 = \hat{a}_1$.

$\square$

__4.2.2.10__: __Definition__: If p and q are two subformulae of C, then p is said to be __attached to__ q in A(C) if and only if for some $a \in C^*$, $(p, a, \emptyset) \vdash^* (q, \mathcal{E}, \emptyset)$. We denote this $p \simeq q \bmod C$. We also say that p is attached to q __by the word a__.

__4.2.2.11__: __Lemma__: $\simeq \bmod C$ is an equivalence relation.

__Proof__: Since $(p, \mathcal{E}, \emptyset) \vdash^0 (p, \emptyset, \emptyset)$ 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 $\hat{a}$. Finally, if $p_1 \simeq p_2 \bmod C$ and

$p_2 \simeq p_3$ 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 $p_1 \simeq p_3$ mod C.

$\Box$

The following lemma establishes the relationship between the output partitions of CLASSIFY(C) and the automaton for C: namely, that the equivalence classes under $\simeq$ mod C are exactly the classes of F.out. In the corollaries, we note the resulting uniqueness of the partitions output by CLASSIFY, and the exact relationship between CLASSIFY and TRANSFORM.

<u>4.2.2.12</u>: <u>Lemma</u>: If F.out is an output partition of CLASSIFY(C), then $p \equiv q$ mod F.out if and only if $p \simeq q$ mod C.

<u>Proof</u>:

(A) Suppose $p \simeq q$ 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.out.

<u>Basis</u>: If $n=0$, then by the definition of $\vdash^0$, $p = q$ so that $p \equiv q$ mod F.out.

<u>Induction</u>: 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

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:

<u>case</u>(<u>a</u>) $e_1 = (p_1, \emptyset, p_2) \in$ TRANS(A(C))

In this case $\{p_1, p_2\} \in C$. Consider an execution of CLASSIFY(C). At C1, S initially contains $\{p_1, p_2\}$ but at the last encounter with C1, S is empty, so on some loop from C1 to C1, $\{p_1, p_2\}$ 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 C1, $[p_1] = [p_2]$, so that $p_1 \equiv p_2$ mod F.out.

$$
\begin{aligned}
\text{Now } (p_2, a_2, \gamma_2) &\vdash^{k-1} (p_{k+1}, a_{k+1}, \gamma_{k+1}) \\
\text{But } \gamma_2 &= \gamma_1 \text{ since } e_1 \in \text{TRANS(A(C))} \\
&= \emptyset \\
\gamma_{k+1} &= \emptyset \\
a_{k+1} &= \emptyset \\
\therefore (p_2, a_2, \emptyset) &\vdash^{k-1} (p_{k+1}, \emptyset, \emptyset) \\
\therefore p_2 &\equiv p_{k+1} \text{ mod F.out} \\
&\quad \text{by the induction hypothesis.} \\
\therefore p_1 &\equiv p_{k+1} \text{ mod F.out} \\
\text{\underline{i.e.} } p &\equiv q \text{ mod F.out}
\end{aligned}
$$

<u>case</u>(<u>b</u>) $e_1 = (p_1, (f,i), p_2) \in$ PUSH(A(C))

Let m be the greatest integer such that $m \geq 2$, and for all $j \in \{2, \ldots, m\}$, $\exists\, \beta_j \in Z^*$ such that $\gamma_j = \beta_j(f,i)$; then $\gamma_m = (f,i) = \gamma_2$, $\gamma_{m+1} = \emptyset$ and $e_m = (p_m, (f,i), p_{m+1}) \in$ POP(A(C)).

Now $(p_j, a_j, \beta_j(f,i) \vdash (p_{j+1}, a_{j+1}, \beta_{j+1}(f,i))$

for all $j \in \{2,\ldots,m-1\}$

So by 4.2.2.8(ii):

$(p_j, a_j, \beta_j) \vdash (p_{j+1}, a_{j+1}, \beta_{j+1})$

for all $j \in \{2,\ldots,m-1\}$

$\therefore (p_2, a_2, \beta_2) \vdash^{m-1} (p_m, a_m, \beta_m)$

But $\beta_2 = \beta_m = \emptyset$, and $a_2 = ba_m$ for some $b \in C^*$. So by 4.2.2.8(i):

$(p_2, b, \emptyset) \vdash^{m-1} (p_m, \emptyset, \emptyset)$

Since $m-2<k$, we can apply the induction hypothesis to obtain:

(i)...............$p_2 \equiv p_m$ mod F.out

Now $e_1 = (p_1,(f,i),p_2) \in$ PUSH(A(C))

and $e_m = (p_m,(f,i),p_{m+1}) \in$ 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 $p_1 = q_{2i}$

$p_{m+1} = q_{mi}$

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 P.out and therefore, because of (ii), that:

(iii).............$p_1 \equiv p_{m+1}$ mod F.out

Finally, $(p_{m+1}, a_{m+1}, \gamma_{m+1}) \vdash^{k-m} (p_{k+1}, a_{k+1}, \gamma_{k+1})$.
But $\gamma_{m+1} = \emptyset = \gamma_{m+1}$, and $a_{m+1} = ba_{m+1}$ for some $b \in C^*$, so by 4.2.2.8(i):

$$(p_{m+1}, b, \emptyset) \vdash^{k-m} (p_{k+1}, \emptyset, \emptyset)$$

$$\therefore p_{m+1} \equiv p_{k+1} \bmod F.out$$

From this and (iii), we obtain:

$$p_1 \equiv p_{k+1} \bmod F.out$$

$$\underline{i.e.}\ p \equiv q \bmod F.out$$

(B) Now suppose that $p \equiv q \bmod F.out$. Let $H(F,S)$ assert that:

"$\underline{if}$ $p \equiv q \bmod F$, or $\{p,q\} \in S$

$\underline{then}$ $p \simeq q \bmod C$ "

Consider any loop from C3 to C3 via PC6, and suppose that $H(F(C3,1), S(C3,1) \cup \{\{p_1,p_2\}\})$ holds. We show that $H(F(C3,2), S(C3,2) \cup \{\{p_1,p_2\}\})$ holds. Since neither F nor S are changed on the path returning to C3 via PC7, we need only consider the path returning via PC8. For this path we have:

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

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

If $p \equiv q \bmod F(C3,2)$, then $p \equiv q \bmod F(C3,1)$, so by hypothesis, $p \simeq q \bmod C$. If $\{p,q\} \in S(C3,2)$, either $\{p,q\} \in S(C3,1)$, 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)........$(p,(f,i),t_1) \in PUSH(A(C))$

and $(t_2,(f,i),q) \in POP(A(C))$

$$\text{Also } t_1 \equiv p_1 \bmod F(C3,1)$$

$$\text{and } p_2 \equiv t_2 \bmod F(C3,1)$$

So by the induction hypothesis:

$$t_1 \simeq p_1 \bmod C$$

$$p_2 \simeq t_2 \bmod C$$

$$\text{and } p_1 \simeq p_2 \bmod C$$

$$\therefore t_1 \simeq t_2 \bmod C$$

So for some $a \in C^*$:

$$(t_1,a,\emptyset) \vdash^* (t_2,\emptyset,\emptyset)$$

and therefore, by 4.2.2.8(ii):

$$(t_1,a,(f,i)) \vdash^* (t_2,\emptyset,(f,i))$$

So because of (i):

$$(p,a,\emptyset) \vdash^* (q,\emptyset,\emptyset)$$

$$\therefore p \simeq q \bmod C$$

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 $p \equiv q \bmod F.in$, then $p = q$, since each class of F.in contains one element, so that $p \equiv q \bmod c$. If $\emptyset = \{p,q\} \in C$, then $(p,\emptyset,q) \in TRANS(A(C))$ so that $(p,\emptyset,\emptyset) \vdash (q,\emptyset,\emptyset)$; that is, $p \simeq q \bmod C$. 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 PC3:

$$F(C1,2) = F(C1,1)$$

$$S(C1,2) = S(C1,1) - \{\{p_1,p_2\}\}$$

If $p \equiv q \bmod F(C1,2)$, then $p \equiv q \bmod F(C1,1)$; if $\{p,q\} \in S(C1,2)$, then $\{p,q\} \in S(C1,1)$. In either case $p \simeq q \bmod C$.

(2) Path from C1 to C1 via PC5:

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

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

Since $H(F(C1,1),S(C1,1))$ holds, $H(F(C3,1),S(C3,1) \cup \{\{p_1,p_2\}\})$ also holds. Suppose C3 is encountered m times, then by the invariance of $H(F,S \cup \{\{p_1,p_2\}\})$ at C3, $H(F(C3,m),S(C3,m) \cup \{\{p_1,p_2\}\})$ holds.

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

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

If $\{p,q\} \in S(C1,2\}$, then $\{p,q\} \in S(C3,m)$ so by $H(F(C3,m),S(C3,m) \cup \{\{p_1,p_2\}\})$, $p \simeq q \bmod C$. If $p \equiv q \bmod F(C1,2)$ we have two cases:

<u>either</u> $p \equiv q \bmod F(C3,m)$

so by $H(F(C3,m),S(C3,m) \cup \{\{p_1,p_2\}\})$, we have:

$p \simeq q \bmod C$.

<u>or</u> $p \equiv p_1 \bmod F(C3,m)$

$q \equiv p_2 \bmod F(C3,m)$

so by $H(F(C3,m),S(C3,m) \cup \{\{p_1,p_2\}\})$ we have:

$p \simeq p_1 \bmod C$

$q \simeq p_2 \bmod C$

$p_1 \simeq p_2 \bmod C$

$\therefore p \simeq q \bmod C$

Hence $H(F,S)$ always holds at C1, so in particular:

$$p \equiv q \bmod F.out \text{ implies } p \simeq q \bmod C$$

□

<u>4.2.2.13</u> <u>Corollary</u>: 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).

<u>Proof</u>: The uniqueness of F.out is obvious from lemma 4.2.2.12, and implies the uniqueness of P.out by lemma 4.2.1.5.

□

<u>4.2.2.14</u>: <u>Corollary</u>: 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.

<u>Proof</u>: 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).

<u>4.2.2.15</u>: <u>Example</u>: Consider the set of constraints C of example 4.2.1.2. We denote the constraints in this set by $\phi_1,\ldots,\phi_4$ as follows:

ϕ_1: {F(x,x), v}

ϕ_2: {v, F(f(a),h(y))}

$\not{c}_3$: $\{F(u,f(y)),\ F(h(g(b)),u)\}$

$\not{c}_4$: $\{h(u),\ h(f(a))\}$

The automaton A(C) for this set is illustrated in figure 4.4. Note from figure 4.3 that:

$$a \equiv g(b) \bmod F.out(C)$$

so by lemma 4.2.2.12:

$$a \simeq g(b) \bmod C$$

Figure 4.5 shows a chain demcnstrating 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 4.1.3). Similarly, from the output partition F.out(C) of CLASSIFY(C), we construct a labelled digraph U(C).

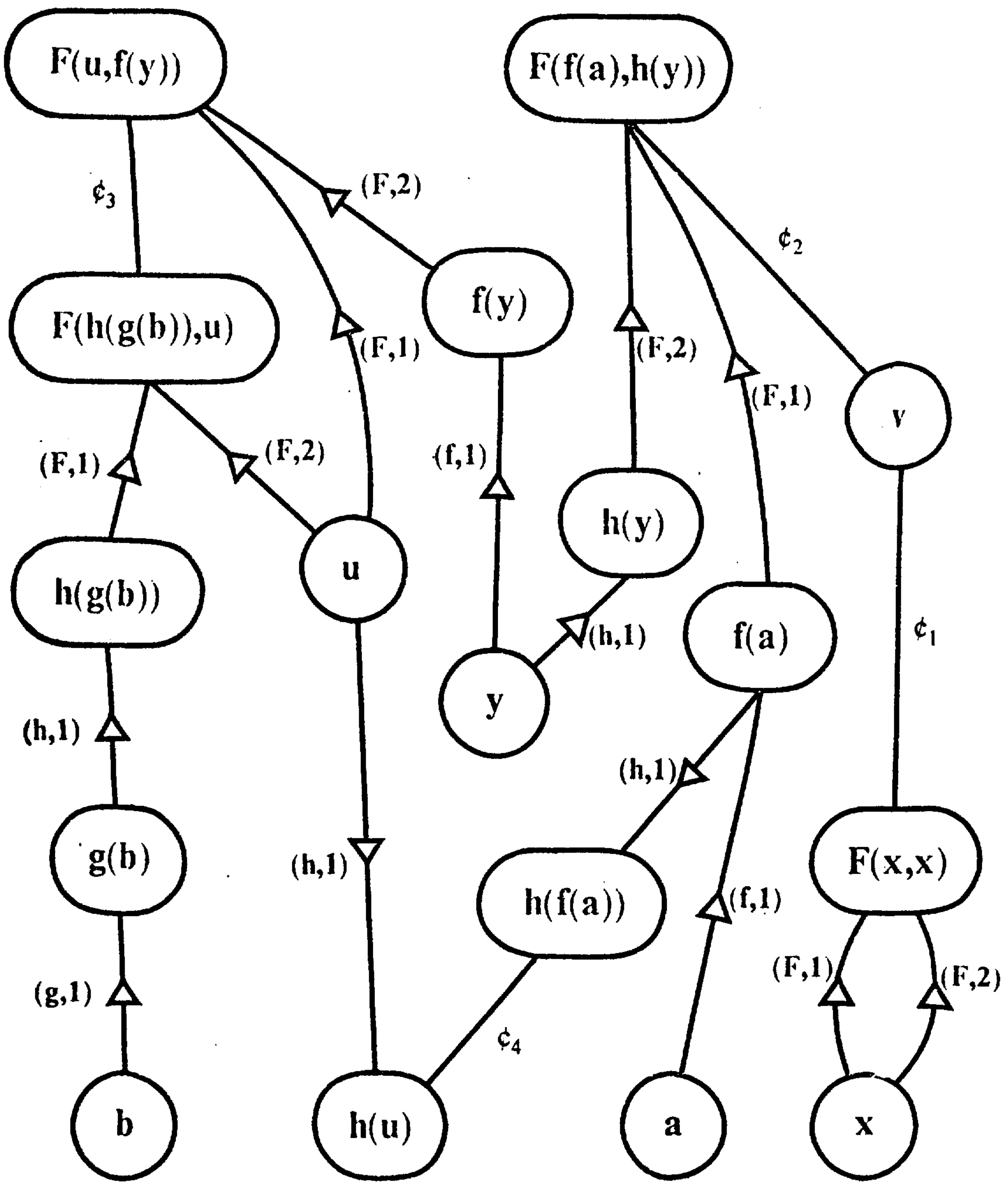
4.2.3.1: Definition: If C is a set of constraints, the unification graph U(C) for C, is a labelled, directed graph, where:

$$V(U(C)) = F.out(C)$$

$$I(U(C)) = M(C) \quad \text{(definition 4.2.2.1)}$$

and the arc set is defined as follows. Suppose there are m classes in P.out(C), and let $t_1,\ldots,t_m$ be m terms such that $\langle t_i \rangle = \langle t_j \rangle$ if and only if $i=j$. Suppose $t_i = f_i(p_{i1},\ldots,p_{in_i})$ for all $i \in \{1,\ldots,m\}$, then:

4.2.3.1



The automaton A(C) for the set of constraints C = {$¢_1, ¢_2, ¢_3, ¢_4$} 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 POP arc from q to p.

<u>Figure 4.4</u>

State	Input	Stack
a	$¢_4¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	Ø
f(a)	$¢_4¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	(f, 1)
h(f(a))	$¢_4¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	(h, 1)(f, 1)
h(u)	$¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	(h, 1)(f, 1)
u	$¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	(f, 1)
F(h(g(b)), u)	$¢_3¢_2¢_1¢_1¢_2¢_4¢_3$	(F, 2)(f, 1)
F(u, f(y))	$¢_2¢_1¢_1¢_2¢_4¢_3$	(F, 2)(f, 1)
f(y)	$¢_2¢_1¢_1¢_2¢_4¢_3$	(f, 1)
y	$¢_2¢_1¢_1¢_2¢_4¢_3$	Ø
h(y)	$¢_2¢_1¢_1¢_2¢_4¢_3$	(h, 1)
F(f(a), h(y))	$¢_2¢_1¢_1¢_2¢_4¢_3$	(F, 2)(h, 1)
v	$¢_1¢_1¢_2¢_4¢_3$	(F, 2)(h, 1)
F(x, x)	$¢_1¢_2¢_4¢_3$	(F, 2)(h, 1)
x	$¢_1¢_2¢_4¢_3$	(h, 1)
F(x, x)	$¢_1¢_2¢_4¢_3$	(F, 1)(h, 1)
v	$¢_2¢_4¢_3$	(F, 1)(h, 1)

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 $¢_4¢_3¢_2¢_1¢_1¢_2¢_4¢_3 \in C^*$, where C is the set of constraints of example 4.2.2.15. (Continued on page 144.)

Figure 4.5

<u>4.3.2</u>: <u>Lemma</u>: 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).

<u>Proof</u>: 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 $(p_{n+1}, a, \emptyset)$ to $(p_1, \emptyset, \gamma)$ for some $a \in C^*$ and $\gamma \neq \emptyset$.

<u>Basis</u>: 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:

$$(p_2, a_2, \emptyset) \vdash^* (q_i, \emptyset, \emptyset)$$

$$\text{and } (t, a_1, \emptyset) \vdash^* (p_1, \emptyset, \emptyset)$$

But $(q_i, (f, i), t) \in \text{PUSH}(A(C))$

$$\therefore (q_i, \emptyset, \emptyset) \vdash (t, \emptyset, (f, i))$$

Hence by 4.2.2.8(i), (ii) and (v):

$$(p_2, a_2 a_1, \emptyset) \vdash^* (p_1, \emptyset, (f, i))$$

So in this case a chain with the required properties exists.

<u>Induction</u>: 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 $\gamma_1, \gamma_2 \neq \emptyset$

$$(p_{n+1}, a_1, \emptyset) \vdash^* (p_n, \emptyset, \gamma_1)$$

$$\text{and } (p_n, a_2, \emptyset) \vdash^* (p_1, \emptyset, \gamma_2)$$

Therefore, by 4.2.2.8(i), (ii) and (v):

$$(p_{n+1},a_1a_2,\emptyset) \vdash^* (p_1,\emptyset,\gamma_1\gamma_2)$$

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).

$\Box$

4.3.3: Lemma: If $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is a chain in A(C) such that $\gamma_1 = \emptyset$, and $\gamma_{n+1} = (f_m,j_m)\ldots(f_1,j_1)$, then there are m unique states $q_1,\ldots,q_m$, m stacks $\alpha_1,\ldots,\alpha_m$, and m integers $i_1,\ldots,i_m$ such that:

(1) $1<i_1<i_2<\ldots<i_m\leq n+1$

(2) $q_j = p_{i_j}$, $\alpha_j = \gamma_{i_j}$ for all $j \in \{1,\ldots,m\}$

(3) q_j is a term beginning with f_j , for all $j \in \{1,\ldots,m\}$

(4) $|\gamma_k| \geq |\alpha_j|$ for $k\geq i_j$

(5) if $m>1$, $([q_j],f_j,[q_{j-1}]) \in E(U(C))$ for all $j \in \{2,\ldots,m\}$

(6) $[q_m] = [p_{n+1}]$

(7) $([q_1],f_1,[p_1]) \in E(U(C))$

Proof: We use induction on m.

Basis: m=1. Let r be the largest integer such that $\gamma_r = \emptyset$; let $i_1 = r+1$, $q_1 = p_{r+1}$, and $\alpha_1 = \gamma_{r+1}$, then:

(1) $1<i_1\leq n+1$

(2) holds trivially

(3) Since $\gamma_r = \emptyset$ and $\gamma_{r+1} = \emptyset$:

$e_r = (p_r,(f_m,j_m),p_{r+1}) \in PUSH(A(C))$

$\therefore$ p_{r+1} is a term beginning with f_1

<u>i.e.</u> q_1 is a term beginning with f_1

(4) Since $\gamma_r = \emptyset$ and $\gamma_k \neq \emptyset$ for $k>r$, we can, for each $k \geq r+1$, write γ_k in the form $\beta_k(f_1,j_1)$ for some $\beta_k \in Z^*$

$$\text{But } \gamma_{r+1} = (f_1,j_1) = \alpha_1$$

$$\therefore |\gamma_k| \geq |\alpha_1| \text{ for } k \geq i_1$$

(6) $$(p_k,a_k,\gamma_k) \vdash_{e_k} (p_{k+1},a_{k+1},\gamma_{k+1})$$

$$\text{for all } k \in \{1,\ldots,n\}$$

$$\text{But } \gamma_k = \beta_k(f_1,j_1)$$

$$\text{for all } k \in \{r+1,\ldots,n\}$$

$$\therefore (p_k,a_k,\beta_k) \vdash_{e_k} (p_{k+1},a_{k+1},\beta_{k+1})$$

$$\text{for all } k \in \{r+1,\ldots,n\}$$

Now $\beta_{r+1} = \emptyset$, $\beta_{n+1} = \emptyset$, and $a_{r+1} = ba_{n+1}$ for some $b \in C^*$

$$\therefore (p_{r+1},a_{r+1},\emptyset) \vdash^* (p_{n+1},\emptyset,\emptyset)$$

So by lemma 4.2.2.12:

$$[p_{r+1}] = [p_{n+1}]$$

$$\underline{\text{i.e.}}\ [q_1] = [p_{n+1}]$$

(7) $$(p_k,a_k,\gamma_k) \vdash_{e_k} (p_{k+1},a_{k+1},\gamma_{k+1})$$

$$\text{for all } k \in \{1,\ldots,r-1\}$$

But $\gamma_1 = \gamma_r = \emptyset$, and $a_1 = ba_r$ for some $b \in C^*$

$$\therefore (p_1,b,\emptyset) \vdash^* (p_r,\emptyset,\emptyset)$$

So by lemma 4.2.2.12:

$$[p_1] = [p_r]$$

Now $e_r = (p_r,(f_1,j_1),p_{r+1}) \in \text{PUSH}(A(C))$

$$\therefore p_{r+1} = f_1(q_1,\ldots,q_k)$$

$$\text{for some } q_1,\ldots,q_k$$

and $p_r = q_{j_1}$

So $([p_{r+1}], f_1, [p_r]) \in E(U(C))$

<u>i.e.</u> $([q_1], f_1, [p_1]) \in E(U(C))$

<u>Induction</u>: assume the result holds for chains whose final configurations have stacks of length less than m.

Let r be the greatest integer such that:

$\gamma_r = (f_{m-1}, j_{m-1}) \ldots (f_1, j_1)$

Then $(p_1, a_1, \gamma_1), e_1, \ldots, e_{r-1}, (p_r, a_r, \gamma_r)$ is a chain such that $\gamma_1 = \emptyset$ and $|\gamma_r| < m$. So by the induction hypothesis, states $q_1, \ldots, q_{m-1}$, stacks $\alpha_1, \ldots, \alpha_{m-1}$, and integers $i_1, \ldots, i_{m-1}$ exist, satisfying the conditions.

Let $i_m = r+1$, $q_m = p_{r+1}$ and $\alpha_m = \gamma_{r+1}$, then:

(1) $1 < i_1 < \ldots < i_{m-1} \le r$ by the induction hypothesis

$\therefore 1 < i_1 < \ldots, i_m = r+1 \le n+1$

(2) holds trivially

(3) Obviously $\gamma_{r+1} = (f_m, j_m) \ldots (f_1, j_1)$

$\therefore e_r = (p_r, (f_1, j_1), p_{r+1}) \in PUSH(A(C))$

$\therefore p_{r+1}$ is a term beginning with f_1

<u>i.e.</u> q_m is a term beginning with f_1

(4) We can write γ_k in the form $\beta_k \gamma_{r+1}$ for each $k \ge r+1$

$\therefore |\gamma_k| \ge |\gamma_{r+1}| = |\alpha_m|$ for $k \ge i_m$

(5) For $j \in \{1, \ldots, m-1\}$, by the induction hypothesis:

$([q_j], f_j, [q_{j-1}]) \in E(U(C))$

Now $e_r = (p_r, (f_m, j_m), p_{r+1}) \in PUSH(A(C))$

So $p_{r+1} = f_m(q_1, \ldots, q_k)$

for some $q_1, \ldots, q_k$

and $p_r = q_{j_m}$

$\therefore$ $([p_{r+1}], f_m, [p_r]) \in E(U(C))$

By the induction hypothesis, $[q_{m-1}] = [p_r]$

$\therefore$ $([q_m], f_m, [q_{m-1}]) \in E(U(C))$

(6) $(p_k, a_k, \gamma_k) \vdash e_k \dashv (p_{k+1}, a_{k+1}, \gamma_{k+1})$

for all $k \in \{1,\ldots,n\}$

But $\gamma_k = \beta_k \gamma_{k+1}$

for all $k \in \{r+1,\ldots,n\}$

Now $\beta_{r+1} = \emptyset$, $\beta_{n+1} = \emptyset$, and $a_{r+1} = ba_{n+1}$ for some $b \in C^*$

$\therefore$ $(p_{r+1}, a_{r+1}, \emptyset) \vdash^* (p_{n+1}, \emptyset, \emptyset)$

So by lemma 4.2.2.12:

$[p_{r+1}] = [p_{n+1}]$

<u>i.e.</u> $[q_m] = [p_{n+1}]$

(7) Holds by the induction hypothesis.

$\square$

<u>4.3.4</u>: <u>Corollary</u>: If the chain in 4.3.3 is a loop, conditions (5), (6) and (7) can be replaced by:

(5') $([q_j], f_j, [q_{j-1}]) \in E(U(C))$ for all $j \in$, $\{2,\ldots,m\}$

$([q_1], f_1, [q_m]) \in E(U(C))$

<u>Proof</u>: If the chain is a loop, $p_1 = p_{n+1}$, so from (6) and (7):

$([q_1], f_1, [q_m]) \in E(U(C))$

$\square$

<u>4.3.5</u>: <u>Corollary</u>: 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.

$\square$

4.3.6: Definition: If A(C) has a loop $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$, 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 $q_k = p_j$, then $|\gamma_i| \geq |\gamma_j|$ for $i \geq j$. Therefore, for each $i \geq j$, we can write γ_i in the form $\beta_i \gamma_j$ for some $\beta_i \in Z^*$. Also, $a_1 = ca_j$ for some $c \in C^*$. Then the q_k-canonical form of the original loop is:

$$(p_j, a_j c, \beta_j), e_j, \ldots\ldots, e_n, (p_{n+1}, c, \beta_{n+1}), e_1,$$
$$(p_2, c, \gamma_2 \beta_{n+1}), e_2, \ldots, e_{j-1}, (p_j, \emptyset, \gamma_j \beta_{n+1}),$$

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:

$$\{\not{c} \mid \not{c} \text{ occurs in } a_1\} = \{\not{c} \mid \not{c} \text{ occurs in } a_j c\}$$

4.3.7: 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 semi-simple if and only if for all i and j such that $1 \leq i < j \leq n+1$, either $p_i \neq p_j$ or $\gamma_i \neq \gamma_j$.

4.3.8: Definition: A chain in A(C):

$$(p_1,a_1,\gamma_1),e_1,\dots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$$

is said to be **simple** if and only if it is semi-simple, and for all i and j such that either $1 \le i < j < n+1$ or $1 < i < j \le n+1$:

if $p_i = p_j$

then $\exists$ k $\in \{i+1,\dots,j-1\}$ such that:

$|\gamma_k| < |\gamma_i|$

and $|\gamma_k| < |\gamma_j|$

4.3.9: Lemma: Let Δ 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 Δ is finite.

Proof: Suppose $(p_1,a_1,\gamma_1),e_1,\dots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is a simple chain in A(C), where $\gamma_1 = \emptyset$ and $|\gamma_{n+1}| > |PUSH(A(C))| + 1$.

Let m be the greatest integer such that $\gamma_m \neq \gamma_{n+1}$, then $|\gamma_m| > |PUSH(A(C))|$.

Since $\gamma_1 = \emptyset$, each element of γ_m is added by some arc e in the chain, where e $\in$ PUSH(A(C)). Hence, since $|\gamma_m| > |PUSH(A(C))|$, two elements of γ_m are added by the same arc in PUSH(A(C)), so therefore there are two integers i and j such that $1 \le i < j < m < n+1$ and:

(i) $e_i = e_j \in PUSH(A(C))$

and (ii) for all k $\in \{i+1,\dots,n\}$,

$\exists\ \beta_k \in Z^*$ such that $\gamma_k = \beta_k\gamma_i$

Because of (i), we have $p_i = p_j$, and from (ii), $|\gamma_k| \geq |\gamma_i|$ 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 $|PUSH(A(C))| + 1$. But the stack alphabet is finite so that the set:

$$V(A(C)) \times \{\gamma \mid |\gamma| \leq |PUSH(A(C))| + 1\},$$

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 Δ. Since $a_{n+1} = b_{n+1} = \emptyset$, 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 Δ. 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 Δ is finite.

$\square$

<u>4.3.10</u>: <u>Lemma</u>: If there is a chain of length n in A(C) from (p,a,γ) to (q,b,α), 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,γ) to (q,b,α), for some $c \in C^*$.

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 $p_1 \neq p_{n+1}$, or $\gamma_1 \neq \gamma_{n+1}$, then $p_i = p_j$ and $\gamma_i = \gamma_j$ for some i and j such that $i<j$ and either $i\neq 1$ or $j \neq n+1$. Now $a_1 = da_i$ and $a_i = ha_j$, for some $d,h \in C^*$.

$\therefore (p_1,da_j,\gamma_1) \vdash^{i-1} (p_i,a_j,\gamma_i) = (p_j,a_j,\gamma_j)$

Also $(p_j,a_j,\gamma_j) \vdash^{n-j+1} (p_{n+1},a_{n+1},\gamma_{n+1})$

$\therefore (p_1,da_j,\gamma_1) \vdash^{n-(j-i)} (p_{n+1},a_{n+1},\gamma_{n+1})$

Hence there is a chain of length $n-(j-i) < n$, from (p_1,da_j,γ_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 $i\neq 1$ or $j\neq n+1$, $p_i = p_j$, $\gamma_i \neq \gamma_j$ and for all $k \in \{1,\ldots,j\}$, either $|\gamma_k| \geq |\gamma_i|$ or $|\gamma_k| \geq |\gamma_j|$.

case(a): $|\gamma_i| \leq |\gamma_j|$

In this case, $|\gamma_k| \geq |\gamma_i|$ for all $k \in \{i,\ldots,j\}$, so for each $k \in \{i,\ldots,j\}$, $\exists\ \beta_k \in Z^*$ such that $\gamma_k = \beta_k\gamma_i$.

But $(p_k,a_k,\gamma_k) \vdash\!\!e_k\!\!\dashv (p_{k+1},a_{k+1},\gamma_{k+1})$

for all $k \in \{1,\ldots,j-1\}$

$\therefore (p_k, a_k, \beta_k) \vdash e_k \dashv (p_{k+1}, a_{k+1}, \beta_{k+1})$

for all $k \in \{i,\ldots,j-1\}$

But for all $k \in \{i,\ldots,j\}$, $\exists\ b_k \in C^*$ such that $a_k = b_k a_j$.

$\therefore (p_k, b_k, \beta_k) \vdash e_k \dashv (p_{k+1}, b_{k+1}, \beta_{k+1})$

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 $\beta_i = \emptyset$, $b_j = \varepsilon$, $p_i = p_j$, 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 $<n$.

<u>case(b)</u>: $|\gamma_j| \leq |\gamma_i|$

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 $\beta_j = \emptyset$, $b_j = \varepsilon$, $p_i = p_j$, $\beta_i \neq \emptyset$.

By lemma 4.2.2.9, therefore, there is a chain in A(C):

$$(p_j, c_j, \beta_j), e_{j-1}^{-1}, \ldots, e_i^{-1}, (p_i, c_i, \beta_i)$$

such that $c_i = \varepsilon$.

The length of this chain is $j-i < n$. 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 $<n$.

$\square$

4.3.12: Definition: A loop in A(C) is said to be <u>fundamental</u> 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.

$\square$

4.3.14: Definition: If p and q are subformulae of C and there is a simple chain from $(p,a,\emptyset)$ to $(q,\emptyset,\emptyset)$ for some $a \in C^*$, then p is said to be <u>simply attached</u> 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 $(p_1,a_1,\gamma_1),e_1,\dots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ be a simple chain such that $\gamma_1=\emptyset$, and $a_1=\emptyset$, then:

$$e_i \in PUSH(A(C)) \cup POP(A(C))$$

for all $i \in \{1,\ldots,n\}$

Now $\gamma_1=\emptyset$, $e_1 \in PUSH(A(C))$, so let k be the smallest integer such that $e_k \in PCP(A(C))$, then:

$$e_{k-1} = (p_{k-1},(f,i),p_k) \in PUSH(A(C))$$

$$\text{and } e_k = (p_k,(f,i),p_{k+1}) \in PCP(A(C))$$

$$\text{and } \gamma_{k-1} = \gamma_{k+1}$$

$$\text{Now } p_k = f(q_1,\ldots,q_m)$$

$$\text{for some } q_1,\ldots,q_m$$

$$\text{and } p_{k-1} = q_j = p_{k+1} \text{ for some } j \in \{1,\ldots,m\}$$

$$\therefore p_{k-1} = p_{k+1}$$

$$\text{and } \gamma_{k-1} = \gamma_{k+1}$$

which contradicts the fact that the chain is simple.

$$\therefore e_i \in PUSH(A(C)) \text{ for all } i \in \{1,\ldots,n\}$$

(i) Suppose the chain demonstrates that p_1 and p_{n+1} are simply attached by a_1, then $\gamma_1 = \gamma_{n+1} = \emptyset$. If $a_1 = \emptyset$, then as shown above, $e_i \in PUSH(A(C))$ for all $i \in \{1,\ldots,n\}$, which implies that $\gamma_{n+1} \neq \emptyset$; a contradiction.

$$\therefore a_1 \neq \emptyset$$

(ii) Suppose the above chain is a loop, then $p_1=p_{n+1}$ and $\gamma_1=\emptyset$. If $a_1=\emptyset$, then as shown above, $e \in PUSH(A(C))$ for all $i \in \{1,\ldots,n\}$.

$$\text{Let } e_i = (p_i,(f_i,j_i),p_{i+1})$$

$$\text{for all } i \in \{1,\ldots,n\}$$

$$\text{Then } p_{i+1} = f_i(q_{i1},\ldots,q_{in_i})$$

$$\text{for some } q_{i1},\ldots,q_{in_i}$$

$$\text{and for all } i \in \{1,\ldots,n\}$$

and $p_i = q_{ij}$ for some $j \in \{1,\dots,n_i\}$

$\therefore \text{ord}(p_i) < \text{ord}(p_{i+1})$ for all $i \in \{1,\dots,n\}$

$\therefore \text{ord}(p_1) < \text{ord}(p_{n+1})$

which contradicts the fact that $p_1 = p_{n+1}$

$\therefore a_1 \neq \emptyset$

□

<u>4.3.16</u>: <u>Lemma</u>: If $C_1 \subseteq C$ and $a \in C_1^+$, 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 $A(C_1)$.

(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 $A(C_1)$.

<u>Proof</u>: Suppose $(p_1,a_1,\gamma_1),e_1,\dots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is a simple chain in A(C) such that $\gamma_1=\emptyset$, $a_{n+1}=\emptyset$ and $a_1=a$. We note the following facts:

(a) $e_i \in \text{TRANS}(A(C))$ implies $p_i, p_{i+1} \in V(C_1)$

Proof: Suppose $e_i = (p_i, \not{c}, p_{i+1})$

Then $\not{c} \in C_1$, since $a_1 \in C_1^+$

But $\{p_i, p_{i+1}\} = \not{c}$

$\therefore p_i, p_{i+1} \in V(A(C_1))$

(b) $e_i \in \text{PUSH}(A(C))$ implies $e_{i+1} \notin \text{POP}(A(C))$

Proof: Suppose $e_i = (p_i,(f,j),p_{i+1}) \in \text{PUSH}(A(C))$

and $e_{i+1} = (p_{i+1},(f,j),p_{i+2}) \in \text{POP}(A(C))$

then $p_{i+1} = f(q_1,\dots,q_m)$

for some $q_1,\dots,q_m$

and $p_i = q_j = p_{i+2}$

also $\gamma_i = \gamma_{i+2}$

contradicting the fact that the chain is simple.

(c) If $p_i \notin V(A(C_1))$ and $e_i \in PUSH(A(C))$

then $p_{i+1} \notin V(A(C_1))$ and $e_{i+1} \in PUSH(A(C))$

Proof: Suppose $p_{i+1} \in V(A(C_1))$

Now $e_i \in PUSH(A(C))$

say $e_i = (p_i, (f, j), p_{i+1})$

$\therefore p_{i+1} = f(q_1, \ldots, q_m)$

for some $q_1, \ldots, q_m$

and $q_j = p_i$

$\therefore p_i \in V(A(C_1))$

which is a contradiction.

Therefore, by (a), since $p_{i+1} \notin V(A(C_1))$:

$e_{i+1} \notin TRANS(A(C))$

But $e_i \in PUSH(A(C))$, so by (b):

$e_{i+1} \notin POP(A(C))$

$\therefore e_{i+1} \in PUSH(A(C))$

(d) If $p_i \notin V(A(C_1))$ and $e_{i-1} \in POP(A(C))$

then $p_{i-1} \notin V(A(C_1))$ and $e_{i-2} \in POP(A(C))$

Proof: Suppose $p_{i-1} \in V(A(C_1))$

Now $e_{i-1} \in POP(A(C))$

say $e_{i-1} = (p_{i-1}, (f, j), p_i)$

$\therefore p_{i-1} = f(q_1, \ldots, q_m)$

for some $q_1, \ldots, q_m$

and $p_i = q_j$

$\therefore p_i \in V(A(C_1))$

which is a contradiction.

Therefore by (a), since $p_{i-1} \notin V(A(C_1))$:

$e_{i-2} \notin$ TRANS(A(C))

But $e_{i-1} \in$ POP(A(C)), so by (b):

$e_{i-2} \notin$ PUSH(A(C))

$\therefore$ $e_{i-2} \in$ POP(A(C))

We now prove the required results.

(1) Now suppose $p_1 \notin$ V(A(C_1))

then $e_1 \notin$ TRANS(A(C)) by (a)

$\therefore$ $e_1 \in$ PUSH(A(C))

So by induction, using (c):

$e_i \in$ PUSH(A(C))

for all $i \in \{1,\ldots,n+1\}$

So since $a_{n+1}=\emptyset$, $a_1=\emptyset$, which is a contradiction.

$\therefore$ $p_1 \in$ V(A(C_1))

(2) Suppose for some $i \in \{2,\ldots,n\}$, that $p_i \notin$ V(A(C_1)).

Then by (a), neither e_{i-1} nor e_i are in TRANS(A(C)).

(A) Suppose $e_{i-1} \in$ PUSH(A(C))

then $e_i \in$ PUSH(A(C)) by (b)

Therefore by induction using (c):

$e_{n+1} \in$ PUSH(A(C))

and $p_{n+1} \notin$ V(A(C_1))

In case (i) of the lemma, $\gamma_{n+1}=\emptyset$, so that:

$e_{n+1} \notin$ PUSH(AC))

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 $e_{i-1} \in PCP(A(C))$, then by induction using (d), $e_1 \in POP(A(C))$, which is impossible since $\gamma_1=\emptyset$.

(3) Finally, for case (i) of the lemma, suppose:

$$p_{n+1} \notin V(A(C_1))$$

Then $e_n \notin TRANS(A(C))$ by (a)

and $e_n \notin PUSH(A(C))$ since $\gamma_{n+1}=\emptyset$

$\therefore$ $e_n \in POP(A(C))$

So by induction using (d), $e_1 \in PCP(A(C))$, which is impossible since $\gamma_1=\emptyset$.

Therefore, $p_{n+1} \in V(A(C_1))$ and for all $i \in \{1,\dots,n\}$, $p_i \in V(A(C_1))$ and $e_i \in E(A(C_1))$. So in both (i) and (ii), the chain in A(C) is also in $A(C_1)$.

$\square$

<u>4.3.17</u>: <u>Lemma</u>: If C_1 and C_2 are sets of constraints such that $C_1 \subseteq C_2$, then every chain in $A(C_1)$ is also a chain in $A(C_2)$.

<u>Proof</u>: $A(C_1)$ is a subgraph of $A(C_2)$, so the result is obvious.

$\square$

<u>4.3.18</u>: <u>Definition</u>: If C is a nonempty set of constraints, a <u>labelling for C</u> is an ordered pair [L,LBL], where:

(i) L is a finite set called the <u>label set</u>,

(ii) LBL is a function from C into $2^L - \{\emptyset\}$, called the *label function*, such that:

$$\bigcup_{\phi \in C} LBL(\phi) = L$$

Note that every set of constraints has a labelling; namely $[C, LBL]$, where $LBL(\phi) = \{\phi\}$ for all $\phi \in C$

4.3.19: *Definition*: If $[L, LBL]$ is a labelling for a set of constraints C, and $L_1 \subseteq L$, we define a subset $C|[L_1, LBL]$ called the *restriction of C to L_1 under $[L, LBL]$*, by:

$$C|[L_1, LBL] = \{\phi \mid \phi \in C \text{ and } LBL(\phi) \cap L_1 \neq \emptyset\}$$

Since we never have occasion to consider more than one labelling at a time, we abbreviate this to $C|L_1$, and say that $C|L_1$ is *the restriction of C to L_1*. If C_1 is a subset of C such that $C_1 = C|L_1$ for some $L_1 \; L$, we say that C_1 is *a restriction of C*. Note that C is a restriction of itself, since $C = C|L$, and that $L_1 \subseteq L_2$ implies $C|L_1 \subseteq C|L_2$. If $L_1 \subseteq L$, then the labelling $[L, LBL]$ of C induces a labelling $[L_1, LBL_1]$ of $C_1 = C|L_1$, where $LBL_1 : C_1 \dashrightarrow 2^{L_1} - \{\emptyset\}$ is defined by:

$$LBL_1(\phi) = LBL(\phi) \cap L_1 \text{ for all } \phi \in C_1$$

4.3.20: *Definition*: If $[L, LBL]$ is a labelling for a set of constraints C, and $L_1 \subseteq L$, then L_1 is said to be *correct* if $C|L_1$ is unifiable. L_1 is *maximally correct with respect to L* if there is no $L_2 \subseteq L$ such that $L_1 \neq L_2$, $L_1 \subseteq L_2$ and L_2 is correct. If C is unifiable, L has only one maximally correct subset, namely L itself.

The relevance of constraint labellings to deduction plans is as follows. If G is a plan, then its constraint set C(G) has a labelling [SCL(G),LBL], where:

$$LBL(\phi) = \{e \mid e \in SOL(G) \text{ and } \phi \in C(e)\}$$

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 SOL(G), considered as the label set in the above labelling, 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]

<u>4.3.21</u>: <u>Definition</u>: 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 $2^L \dashrightarrow \{0,1\}$ defined by:

$$[0](L_1) = 0 \quad \text{for all } L_1 \subseteq L$$

$$[1](L_1) = 1 \quad \text{for all } L_1 \subseteq L$$

$$[l](L_1) = 0 \quad \text{iff } l \in L_1$$

$$[B_1+B_2](L_1) = [B_1](L_1) + [B_2](L_1)$$

$$[B_1 \cdot B_2](L_1) = [B_1](L_1) \cdot [B_2](L_1)$$

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:

$$\mathrm{ATTACH}(p,q) = \{a \mid p \text{ is simply attached to } q \text{ by } a\}$$

(ii) $$\mathrm{CONFLICT} = \{\{p,q\} \mid [p] = [q] \text{ and } \langle p\rangle \neq \langle q\rangle\}$$

(iii) For any subformula p of C:

$$\mathrm{LOOP}(p) = \{a \mid \exists \text{ a simple loop on } p \text{ with value } a\}$$

(iv) For any arc $e \in E(U(C))$:

$$\mathrm{TAIL}(e) = \{t \mid t \text{ begins with } f, \text{ where } e = ([t],f,[p])\}$$

(v) $$\mathrm{CIR} = \text{set of all cycles of } U(C)$$

(vi) Let H be the assertion defined by:

$$H(\mathcal{E}) \text{ iff for all } k \in \mathrm{CIR},\ \mathcal{E} \cap E(k) \neq \emptyset$$

Then let CCVER be any subset of E(U(C)) satisfying the condition:

$$|\mathrm{COVER}| = \min_{\substack{\mathcal{E}\subseteq E(U(C)) \\ \text{and } H(\mathcal{E})}} |\mathcal{E}|$$

Clearly if CIR = $\emptyset$, COVER = $\emptyset$

4.3.23: ***Definition***: We now define several Boolean expressions over L as follows:

(i) If $a \in C^+$, $$B_W(a) = \sum_{\not{c}\text{ occurs in } a} \ \prod_{l \in \mathrm{LBL}(\not{c})} l$$

(ii) If p and q are distinct subformulae of C:

$$B_A(p,q) = \begin{cases} 1 & \text{if } \mathrm{ATTACH}(p,q) = \emptyset \\ \prod_{a \in \mathrm{ATTACH}(p,q)} B_W(a) & \text{otherwise} \end{cases}$$

Note that by lemma 4.3.15 if $a \in \mathrm{ATTACH}(p,q)$ then $a \neq \emptyset$, so that $B_W(a)$ is defined. Also, by lemma 4.3.9, ATTACH(p,q) is finite.

(iii)

$$B_{CON} = \begin{cases} 1 & \text{if } \mathrm{CONFLICT} = \emptyset \\ \prod_{\{p,q\} \in \mathrm{CONFLICT}} B_A(p,q) & \text{otherwise} \end{cases}$$

(iv) For any subformula p of C:

$$B_L(p) = \begin{cases} 1 & \text{if } \mathrm{LOOP}(p) = \emptyset \\ \prod_{a \in \mathrm{LOOP}(p)} B_W(a) & \text{otherwise} \end{cases}$$

Note that by lemma 4.3.15, if $a \in \mathrm{LOOP}(p)$, then $a \neq \emptyset$, so that $B\ (a)$ is defined; and by lemma 4.3.9, LOOP(p) is finite.

(v) For any $e \in E(U(C))$:

$$B_T(e) = \prod_{t \in \mathrm{TAIL}(e)} B_L(t)$$

Note that $\mathrm{TAIL}(e) \neq \emptyset$

(vi) $$B_{CYC} = \begin{cases} 1 & \text{if COVER} = \emptyset \\ \prod_{e \in COVER} B_T(t) & \text{otherwise} \end{cases}$$

(vii) $$B_{UNIF} = B_{CON} \cdot B_{CYC}$$

In the rest of this section, we consider a set of constraints C with a labelling [L,LBL].

4.3.24: Lemma: If $L_1 \subseteq L$, and $a \in C^+$, then:

$$[B_W(a)](L_1) = 0 \text{ if and only if } a \in (C|L_1)^+$$

Proof: For all ¢ $\in$ C:

$$\begin{aligned} ¢ \in C|L_1 \ &\text{iff } LBL(¢) \cap L_1 \neq \emptyset \\ &\text{iff } \exists\ l \in LBL(¢) \cap L_1 \\ &\text{iff } \exists\ l \in LBL(¢) \text{ such that } [l](L_1) = 0 \\ &\text{iff } \Big[\prod_{l \in LBL(¢)} l\Big](L_1) = 0 \end{aligned}$$

$$\begin{aligned} \therefore\ a \in (C|L_1)^* \ &\text{iff } ¢ \in C|L_1 \text{ for all } ¢ \text{ occurring in } a \\ &\text{iff } \Big[\prod_{l \in LBL(¢)} l\Big](L_1) = 0 \\ &\qquad\qquad \text{for all } ¢ \text{ occurring in } a \\ &\text{iff } [B_W(a)](L_1) = 0 \end{aligned}$$

□

4.3.25: **Lemma:** If $L_1 \subseteq L$, and $A(C|L_1)$ either has a loop, or has a chain that is semi-simple but not simple, then $[B_{UNIF}](L_1) = 0$.

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 $a \in (C|L_1)^+$. 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:

$\{\not c \mid \not c \text{ occurs in } b\} = \{\not c \mid \not c \text{ occurs in } a\} \subseteq C|L_1$,

so that $[B_W(b)](L_1) = 0$, by lemma 4.3.24

$\therefore [B_{UNIF}](L_1) = 0$

$\square$

4.3.26: **Lemma:** If $L_1 \subseteq L$, then:

$$[B_{UNIF}](L_1) = 1 \text{ iff } C|L_1 \text{ is unifiable}$$

Proof:

(A) Suppose $C|L_1$ is not unifiable, then by theorem 4.2.3.3, we have two cases:

<u>case</u>(<u>a</u>): There exist subformulae p and q such that $p \equiv q \bmod F.out(C|L_a)$ and $p \not\equiv q \bmod P.out(C|L_a)$. By lemma 4.2.2.12, $p \simeq q \bmod C|L_a$, so there exists a chain in $A(C|L_a)$ from $(p,a,\emptyset)$ to $(q,\emptyset,\emptyset)$ for some $a \in (C|L_a)^*$, so by lemma 4.3.10, there is a semi-simple chain in $A(C|L_a)$ from $(p,b,\emptyset)$ to $(q,\emptyset,\emptyset)$. Note that $b \in (C|L_a)^*$. 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, $p \not\equiv q \bmod P.out(C)$

$$\therefore \{p,q\} \in CONFLICT$$

$$\text{and } b \in ATTACH(p,q)$$

Therefore $B_W(b)$ is a factor in the product B_{UNIF}. But $b \in (C|L_a)^*$, so by lemma 4.3.24:

$$[B_W(b)](L_a) = 0$$

$$\therefore [B_{UNIF}](L_a) = 0$$

(ii) If this chain is not simple, then by lemma 4.3.25:

$$[B_{UNIF}](L_a) = 0$$

<u>case</u>(<u>b</u>): $U(C|L_a)$ has a cycle. In this case, by corollary 4.3.5, $A(C|L_a)$ has a loop, so by lemma 4.3.25:

$$[B_{UNIF}](L_a) = 0$$

(B) Now suppose that $[B_{UNIF}](L_a) = 0$. Then we have two cases:

<u>case</u>(<u>a</u>): $[B_{CYC}](L_a) = 0$

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.

<u>case</u>(<u>b</u>) $[B_{CON}](L_a) = 0$

In this case, for some subformulae p and q of C, $p \equiv q$ mod F.out(C), $p \not\equiv q$ mod P.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 $p \simeq q$ mod $C|L_a$, so by lemma 4.2.2.12, $p \equiv q$ mod $F.out(C|L_a)$. 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 $P.out(C|L_a)$. Therefore, by theorem 4.2.3.3, $C|L_a$ is nonunifiable.

$\square$

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'.

(b) No product in B' contains repeated variables.

(c) B' defines the same Boolean function as B.

Also, for any two Boolean expressions B_1 and B_2, $[B_1] = [B_2]$ if and only if B_1 and B_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 B'_{UNIF} .

Proof:

(A) Suppose $L_1 = L - \{l_1,\ldots,l_n\}$ where $l_1\ldots l_n$ is a product in B'_{UNIF} .

$$\text{Then } [l_1\ldots l_n](L_1) = \prod_{i=1}^{n} [l_i](L_1)$$

$$= 1 \quad \text{since } l_i \notin L_1 \text{ for all } i \in \{1,\ldots,n\}$$

$$\therefore [B'_{UNIF}](L_1) = 1$$

$\therefore$ L_1 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 B'_{UNIF} such that:

$$[m_1\ldots m_k](L_2) = 1$$

$$\therefore m_i \notin L_2 \text{ for all } i \in \{1,\ldots,k\}$$

$$\therefore\ m_i \notin L_1 \text{ for all } i \in \{1,\ldots,k\}$$

$$\therefore\ \{m_1,\ldots,m_k\} \subseteq \{l_1,\ldots,l_n\}$$

If these sets are not equal, then the product $m_1 \ldots m_k$ subsumes the product $l_1 \ldots l_n$, which is impossible. Therefore $\{m_1,\ldots,m_k\} = \{l_1,\ldots,l_n\}$, so that $L_1 = L_2$

$$\therefore\ L_1 \text{ is maximally correct}$$

(B) Suppose $L_1 \subseteq L$ is maximally correct.

Let $L_1 = L - \{l_1,\ldots,l_n\}$. Now $[B^!_{UNI\,F}](L_1) = 1$, since L_1 is correct. Therefore there exists a product $m_1 \ldots m_k$ in $B^!_{UNI\,F}$ such that:

$$\prod_{i=1}^{k} [m_i](L_1) = 1$$

$$\therefore\ m_i \notin L_1 \text{ for all } i \in \{1,\ldots,k\}$$

$$\therefore\ \{m_1,\ldots,m_k\} \subseteq \{l_1,\ldots,l_n\}$$

Let $L_2 = L - \{m_1,\ldots,m_k\}$. Then L_2 is correct by (A), and $L_1 \subseteq L_2$. But L_1 is maximally correct, so that $L_2 = L_1$.

$$\therefore\ l_1 \ldots l_n \text{ is a product in } B^!_{UNI\,F}$$

□

4.3.28: <u>Example</u>: Consider the set of constraints C as follows:

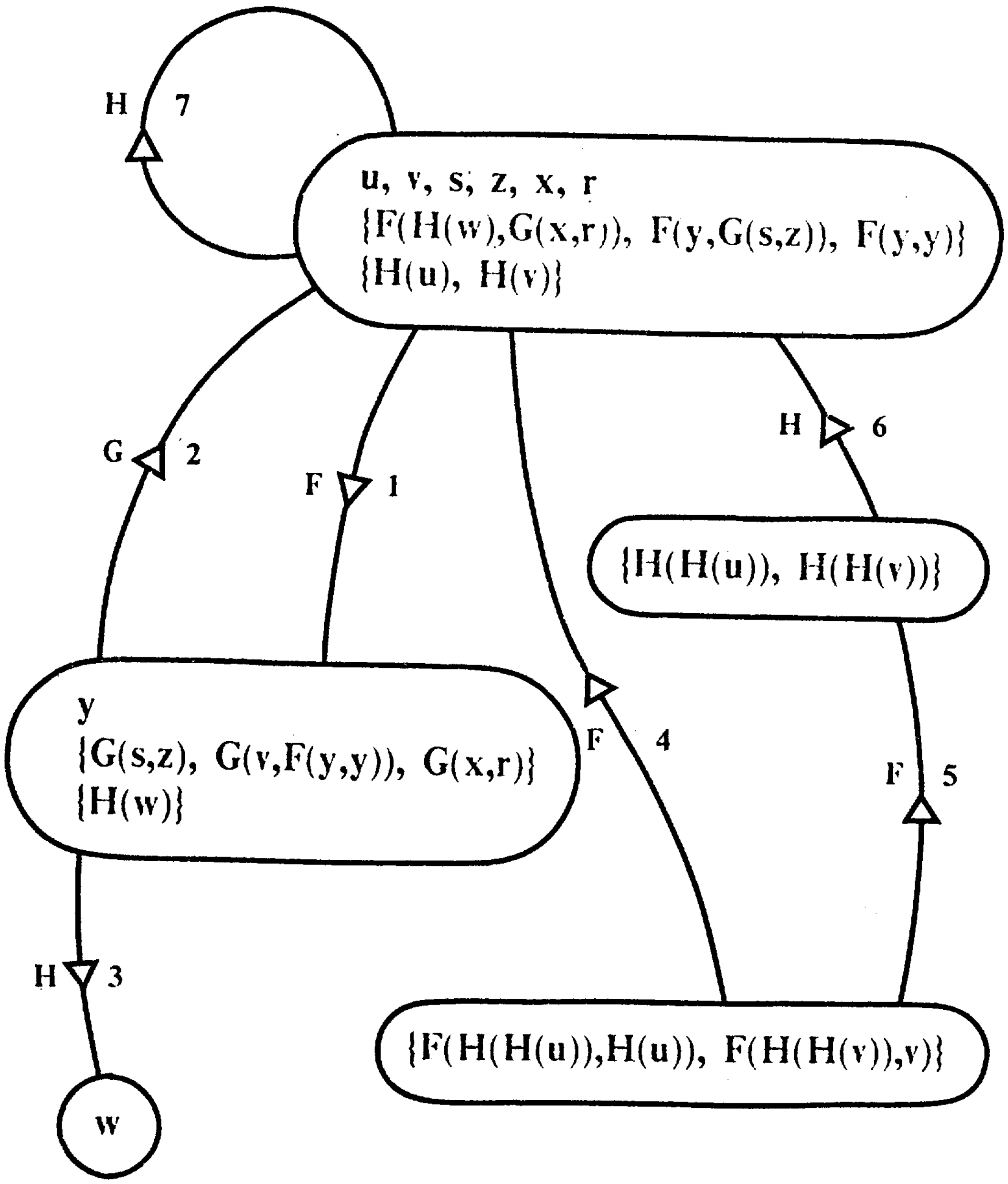
ϕ_1: $\{G(s,z),\ G(v,F(y,y))\}$

ϕ_2: $\{u,\ F(y,G(s,z))\}$

ϕ_3: $\{u,\ F(H(w),G(x,r))\}$

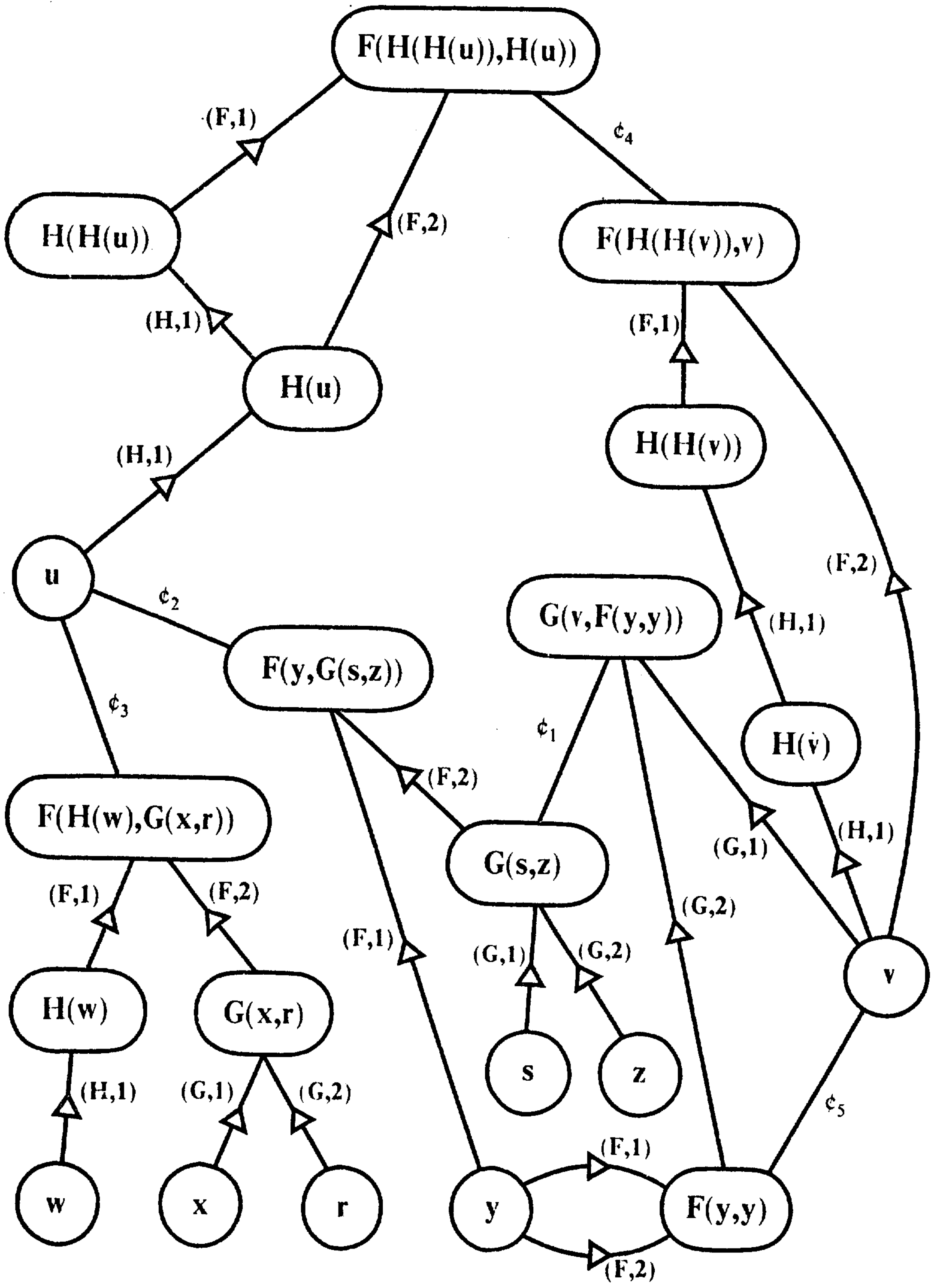
ϕ_4: $\{F(H(H(u)),H(u)),\ F(H(H(v)),v)\}$

ϕ_5: $\{v,\ F(y,y)\}$


u, v, s, z, x, r
{F(H(w),G(x,r)), F(y,G(s,z)), F(y,y)}
{H(u), H(v)}
H 7
G 2
F 1
H 6
{H(H(u)), H(H(v))}
y
{G(s,z), G(v,F(y,y)), G(x,r)}
{H(w)}
F 4
F 5
H 3
{F(H(H(u)),H(u)), F(H(H(v)),v)}
w

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 P.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 e_1,...etc.

<u>Figure 4.6</u>



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:

$$\begin{aligned} \mathrm{CONFLICT} = \{ & \{F(H(w),G(x,r)),\ H(u)\}, \\ & \{F(H(w),G(x,r)),\ H(v)\}, \\ & \{F(y,G(s,z)),\ H(u)\}, \\ & \{F(y,G(s,z)),\ H(v)\}, \\ & \{F(y,y),\ H(u)\}, \\ & \{F(y,y),\ H(v)\}, \\ & \{G(s,z),\ H(w)\}, \\ & \{G(v,F(y,y)),H(w)\}, \\ & \{G(x,r),\ H(w)\} \quad \} \end{aligned}$$

The set of cycles of U(C) is:

$$\mathrm{CIR} = \{ ([u],e_1,[y],e_2,[u]),\ ([u],e_7,[u]) \}$$

Cf the two possible "coverings" for CIR, we choose:

$$\mathrm{CCVER} = \{e_1,\ e_7\}$$

So the sets of states of A(C) which must be investigated for loops is:

$$\mathrm{TAIL}(e_1) = \{F(y,y),\ F(H(w),G(x,r)),\ F(y,G(s,z))\}$$

$$\mathrm{TAIL}(e_7) = \{H(u),\ H(v)\}$$

By investigating the automaton A(C) (figure 4.7) we obtain the following:

$$\mathrm{ATTACH}(F(H(w),G(x,r)),H(u)) = \{¢_3¢_4¢_4\}$$

$$\mathrm{ATTACH}(F(H(w),G(x,r)),H(v)) = \{¢_3¢_4¢_4¢_4\}$$

$$\mathrm{ATTACH}(F(y,G(s,z)),H(u)) = \{¢_2¢_4¢_4\}$$

$$\mathrm{ATTACH}(F(y,G(s,z)),H(v)) = \{¢_2¢_4¢_4¢_4\}$$

$$\text{ATTACH}(F(y,y),H(u)) = \{¢_4\}$$

$$\text{ATTACH}(F(y,y),H(v)) = \{¢_4¢_4\}$$

$$\text{ATTACH}(G(s,z),H(w)) = \{¢_2¢_4¢_5¢_5¢_4¢_3,\ ¢_2¢_4¢_5¢_2¢_3\}$$

$$\text{ATTACH}(G(v,F(y,y)),H(w)) = \{¢_3¢_2¢_5¢_4¢_2¢_1,\ ¢_3¢_4¢_5¢_5¢_4¢_2¢_1\}$$

$$\text{ATTACH}(G(x,r),H(w)) = \{¢_3¢_4¢_5¢_5¢_4¢_3,\ ¢_3¢_2¢_5¢_4¢_3\}$$

$$\text{LOOP}(F(y,y)) = \{¢_1¢_2¢_4¢_5\}$$

$$\text{LOOP}(F(H(w),G(x,r))) = \emptyset$$

$$\text{LOOP}(F(y,G(s,z))) = \{¢_2¢_4¢_5¢_1\}$$

$$\text{LOOP}(H(u)) = \{¢_5¢_1¢_2,\ ¢_1¢_2\}$$

$$\text{LOOP}(H(v)) = \{¢_4\}$$

Now consider the labelling [C,LBL] for C, where for all $¢ \in C$, $\text{LBL}(¢) = \{\ell\}$. Note that this labelling is not the cne 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 sirplest possible labelling for illustrative purposes.

We obtain the Eoolean sum of products over C:

$$B^1_{UNIF} = ¢_4¢_1 + ¢_4¢_2$$

Therefore the label set C has twc maximally correct subsets:

$$\{¢_2,\ ¢_3,\ ¢_5\}$$

and $\{¢_1,\ ¢_3,\ ¢_5\}$

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.

5.1: 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 $\phi \in C(G)$:

$$LBL(\phi) = \{e \mid \phi \in C(e)\}$$

The following description has an inductive structure to parallel the construction of plans.

Basis: If G is a basic plan, C(G) = Ø, 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 F.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 P.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 POP arcs. If ϕ = $\{p_1, p_2\}$ $\in$ C(e)−C(G), two new arcs (p_1, ϕ, p_2) and (p_2, ϕ, p_1) are added to TRANS(A(C(G))), and the label for ϕ is defined by:

$$LBL(\phi) \longleftarrow \{e\}$$

If $\phi \in C(e) \cap C(G)$, LBL(ϕ) is updated as follows:

$$LBL(\phi) \longleftarrow LBL(\phi) \cup \{e\}$$

(b) Execution of CLASSIFY is resumed at point C_1 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 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 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 B_{MAX} over SOL(G') as follows. If e' = (v,w) ∈ 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 $\underline{e}'$, the Boolean product $e'.e_1\ldots e_n$. Now we replace every e" in B'_{UNIF}, if it is in

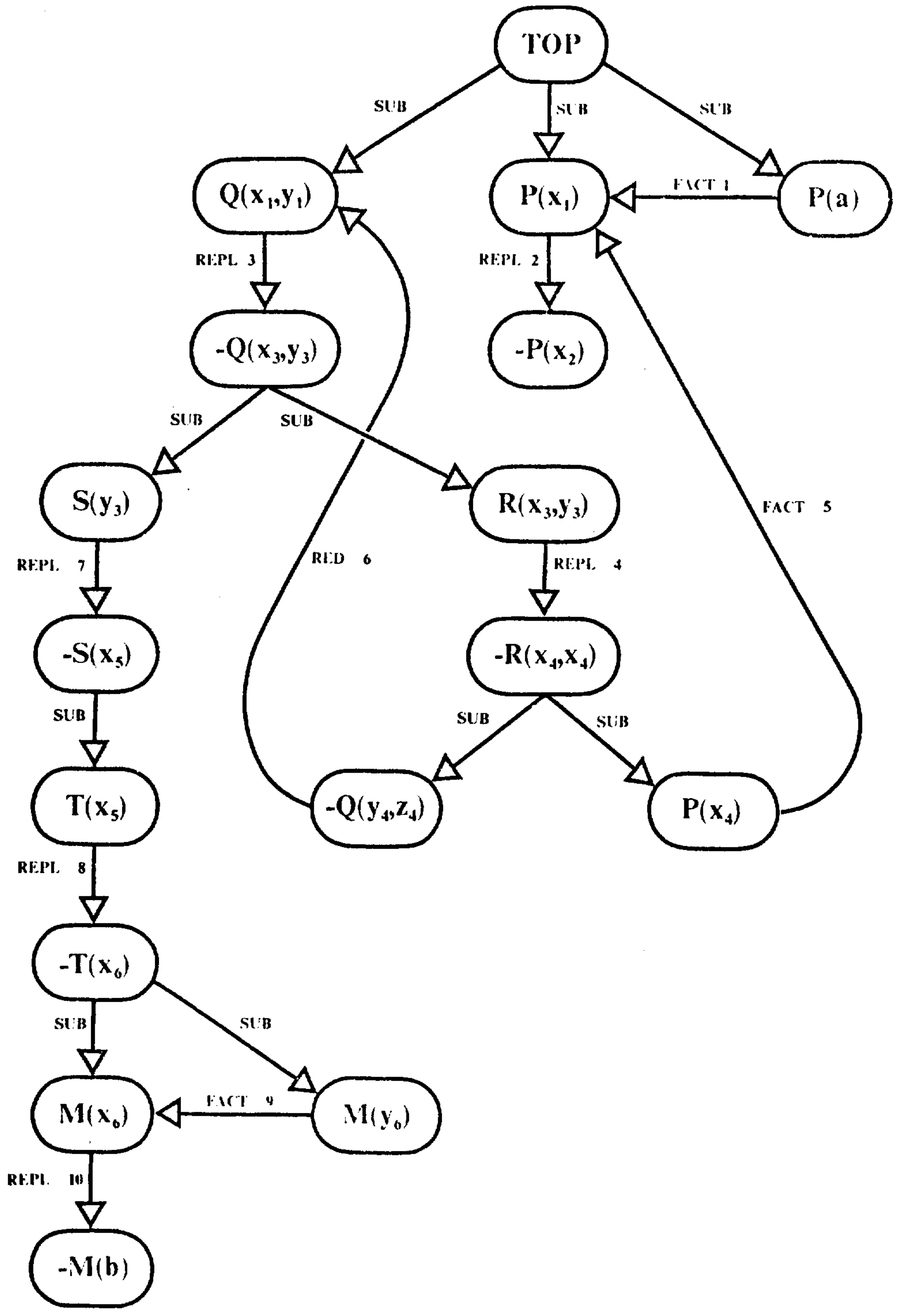
REPL(G'), by $\mathcal{S}''$; the resulting sum of products is B_{MAX}. Clearly, if we now remove from G' all the arcs in some product of B'_{MAX}, we obtain a correct subplan of G' which is maximal in the sense that it is not a subplan of any larger correct subplan. Note that since C(G) is unifiable and C(G') is not, G is one of the maximal correct subplans of G'; hence one of the products of B'_{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 $\mathcal{S}$ be the set of clauses:

$$\begin{aligned}\{\quad &\{Q(x,y),\ P(x),\ P(a)\},\\ &\{-P(x)\},\\ &\{-Q(x,y),\ S(y),\ R(x,y)\},\\ &\{-R(x,x),\ -Q(y,z),\ P(x)\},\\ &\{-S(x),\ T(x)\},\\ &\{-T(x),\ M(x),\ M(y)\},\\ &\{-M(b)\}\quad\}\end{aligned}$$

where a and b are constants. Figure 5.1 illustrates a closed plan G for $\mathcal{S}$, in which the arcs in 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.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

$¢_1$: $\{P(a), P(x_1)\}$

$¢_2$: $\{P(x_1), P(x_2)\}$

$¢_3$: $\{Q(x_1,y_1), Q(x_3,y_3)\}$

$¢_4$: $\{R(x_3,y_3), R(x_4,x_4)\}$

$¢_5$: $\{P(x_4, P(x_1)\}$

$¢_6$: $\{-Q(y_4,z_4), -Q(x_1,y_1)\}$

$¢_7$: $\{S(y_3), S(x_5)\}$

$¢_8$: $\{T(x_5), T(x_6)\}$

$¢_9$: $\{M(y_6), M(x_6)\}$

$¢_{10}$: $\{M(x_6), M(b)\}$

The constraint set C(G) for the plan G of figure 5.1

<u>Figure 5.2</u>

G is not correct since $a \equiv b \bmod F.out(C(G))$. Note that $LBL(¢_i) = i$ for all $¢_i \in C(G)$. By applying the procedure described above, we obtain:

$$B'_{UNIF} = 1 + 4 + 7 + 8 + 10 + 3.5$$

$$\therefore B_{MAX} = 1 + \underline{4} + \underline{7} + \underline{8} + 10 + \underline{3}.5$$

$$\text{where } \underline{4} = 4.5.6$$

$$\underline{7} = 7.8.9.10$$

$$\underline{8} = 8.9.10$$

$$\underline{3} = 3.4.5.6.7.8.9.10$$

$$\therefore B'_{MAX} = 1 + 4.5.6 + 10$$

$\{P(a),\ P(x_1),\ P(x_2),\ P(x_4)\}$

$\{Q(x_1,y_1),\ Q(x_3,y_3),\ Q(y_4,z_4)\}$

$\{-Q(y_4,z_4),\ -Q(x_1,y_1)\}$

$\{R(x_3,y_3),\ R(x_4,x_4)\}$

$\{S(y_3),\ S(x_5)\}$

$\{T(x_5),\ T(x_6)\}$

$\{M(y_6),\ M(x_6),\ M(b)\}$

$\{x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ y_1,\ y_3,\ y_4,\ y_6,\ z_4,\ a,\ b\}$

The partition F.out(C(G)) for the constraint set C(G) of figure 5.2

<u>Figure 5.3</u>

Since $M(x_6)$ has no other sclution than that represented by the arc 10, we will not backtrack by removing this arc. This leaves twc 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 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 Zamov 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: <u>Example</u>: Consider the set of clauses $\mathcal{S}$ 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), -Q(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 $\mathcal{S}$ 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) top

(8) Q(a,y), P(a) fact

(9)	Q(a,y), $\boxed{P(a)}$	ext(2)
(10)	Q(a,y)	cont
(11)	$\boxed{Q(a,y)}$, S(y), R(a,y)	ext(3)
(12)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, -Q(w,z), P(a)	ext(4)
(13)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, -Q(w,z), $\boxed{P(a)}$	ext(2)
(14)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, -Q(w,z)	cont
(15)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$	red
(16)	$\boxed{Q(a,a)}$, S(a)	cont
(17)	$\boxed{Q(a,a)}$, $\boxed{S(a)}$, T(a)	ext(5)
(18)	$\boxed{Q(a,a)}$, $\boxed{S(a)}$, $\boxed{T(a)}$, M(a), M(y)	ext(6)
(19)	$\boxed{Q(a,a)}$, $\boxed{S(a)}$, $\boxed{T(a)}$, M(a)	fact
	backtrack to (18)	
(20)	$\boxed{Q(a,a)}$, $\boxed{S(a)}$, $\boxed{T(a)}$, M(a), $\boxed{M(b)}$	ext(7)
(21)	$\boxed{Q(a,a)}$, $\boxed{S(a)}$, $\boxed{T(a)}$, M(a)	cont
	backtrack to (14)	
(22)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, $\boxed{-Q(w,z)}$, P(w), P(a)	ext(1)
(23)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, $\boxed{-Q(a,z)}$, P(a)	fact
(24)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, $\boxed{-Q(a,z)}$, $\boxed{P(a)}$	ext(2)
(25)	$\boxed{Q(a,a)}$, S(a), $\boxed{R(a,a)}$, $\boxed{-Q(a,z)}$	cont
(26)	= (15)	cont
(27)	= (16)	cont
(28)	= (17)	ext(5)
(29)	= (18)	ext(6)
(30)	= (19)	fact
	backtrack to (29)	
(31)	= (20)	ext(7)
(32)	= (21)	cont

backtrack to (22)

(33)	$\boxed{Q(a,a)}$, $S(a)$, $\boxed{R(a,a)}$, $\boxed{-Q(w,z)}$, $P(w)$, $\boxed{P(a)}$	ext(2)
(34)	$\boxed{Q(a,a)}$, $S(a)$, $\boxed{R(a,a)}$, $\boxed{-Q(w,z)}$, $P(w)$	cont
(35)	$\boxed{Q(a,a)}$, $S(a)$, $\boxed{R(a,a)}$, $\boxed{-Q(w,z)}$, $\boxed{P(w)}$	ext(2)
(36)	$\boxed{Q(a,a)}$, $S(a)$, $\boxed{R(a,a)}$, $\boxed{-Q(w,z)}$	cont
(37)	= (15)	cont
(38)	= (16)	cont
(39)	= (17)	ext(5)
(40)	= (18)	ext(6)
(41)	= (19)	fact

backtrack to (40)

(42)	= (20)	ext(7)
(43)	= (21)	cont

backtrack to (1)

(44)	$Q(x,y)$, $P(x)$, $\boxed{P(a)}$	ext(2)
(45)	$Q(x,y)$, $P(x)$	cont
(46)	$Q(x,y)$, $\boxed{P(x)}$	ext(2)
(47)	$Q(x,y)$	cont
(48)	$\boxed{Q(x,y)}$, $S(y)$, $R(x,y)$	ext(3)
(49)	$\boxed{Q(x,x)}$, $S(x)$, $\boxed{R(x,x)}$, $-Q(w,z)$, $P(x)$	ext(4)
(50)	$\boxed{Q(x,x)}$, $S(x)$, $\boxed{R(x,x)}$, $-Q(w,z)$, $\boxed{P(x)}$	ext(2)
(51)	$\boxed{Q(x,x)}$, $S(x)$, $\boxed{R(x,x)}$, $-Q(w,z)$	cont
(52)	$\boxed{Q(x,x)}$, $S(x)$, $\boxed{R(x,x)}$	red
(53)	$\boxed{Q(x,x)}$, $S(x)$	cont
(54)	$\boxed{Q(x,x)}$, $\boxed{S(x)}$, $T(x)$	ext(5)
(55)	$\boxed{Q(x,x)}$, $\boxed{S(x)}$, $\boxed{T(x)}$, $M(x)$, $M(y)$	ext(6)
(56)	$\boxed{Q(x,x)}$, $\boxed{S(x)}$, $\boxed{T(x)}$, $M(x)$	red

(57)	$\boxed{[Q(b,b)]}$,$\boxed{[S(b)]}$,$\boxed{[T(b)]}$,$\boxed{[M(b)]}$	ext(7)
(58)	$\boxed{[Q(b,b)]}$,$\boxed{[S(b)]}$,$\boxed{[T(b)]}$	cont
(59)	$\boxed{[Q(b,b)]}$,$\boxed{[S(b)]}$	cont
(60)	$\boxed{[Q(b,b)]}$	cont
(61)	□	cont

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 $\mathcal{S}$, 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 $\mathcal{S}$, 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.

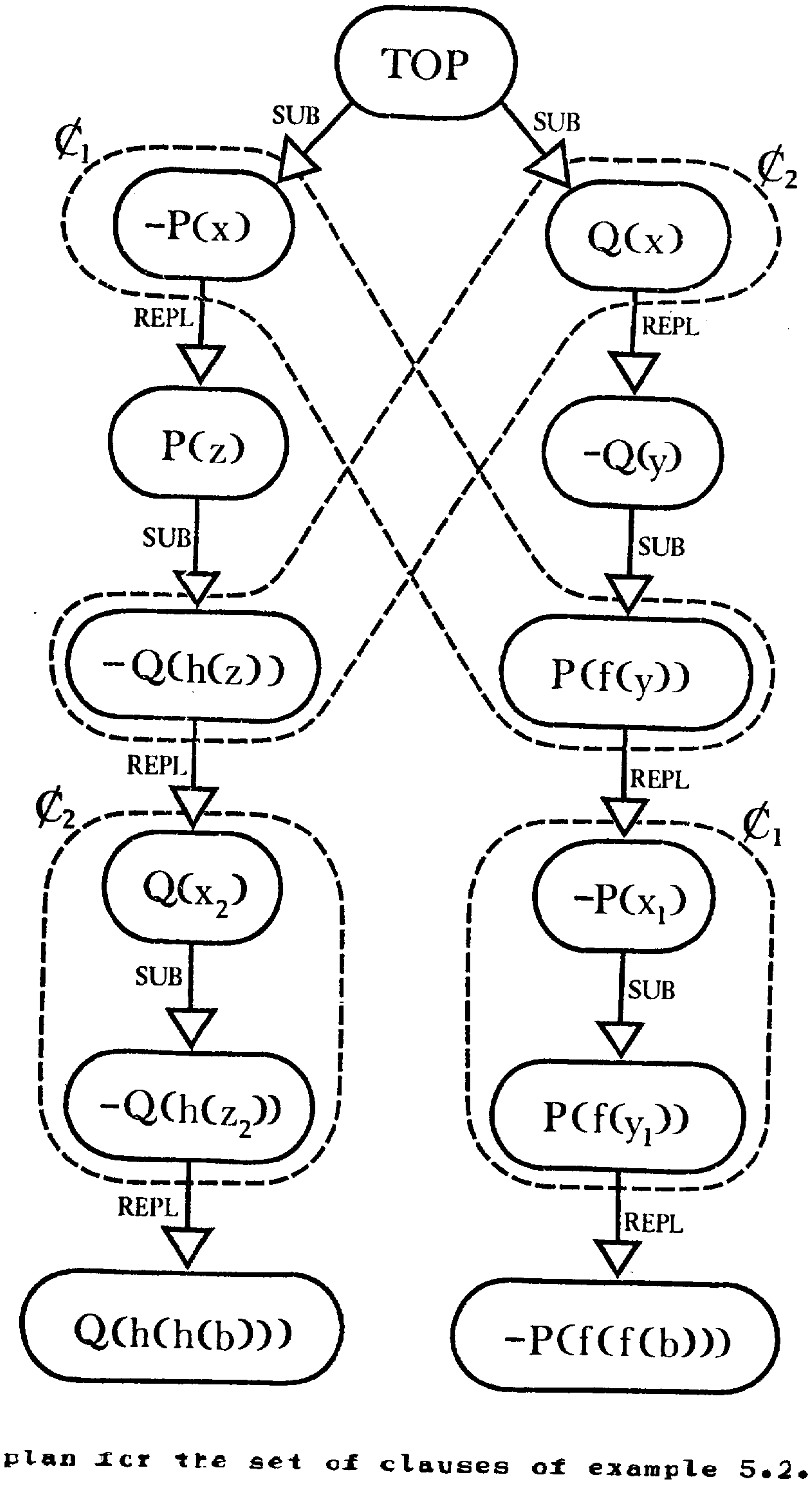
5.2.2: Example: Let $\mathcal{S}$ be the set of clauses:

$$\{\ \{-P(x),\ Q(x)\},\ \{-Q(y),\ P(f(y))\},\ \{P(z),\ -Q(h(z))\},\ \{-P(f(f(b)))\},\ \{Q(h(h(b)))\}\ \}$$

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 Huet'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 $\mathcal{S}$, it is necessary to build a clause graph for $\mathcal{S}$ 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 _refutation graph_ for the set of clauses.

The _resolution graphs_ of Yates, Raphael and Hart [43] are quite similar to Shostak's graphs.

In [39] Sickel describes _clause interconnectivity graphs_ which are identical to Kowalski's _connection graphs_ [22]. The vertices of these graphs are the literals of the clauses in the set $\mathcal{S}$ 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).

[2] Battani G. and Meloni H.
Interpreteur de language de programmation PROLOG
Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Marseille (1973).

[3] Baxter L.D.
An Efficient Unification Algorithm
Research Report CS-73-23, Department of Computer Science, University of Waterloo (1973).

[4] Baxter L.D.
A Practically Linear Unification Algorithm
Research Report CS-76-13, Department of Computer Science, University of Waterloo (1976).

[5] Baxter L.D.
The Complexity of Unification
Ph.D. Thesis, Department of Computer Science, University of Waterloo (1976).

[6] Bergman M. and Kanoui H.
Application of Mechanical Theorem Proving to Symbolic Calculus
Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Marseille (1973).

[7] Bondy J.A. and Murty U.S.R.
Graph Theory with Applications
MacMillan (1976).

[8] Boyer R.S. and Moore J.S.
The sharing of structure in theorem-proving programs
in Machine Intelligence 7, 101-116, John Wiley and Sons (1972).

[9] Brzozowski J.A. and Yoeli M.
Digital Networks
Prentice-Hall (1976).

[10] Chang C. and Lee R.C.
Symbolic Logic and Mechanical Theorem Proving
Academic Press (1973).

[11] Colmerauer A., Kanoui H., Pasèro R. and Roussel P.
Un système de communication homme-machine en francais
Groupe d'Intelligence Artificielle, U.E.R. Luminy, Marseilles (1972).

[12] Cox P.T. and Pietrzykowski T.
A Graphical Deduction 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).

[15] Hewitt C.
PLANNER: A language for proving theorems in robots
Proc. IJCAI, 295-302 (1969).

[16] Huet 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, Pennsylvania State University (1973).

[18] Kanoui H.
Application de la demonstration automatique aux manipulations algébrique et à l'integration formelle sur ordinateur
Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Marseille (1973).

[19] Knuth D.E.
The Art of Computer Programming, Volume I: Fundamental Algorithms
Addison-Wesley (1968).

[20] Kowalski R.A.
Predicate logic as a programming language
Proc. IFIP, 569-574 (1974).

[21] Kowalski R.A.
<u>Logic for problem-solving</u>
DCL Memo 75, Department of Artificial Intelligence, University of Edinburgh (1974).

[22] Kowalski R.A.
A <u>proof procedure using connection graphs</u>
J.ACM 22, no. 4, 512-595 (1975).

[23] Kowalski R.A. and Kuehner D.
<u>Linear resolution with selection function</u>
Artificial Intelligence 2, 227-260 (1971).

[24] Loveland D.W.
<u>Mechanical theorem-proving by model elimination</u>
J.ACM 15, no. 2, 236-251 (1968).

[25] Loveland D.W.
A <u>simplified format for the model elimination theorem-proving procedure</u>
J.ACM 16, no. 3, 349-363 (1969).

[26] Loveland D.W.
A <u>linear format for resolution</u>
Proc. IRIA Symp. Auto. Demon., 147-162, Springer-Verlag (1970).

[27] Loveland D.W.
A <u>unifying view of some linear herbrand procedures</u>
J.ACM 19, no. 2, 366-384 (1972).

[28] Luckham D.
<u>Refinements in resolution theory</u>
Proc. IRIA Symp. Auto. Demon., 163-190, Springer-Verlag (1970).

[29] Manna Z. and Waldinger R.
<u>The Logic of Computer Programming</u>
to appear in Computing Surveys

[30] Nilsson N.J.
<u>Problem Solving Methods in Artificial Intelligence</u>
McGraw-Hill (1971).

[31] Nilsson N.J. and Fikes R.E.
<u>STRIPS</u>: A <u>new approach to the application of theorem proving to problem solving</u>
Technical note 43, Artificial Intelligence Group, Stanford Research Institute (1970).

[32] Pasèro R.
Representation du francais en logique du premier orde en vue de dialoguer avec un ordinateur
Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Marseille (1973).

[33] Paterson M.S. and Wegman M.N.
Linear Unification
Proc. Symp. on Theory of Computing, SIGACT (1976).

[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 PROLOG
M.Math. thesis, Department of Computer Science, University of Waterloo (1977).

[36] Robinson J.A.
A machine oriented logic based on the resolution principle
J.ACM 12, no. 1, 23-41 (1965).

[37] Rulifson J.F.
QA4 programming concepts
Technical note 60, Artificial Intelligence Group, Stanford Research Institute (1971).

[38] Shostak R.E.
Refutation Graphs
Artificial Intelligence 7, 51-64 (1976).

[39] Sickel S.
A Search Technique for Clause Interconnectivity Graphs
IEEE Transactions on Computers vol.C-25, no.8 (1976).

[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 M.
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.H.D.
WARPLAN: a system for generating plans
DCL Memo 76, Department of Artificial Intelligence, University of Edinburgh (1974).

[43] Yates R., Raphael B. and Hart T.
Resolution Graphs
Artificial Intelligence 1, 224-239 (1970).

[44] Zamov N.K. and Sharonov V.I.
On a class of strategies which can be used to establish decidability by the resolution principle
Issled, po konstrukivnoye matematikye i matematicteskoie logikye v.3 no.16, 54-64 (1969).