

Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

A DEFINITIONALLY EXTENDIBLE TYPE-LOGIC
FOR MECHANICAL THEOREM PROVING

by

George Horace Woodmansee

Technical Report #121

March 1971

A Definitionally Extendible Type-Logic
For Mechanical Theorem Proving

A thesis submitted to the Graduate School of
the University of Wisconsin in partial fulfillment
of the requirements for the degree of Doctor of
Philosophy.

by

George Horace Woodmansee

Degree to be awarded

January 19~~70~~⁷¹

June 19—

August 19—

A DEFINITIONALLY EXTENDIBLE TYPE-LOGIC FOR
MECHANICAL THEOREM PROVING

G. H. Woodmansee

Under the supervision of Associate Professor Larry E. Travis

If a mechanical theorem prover is to be utilized as the basis of a mathematically oriented question answering system which is applicable to arbitrary user-specified theories it is desirable that the theorem prover have direct means of handling concepts peculiar to the user's theory. Current work in mechanical theorem proving has regarded the problem of application as extra-logical and has concentrated on producing proof procedures which are adaptable to particular theories only insofar as axioms specifying properties of primitive concepts of the theory can be introduced along with expressions to be proved.

In this thesis, we take the view that the logic, which underlies a general purpose mechanical theorem prover, must be responsive to the peculiarities of a given theory and should provide a framework in which strategies, which manipulate the inference rules of the logic, may take advantage of the state of development of the theory. Specifically, we propose a refutation-type-logic \bar{E} which extends the first order Analytic Tableaux of R. M. Smullyan to a higher order type-language similar to the B and C languages of Carnap. This logic provides for the introduction of defined constants and

can utilize derived rules of inference computed from defined constants, axioms and previously established theorems. Unabbreviation of subexpressions containing defined constants is treated as a primitive rule of inference of the logic, thus making it possible for strategies which utilize E to take advantage of the semantic structure of an expression which is implied by occurrences of defined constants within the expression.

Quantification is not explicitly represented in E , but, as in the case of many current systems, occurs implicitly in the form of variable dependencies. These dependencies are explicitly represented in two ways: via Skolem functions and by structures called dependency forests. Useful instances of a variable are ultimately determined by a unification procedure (similar to that of J. A. Robinson) which utilizes the Skolem function representation of variable dependencies. Skolem function representations of variable dependencies are calculated from dependency forests which are utilized by the primitive unabbreviation rule of inference (and a rule of inference called λ -reduction) to calculate dependencies between variables occurring in a defined constant's definition and variables in whose scope the defined constant occurs.

Dependency forests allow Skolem functions to be introduced into an expression at arbitrary points in the course of a proof any time new dependencies are uncovered by unabbreviation. (Such introduction is automatically taken care of by the inference operators of E .) It is thus possible for a strategy to incrementally (and selectively)

unabbreviate a given expression. Such a capability is not only a user convenience in the sense that inclusion of defined constants allows concise representation of concepts peculiar to the user's theory, but can be strategically important since the degree of unabbreviation can be controlled by the strategy and thus sensitive to the state of development of the theory of which the expression being processed is a part. The logic \bar{E} and associated notions thus provide a framework in which adaptive strategies may operate.

Finally, as a demonstration of the generality of \bar{E} , we show how four proof procedures (due to Prawitz, Robinson, Loveland and Friedman respectively) for the first order predicate calculus may be simulated within the framework of \bar{E} . This raises the possibility that mixed strategies for \bar{E} , which utilize current first order strategies as subparts, may be written.

Approved by:

Larry E Travis

ACKNOWLEDGEMENTS

I wish to express my gratitude to a number of people who either directly or indirectly have influenced the shape of this work.

To Professor Larry E. Travis for guidance and encouragement.

To Professor Edward F. Moore for advice, insights and (on occasion) anecdotes.

To Professor Donald R. Fitzwater for perspective.

To Stu Shapiro and Phyllis Roney my comrades in arms for encouragement, moral support, succor and four years of shared interests.

To Pat Hanson for typing and art-work above and beyond the call of duty.

Finally, and especially to my wife Louise whose many sacrifices made the whole thing possible.

The research reported herein was partially supported by a grant from the National Science Foundation (GP-7069).

TABLE OF CONTENTS

PREFACE.....		ii
ACKNOWLEDGEMENTS.....		v
CHAPTER I	Introduction.....	1
SECTION	1.0 The current state of mechanical theorem proving.....	1
	1.1 Some observations concerning the inadequacy of modularity and non-adaptive strategies.....	5
	1.2 The nature and content of this paper.....	8
CHAPTER II	Specification of a definitionally extendable type- logic.....	10
SECTION	1.0 Trees and forests.....	11
	1.1 Trees with content.....	11
	1.2 Further tree-related terminology (parts and images).....	14
	1.3 Further tree-related terminology (tree operations).....	16
	1.4 Forests and forest-related terminology.....	18
SECTION	2.0 The unextended system U	21
	2.1 The language U_L	23
	2.1.1 Language (syntax).....	23
	2.1.2 Substitutions, instantiation and unification.....	40
	2.1.3 Language (semantics).....	43
	2.2 Inference operators of U	52
SECTION	3.0 The extended system E	89
	3.1 Definitional extension of an object language...	91

3.2	The language E_L	95
3.3	Inference operators of E	107
3.4	Derived rules of inference and theorem utilization in E	111
3.4.1	Calculation of truth-function analysis trees for defined constants.....	115
3.4.2	Utilizing theorems in E	131
CHAPTER III	Defined rules of inference.....	139
SECTION 1.0	Introduction.....	139
SECTION 2.0	Simulation of selected proof procedures.....	153
2.1	A defined rule of inference which simulates a proof procedure of Prawitz.....	154
2.2	Simulated Binary Resolution.....	160
2.3	Simulated Model Elimination.....	173
2.4	Simulation of Friedman's decision procedure for the prefix $(\exists y_1)(\exists y_2)(\forall z_1)\dots(\forall z_n)$	187
CHAPTER IV	Concluding remarks.....	204
APPENDIX		
A	Implementation of defined rules of inference....	208
B	Functions related to unification.....	225
C	Image of a node in a descendant tree.....	227
D	Definition of the function ζ	230
E	Abbreviation and symbol glossary.....	235
F	Subject index.....	238
G	Alphabetic listing and description of additional DELS functions.....	240

H	Primitive effector operations of DELS.....	245
I	Proofs of the truth preserving properties of selected operators.....	246
BIBLIOGRAPHY.....		254



CHAPTER I (INTRODUCTION)1.0 The current state of mechanical theorem proving
modularity

Historically, mechanical theorem proving research has been concerned with the problem of constructing general purpose proof procedures for areas amenable to mathematical representation and manipulation. Such procedures must embody formal systems of reasoning which possess expressive power sufficient to allow formulation of arbitrary mathematical concepts. The early observation by Hilbert that all of classical mathematics could be formalized within quantification theory has led to almost universal adoption of the first order predicate calculus for this purpose, the reasoning being that a general purpose first order proof procedure would also be a general purpose mathematical proof procedure. This view neatly splits the general mechanical theorem proving problem in two, producing the subproblems of determining suitable translations from mathematics into the first order predicate calculus and constructing first order proof procedures. Current researchers, spurred on by this divide and conquer philosophy, have, almost without exception, addressed themselves to the latter task and more or less ignored the former.

Efficiency, perhaps the most important consideration after consistency, has further limited the form of the mechanical theorem proving problem as it is viewed today. It is reasoned that the more one can say concerning the form of an expression to be processed,

the better the chance for taking advantage of the particular syntactical characteristics of the form. This consideration and the fact that an arbitrary expression of the first order calculus has a canonical representation in a prenex normal form in which the quantifier-free matrix is in conjunctive normal form, has led to widespread adoption of this canonical form of input for first order proof procedures.

We may thus characterize much of the current mechanical theorem proving research as research concerned with devising efficient proof procedures for those formulas of the predicate calculus which are in the indicated canonical form. This is felt to be equivalent to devising efficient proof procedures for arbitrary areas of mathematics since any general purpose proof procedure may then be envisioned as consisting of a translator preprocessor and a core first order proof procedure. We shall call this conceptual partitioning of the general mechanical theorem proving problem into a preprocessing stage and a first order proof procedure stage, the modularity principle.

limited scope non-adaptive strategies

One may naturally focus attention on two aspects of any proof procedure: the underlying logic and the strategy which applies the logic. The efficiency of a proof procedure derives both from the efficiency of its logic and the efficiency of the strategy which it employs. Research in the late fifties and early sixties produced a

series of proof procedures each of which attempted to improve on the efficiency of its predecessors (mainly) by proposing more efficient strategies. Since each of these more or less directly implemented existing classical formulations of first order logic, each incurred inefficiencies due to the mismatch between the logic's formulation and the computer. Robinson [34] was the first to consider how the logic's formulation might be matched to the computer. His formulation, the resolution principle, was to prove a milestone in mechanical theorem proving research and has played a major role in shaping the field of mechanical theorem proving as it exists today.

At the time the original resolution paper appeared, the role played by a strategy in a proof procedure had been fairly well established. A strategy was an overall plan for applying the rules of the underlying logic to input expressions and their descendants with the goal of producing a proof. The global behavior of the strategy was essentially independent of the particular nature of the input expression. In essence, the scope of a strategy extended over one proof¹. It was thus impossible for a strategy to produce results applicable not only to the current expression, but which

¹This is not quite true for all systems under consideration. Certain systems allow a kind of minimal exploitation of previously proved results in the following sense: Suppose that $\bar{\Sigma}$ constitutes the set of axioms of some theory and that T_1, \dots, T_n are previously proved theorems of this theory, i.e. $\bar{\Sigma} \vdash T_i$ $i = 1, \dots, n$. Further suppose that we wish to demonstrate $\bar{\Sigma} \vdash A$ for some expression A . It then suffices to demonstrate $\beta \vdash A$ for some $\beta \subset \bar{\Sigma} \cup \{T_1, \dots, T_n\}$.

later could be applied to other input expressions. Thus, a strategy was conceived of as being essentially nonadaptive, and this meant that the proof procedure was no better prepared to prove the thousandth input expression than it had been prepared to prove the first².

Binary resolution and its progeny have, for the most part, accepted this conception of strategy³ and settled down to the task of devising resolution strategies which limit the potential combinatorial excursions of these clash oriented logics.

Thus, if we may characterize the mainstream of mechanical theorem proving research as it exists today, we see a field of ever newer and better resolution (based) proof procedures which are motivated by the modularity principle and embody strategies which are essentially non-adaptive and which make minimum use of previously proved results.

²At best, the systems under consideration utilize previous results in the manner outlined in footnote 1. Such systems must assume that a small set of relevant theorems has been supplied by some unanalyzed external agency or the procedure quickly bogs down under the sheer weight of irrelevant material. The question of relevancy, which is crucial if such a technique is to work, is considered to be a problem isolated from that of determining an efficient core procedure.

³The set of support strategy [49] probably comes closest to being adaptive. However, its success as an adaptive system depends on the relevancy of the set of support and thus the comments of footnote 2 apply.

1.1 Some observations concerning the inadequacy of modularity and non-adaptive strategies

Now, if we can produce acceptably efficient proof procedures, which are appropriate for arbitrary branches of mathematics, using the modularity principle and limited scope non-adaptive strategies, then the preceding considerations are of little concern. But can we? Is there any strong evidence that efficient general purpose proof procedures are being produced (or can be produced) using the modularity principle and non-adaptive strategies? I would argue there is none. For the current state of proof procedure generation has produced programs which have been applied only to toy problems and which produce as indicators of their worth relative rather than absolute measures of efficiency⁴ and the somewhat irrelevant mantle of completeness⁵. Such measures shed relatively little light on whether or not continued relative improvement of this sort will

⁴usually of the form ... this procedure is more efficient than its predecessors because in a given situation it applies operation x fewer times ...

⁵Completeness, a useful concept in formal logic, is somewhat irrelevant for the purposes of mechanical theorem proving since programs which implement first order proof procedures are never complete in practice. Furthermore, the fact that a particular procedure is incomplete does not necessarily mean that it is not useful. (Consider for example Friedman's semi-decision procedure [10] which is incomplete for the entire first order logic.) Completeness guarantees nothing concerning the amount of effort required to prove a given result. (For certain early systems, which uniformly instantiated the Herbrand universe, such effort quickly became excessive even for trivial proofs. (cf. Robinson 1963))

eventually lead to the desired end. Furthermore, we may argue that for a proof procedure to be acceptably efficient, it must on the average, expend only a modest amount of its resources in proving valid arbitrary formulas of modest difficulty. But, the difficulty of establishing a given result depends to a large extent on the state of development of the theory of which the expression is a part. Since limited scope non-adaptive strategies make minimal use of contextual information, we might expect expressions of even a moderately advanced theory to be excessive resource consumers because the proof procedure must, in a certain sense, start from scratch each time it encounters a new expression.

Now, it could be argued that even if some sort of adaptability is necessary for the efficient operation of a proof procedure, it could be handled within the framework of modularity. (That is, one can envision the proof procedure as consisting of a core proof procedure and a hybrid preprocessor, which in addition to translating from mathematics to canonical form predicate calculus, also reflects the current state of the theory to which the stream of input expressions belong and which can take advantage of it by supplying this information to the non-adaptive procedure.) I would argue that this is not the case. From a general point of view, such a division makes the preprocessor a major component of the total proof procedure with its own structures and strategy. In addition, since the efficiency of the whole is not necessarily equal to the sum of the efficiencies of its parts, these structures

and strategy must be matched to the core procedure and the core procedure to them. Thus the proof procedure must be designed as a whole. This philosophy is at odds with the principle of modularity which requires that the core procedure be designed totally out of context. This is basically an argument about problems associated with interfacing autonomous procedures. However, problems associated with modularity run deeper than the interface between the pre-processor and the core procedure, for the sort of adaptability represented by inclusion of defined constants requires that the core procedure's strategy have an incremental unabbreviation capability⁶ which is only possible if the core procedure can deal directly with expressions containing abbreviations.

⁶It is common practice in mathematics and other fields to introduce abbreviations for frequently used or important concepts. The set of defined constants used in abbreviations reflect certain aspects of the stage of development of a theory and thus constitute adaptive information. As a particular theory expands and new abbreviations are found to be necessary, statements of the theory tend to include deeply nested abbreviations. A statement of a moderately advanced theory may thus be highly stratified with each successive level of abbreviation obscuring the structure of the previous level until unabbreviation brings it into view. In terms of primitive concepts the obscured structure might be highly complex. Additionally, the very nature of an abbreviation - i.e. user supplied shorthand for important concepts of the theory - tends to make the abbreviational structure strategically significant.

Now, if we wish to allow abbreviations in a system which obeys the modularity principle, unabbreviation must be total and handled as part of the preprocessing operation. This means that even though the canonical representation obtained by preprocessing is logically equivalent to the input expressions, the strategic hints implicit in the abbreviational structure are lost. In addition, the inferential portion of the core procedure is confronted with the entire complex structure of the expression rather than the series of relatively simple structures that abbreviation affords.

1.2 The nature and content of this paper

The advent of resolution constituted recognition of the fact that a proof procedure's underlying logic must reflect its execution context, i.e. that it is to be applied by a machine rather than a human. This paper takes the view that the logic must also reflect the fact that input expressions are not isolated entities, but rather part of some structured body of knowledge.

Specifically, we here develop a system of logic and a notion of strategy relative to this logic which can make direct use of certain types of adaptive information. The logic provides for introduction of defined constants and can utilize derived rules of inference using defined constants, axioms and previously established theorems. Unabbreviation of expressions containing defined constants is treated as a primitive rule of inference of the logic rather than a preprocessing step. It is thus possible for proof generating strategies to exploit the occurrence of defined constants in the expression.

Within the framework of the logic presented in this thesis, it is also possible to simulate current first-order strategies such as [10], [19], [30] and [34].

In the appendix, we suggest a language for writing strategies which utilize the logic. The notion of strategy relative to this language and the proposed logic is sufficiently general to allow strategies to be written which are closed and total (i.e. require no human intervention and have as goal, production of a proof of

the input expression) or open and partial (i.e. require human intervention and have as goal some useful intermediate result such as producing all resolvents of some pair of clauses).

CHAPTER II SPECIFICATION OF A DEFINITIONALLY EXTENDIBLE TYPE-LOGICIntroduction

The definitionally extendible type-logic \bar{E} specified in this chapter, is based on a primitive system U and various notions related to the concepts of trees and forests. The presentation is divided into three major sections: A section which provides the necessary material on trees, a section which presents the unextended system U upon which \bar{E} is based, and a section which presents the extendible logic \bar{E} . We begin with the notions of trees and forests.

1.0 Trees and Forests

Trees and forests play an important role both in the unextended system U and in its extension E . For this reason, it is useful to have a precise definition of tree-related terminology at hand for future reference.

1.1 Trees with content

Both U and E utilize trees to structurally represent the consequences of truth value assignments to sentences of the logic. In this application, it must be possible to associate certain types of information with the nodes of any given tree. Trees with content will serve in this capacity and are introduced in terms of the more general notions of tree and ordered tree.

tree A tree t consists of the following:

1. A set of elements N_t called nodes.
2. A binary relation $P_t(n,m)$ defined in $N_t \times N_t$ which is read "n is the immediate predecessor of m" or "m is the immediate successor of n." Furthermore, P_t satisfies the following conditions:
 - a) There is a unique node n , called the origin and denoted by " $\text{org}(t)$," which has no immediate predecessor.
 - b) Excepting the origin, every node has a unique immediate predecessor.

- c) Excepting the origin, $k \in N_t$ implies there exist nodes $k_1, \dots, k_m \in N_t$ such that $k_1 = k$, $k_m =$ origin and k_{i+1} is the immediate predecessor of k_i for $1 \leq i < m$.

A node having no immediate successor is called a leaf. All other nodes including the origin are called internal nodes. A path is a finite or denumerable sequence of nodes, beginning with the origin and having the property that each term of the sequence (except the last if the sequence is finite) is the immediate predecessor of the next. A branch is a path connecting the origin and a leaf.

A node n will be said to be an ancestor of a node m (denoted $n < \cdot_t m$) if either 1. $P_t(n, m)$
 or 2. there exists node z such that $P_t(n, z)$
 and $z < \cdot_t m$

The specification of a tree induces a unique 1:1 correspondence between the leaves of the tree and branches of the tree on which they appear. Thus, if k is the leaf determining the branch b , we shall sometimes use the phrase "the branch k " to denote b .

If x is an ancestor of y and y an ancestor of z , then y will be said to be between x and z . In those cases where no confusion can arise, the subscripts denoting the tree's name will be dropped.

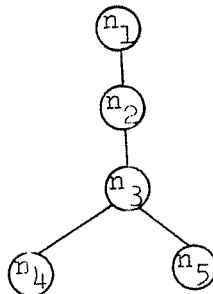
ordered tree An ordered tree t is a tree together with a function θ_t which assigns to each internal node an enumeration of its immediate successors. Given the tree, $\langle N_t, P_t \rangle$, an ordered tree t' is $\langle N_{t'}, P_{t'}, \theta_{t'} \rangle$ where $N_{t'} = N_t$ and $P_{t'} = P_t$. Furthermore, an ordered tree can be represented alternately as a geometric diagram with the origin at the top and immediate successor nodes below their immediate predecessors and ordered from left to right. Arbitrary trees may also be represented this way if the left-right order is ignored.

example t is $\langle N, P, \theta \rangle$ where:

$$N = \{n_1, n_2, n_3, n_4, n_5\}, \quad P = \{\langle n_1, n_2 \rangle, \langle n_2, n_3 \rangle, \langle n_3, n_4 \rangle, \langle n_3, n_5 \rangle\}$$

$$\theta = \{\langle n_1, \langle n_2 \rangle \rangle, \langle n_2, \langle n_3 \rangle \rangle, \langle n_3, \langle n_4, n_5 \rangle \rangle\}$$

$\text{org}(t) = n_1$, the leaves are n_4 and n_5 and the internal nodes are n_1, n_2, n_3 . The geometric representation is given by:



Notice that the names n_1, \dots, n_5 are redundant in the geometric diagram.

trees with contents from S A tree t with contents from a set S is a tree together with a function C_t on the nodes of the tree

into the set S . If $C_t(n) = x$ where $n \in N_t$ and $x \in S$ we say that x is the content of n . Trees with contents from S will be represented geometrically by displaying the content of a node near the node in the geometric representation. A tree t is said to contain an element x if there exists a node $n \in N_t$ such that $C_t(n) = x$. The tree is said to contain x once if there is one and only one n such that $C_t(n) = x$.

1.2 Further tree-related terminology (parts and images)

Various relationships between trees and tree parts are of interest in U and E . For the most part, these relationships are structural in the sense that they may be defined for trees or ordered trees and extended in an obvious manner to trees with content. Concepts arising from four such relationships are introduced below.

subtrees A tree \hat{t} is called a subtree of t if:

1. $N_{\hat{t}} \subset N_t$
- and 2. $P_{\hat{t}} \subset P_t$ with the proviso that $n, m \in N_{\hat{t}}$ and $P_t(n, m)$ implies $P_{\hat{t}}(n, m)$.

If \hat{t} is a subtree of the ordered tree t , then the relative order induced by t on \hat{t} is given by

$\theta_{\hat{t}} \subset \theta_t$ with $\theta_{\hat{t}}$ such that $n, m_i \in N_{\hat{t}}$ and $\langle n, \langle m_1, \dots, m_p \rangle \rangle \in \theta_t$ implies $\langle n, \langle m_1, \dots, m_p \rangle \rangle \in \theta_{\hat{t}}$

If t is a subtree of a tree with contents from S then a content function $C_{\hat{t}}$ may be obtained by suitably restricting the content

function of the parent tree. Unless otherwise noted, a subtree will be considered to be of the same type as the parent, i.e. if the parent tree is an ordered tree, then its subtrees will be considered ordered and if the parent tree is a tree with contents from S then its subtrees will be considered to be trees with contents from S .

major subtrees The major subtrees of a tree t are the subtrees of t whose origins comprise the set $\theta_t(\text{org}(t))$.

tree isomorphism Two trees t and \hat{t} will be said to be isomorphic if there exists a 1-1 mapping Ψ with the following properties:

1. Ψ maps N_t onto $N_{\hat{t}}$
2. If $n_1, n_2 \in N_t$, then $P(n_1, n_2)$ if and only if $P_{\hat{t}}(\Psi(n_1), \Psi(n_2))$

In addition, if t and \hat{t} are ordered and 3. holds

3. If $n, n_1, \dots, n_p \in N_t$, then $\langle n, \langle n_1, \dots, n_p \rangle \rangle \in \theta_t$ if and only if $\langle \Psi(n), \langle \Psi(n_1), \dots, \Psi(n_p) \rangle \rangle \in \theta_{\hat{t}}$

then t and \hat{t} are said to be isomorphic to within order (or order isomorphic).

image of t in s If t and s are trees and there exists a subtree \tilde{t} of t which is isomorphic or order isomorphic to s , then \tilde{t} is said to be an image of s in t .

1.3 Further tree-related terminology (tree operations)

In U and E we are particularly interested in certain operations which produce new trees by altering or merging the structures of existing trees. Three of these operations are introduced below. In practice they apply to trees with content, however, since the operations are basically structural and thus independent of node content, we shall define them for the somewhat more general concept of ordered trees.

appending Let t and s be two ordered trees having disjoint node sets. The ordered tree r is said to result from appending t to the leaf $n \in N_s$ if

$$N_r = N_s \cup (N_t - \{\text{org}(t)\})$$

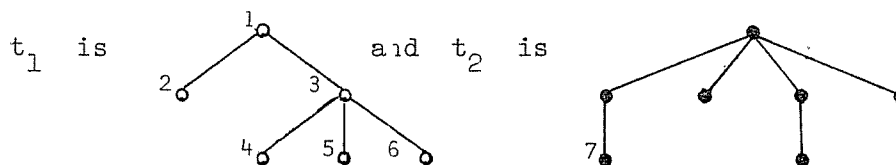
$$P_r = P_s \cup (P_t - \{\langle \text{org}(t), k \rangle : k \in Q\}) \cup \{\langle n, k \rangle : k \in Q\}$$

$$\theta_r = \theta_s \cup (\theta_t - \{\langle \text{org}(t), \langle k_1, \dots, k_i \rangle \rangle : k_j \in Q\}) \cup \{\langle n, \langle k_1, \dots, k_i \rangle \rangle : k_j \in Q\} \text{ where } Q = \{k : \langle \text{org}(t), k \rangle \in P_t\}$$

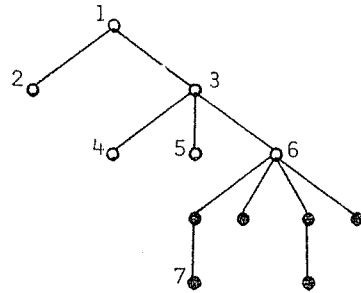
and cardinality of $Q = i$.

The operation of appending one tree to the leaf of another has a simple geometric interpretation which is illustrated in the following example:

example Let t_1 and t_2 be as given



Then the result of appending t_2 to the leaf 6 of t_1 is



Appending may be defined for trees with content by defining C_r as follows: $C_r = C_s \cup (C_t - \{\langle \text{org}(t), C_t(\text{org}(t)) \rangle\})$. This will be considered in detail later.

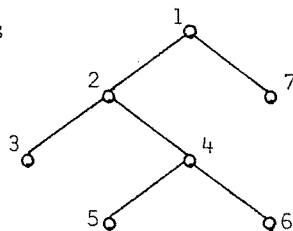
node removal Suppose $n \in N_t$, $n \neq \text{org}(t)$ and further m is the immediate predecessor of n and n_1, \dots, n_k are the immediate successors of n . The tree (ordered) q which results from removing the node n from t is defined as

$$N_q = N_t - \{n\}$$

$$P_q = (P_t - \{\langle m, n \rangle, \langle n, n_1 \rangle, \dots, \langle n, n_k \rangle\}) \cup \{\langle m, n_1 \rangle, \dots, \langle m, n_k \rangle\}$$

$$\theta_q = (\theta_t - (\{\langle n, \langle n_1, \dots, n_k \rangle \rangle\} \cup \{\langle m, \langle m_1, \dots, m_\ell \rangle \rangle\})) \cup \{\langle m, \langle m_1, \dots, n_1, \dots, n_k, \dots, m_\ell \rangle \rangle\}$$

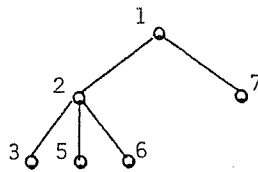
example If t is



and q is obtained

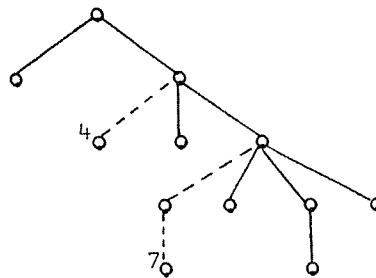
by removing node 4,

then q is



branch removal Suppose b is a branch consisting of the elements b_1, \dots, b_k (where b_1 is a leaf and b_{i+1} is the immediate predecessor of b_i). If b is not the only branch then the tree q resulting from the removal of b from the tree t is obtained by removing the nodes b_1, b_2, \dots, b_{m-1} where m is the smallest number $\leq k$ for which b_m has more than one immediate successor. (Since b_1 is a leaf, m is necessarily greater than 1.) If b is the only branch, then remove nodes b_1, \dots, b_k .

example The tree which results from the removal of the branches designated by leaves 4 and 7 in the append example, is illustrated below: (Broken lines indicate removed branch.)



1.4 Forests and forest related terminology

It is often useful to group trees according to some common property. Such groupings are called forests and occur in U and

\bar{E} as the result of performing certain operations on trees.

A forest is a set (possibly empty) of trees such that if s and t are in the set, then $N_t \cap N_s = \phi$. A forest will be said to contain an element x if there exists a tree in the forest which contains x . A forest will be said to contain x once if there exists a unique tree t which contains x and t contains x once. Each category of tree produces a corresponding category of forest. Thus, for example, a forest with contents in S is simply a forest consisting entirely of trees with contents in S .

If F is a forest, then N_F and P_F will denote the node set and immediate predecessor relationship respectively. $N_F = \bigcup_{t \in F} N_t$ and $P_F = \bigcup_{t \in F} P_t$.

The notions of isomorphism and image can be extended to forests in the obvious manner.

the forest F determined by the tree t and the node set \hat{N}

Suppose $t = \langle N, P, \theta, C \rangle$ and $\hat{N} \subset N$

1. \hat{C} is the restriction of C to \hat{N}
2. for each $n, m \in \hat{N}$ $\langle n, m \rangle \in \hat{P}$ if and only if

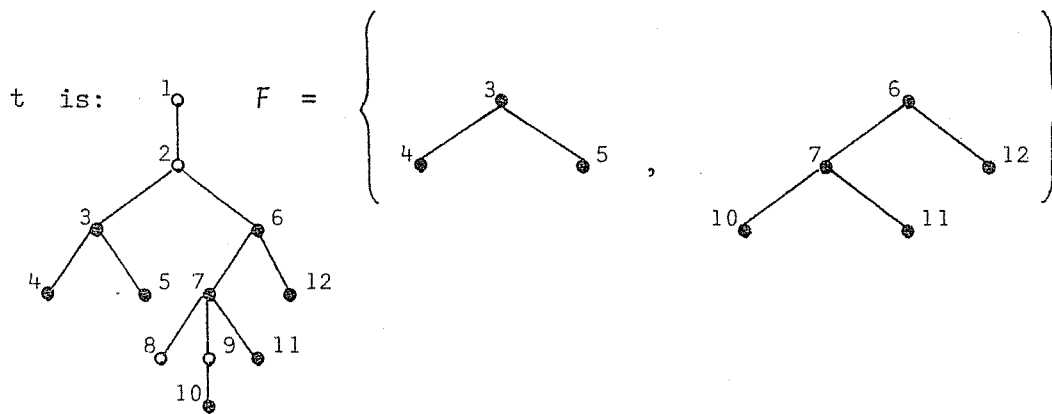
a) $n < \cdot_t m$

and b) there is no node $q \in \hat{N}$ between n and m .

$$F = \langle \hat{N}, \hat{P}, \hat{\theta}, \hat{C} \rangle$$

where $\hat{\theta}$ is the union of the relative orders induced by t on the subtrees of t belonging to $\langle \hat{N}, \hat{P}, \hat{C} \rangle$.

example Let N consist of the solid nodes.



2.0 The unextended system U (Introduction)

The unextended system U , presented in this section, consists of a language, and a set of inference operators. The presentation is divided into two major subsections: A language section which specifies the syntax and semantics of U and a theory section which specifies the ten inference operators of U and develops the notion of "provable in U ".

U_L , the language component of U , is based on the syntax and semantics of Carnap's "B and C languages" [3] and is a particular species of the higher-order languages known collectively as "type languages." These languages preserve the expressive power of higher-order languages while avoiding the logical paradoxes that can arise when a language's formation rules are not sufficiently restrictive.

U_L differs from the B and C languages in several respects. In U_L , predicates are considered to be a special kind of functor and expressions are formulated in terms of Sheffer stroke \oplus and \forall rather than \neg , \vee , $\&$, \supset , \equiv , \forall^1 . This latter difference allows the presentation to be somewhat less cluttered than would be the case if the traditional operators were used. The logical power of the system is not compromised by the smaller set of operators and the convenience afforded in practice by the larger

¹We shall follow the convention of allowing such object language symbols to stand for themselves. (cf. Carnap's concept of autonomous symbols. [3].)

set is recaptured in the extended system E since \neg , \vee , $\&$, \supset , \equiv can be considered defined constants of the extended system.

The inferential (or theory) portion of U is patterned after the analytic tableaux of R. M. Smullyan [43] which is a complete and consistent characterization of first-order logic and bears a family resemblance to the system of Beth [2], Hintikka [16] and, more remotely, Gentzen [12].

A deduction in U , consists of a series of trees whose nodes have as content single formulas with associated truth-values. The inference operators formalize the process of making explicit the various atomic truth-value assignments implied by a formula's associated truth-value. To prove a formula, one begins with a tree whose origin is a formula which is satisfiable if and only if the negation of the formula to be proved is satisfiable, and generates other trees using the inference operators of the system. If a special tree (called the empty tree) is produced, the initial formula is said to be provable. As with other logical systems, the syntactical notion of provability can be tied to the semantic notion of validity. Formulas that are provable in U are valid.

In certain cases, features of the unextended system, which are independent of the syntax of U_L , can be illustrated using examples from a language of the sort presented in Russell and Whitehead's *Principia Mathematica* [47]. Where possible, this will be done since such examples usually provide a more lucid illustration of the concepts under consideration than would examples involving expressions of U_L .

2.1 The language U_L

2.1.1 Language (syntax)

Sentences of U_L , are called polarized expressions and are constructed from a vocabulary of symbols according to a set of formation rules. These rules involve an intermediate class of objects (called expressions) which may be subcategorized according to the last formation rule applied in their construction. Expressions may be further subcategorized according to type.

type

1. 0 and 1 are types
 2. If $\tau_1, \dots, \tau_n, \tau$ are types, then $(\tau_1, \dots, \tau_n : \tau)$ is a type.
- "0" is the type of an individual, "1" the type of a truth-value and $(\tau_1, \dots, \tau_n : \tau)$ the type of a functor (function symbol) having arguments of type τ_1, \dots, τ_n and which, when applied to a full set of arguments of the correct type produces an object of type τ , otherwise non well formed.

symbols

1. For any type - a denumerable set of
 - a) variables
 - b) Skolem function symbols¹
2. The special symbols \bigoplus , \forall , \bigotimes , \bigotimes , λ , \bullet

¹Calling these symbols "Skolem function symbols" gives an indication of their intended use. As they stand, they are of course just distinguishable arbitrary symbols.

The class of variables (of all types) will be denoted as V , and the class of Skolem function symbols as S . Any element of $V \cup S \cup \{\phi\}$ will be called a type system symbol (TSS). It is assumed that V , S and the set of special symbols are pairwise disjoint.

formation rules (for expressions)

Expressions belonging to U_L are constructed according to the rules R1-R4 given below. If e is an expression formed according to these rules and R_i was the last rule invoked in its construction, e will be called an R_i expression. For each rule R_i , we shall assume an exclusion clause of the form, "Only expressions formed according to this rule are R_i expressions."

In addition to specifying the form of a given type of expression, we shall specify which parts of an expression are well formed² and under what conditions variables occurring in subparts of the expression become bound. These notions enter into the definitions of subexpression and variable dependencies given later.

R1 - Any TSS of type τ is an expression of type τ .

The type of \bigcirc is $(1,1:1)$.

R2 - If e is an expression of type $(\tau_1, \dots, \tau_n : \tau)$ and e_i is an expression of type τ_i , then $ee_1 \dots e_n$ is an expression of type τ .

²Arbitrary character substrings of an expression are not necessarily expressions. An analogous situation holds in the propositional calculus where the substring $) \& C$ of the formula $(A \vee B) \& C$ is not well formed.

The expressions e_1, \dots, e_n are said to be within the scope of e . The expression e is the operator and e_1, \dots, e_n the arguments. Bound³ variables occurring in e, e_1, \dots, e_n must be distinct⁴ and are considered to be bound in $ee_1 \dots e_n$.

R3 - If e is an expression of type τ , x_i a variable of type τ_i ($i = 1, \dots, n$) with the property that x_i is distinct from any x_j ($j \neq i, j = 1, \dots, n$) and from all bound variables occurring in e , then $\lambda x_1 \dots x_n \bullet e$ is an expression of type $(\tau_1, \dots, \tau_n : \tau)$. Each variable y which occurs bound in e and each x_i is bound in $\lambda x_1 \dots x_n \bullet e$.

³Rules R1-R4 constitute a recursive definition of "expression" and "bound." We note, for example, that a variable may become bound only through application of rules R3 and R4. Once bound, the rules insure that a variable remains bound in any expression constructed from the expression in which it is originally bound. Rules R3 and R4 provide a basis for determining whether a given variable is bound in an expression. Thus x is bound in the expression e if e is an R3 expression of the form $\lambda x_1 \dots x_n \bullet e$ or an R4 expression of the form $\forall x e_1$, or if x is bound in some constituent expression of e . Note that since a constituent expression is smaller than the containing expression, the recursion will eventually terminate. (We assume the exclusion clause "the only bound variables are those prescribed by rules R2-R4".)

⁴The requirement, that a set of variables S occurring in expressions a_1, \dots, a_k be distinct, can be satisfied by revising the variables. Thus, if v_1 and v_2 are the same variable (both occurring in S), then we choose a variable v not contained in S and replace all occurrences of v_2 in a_1, \dots, a_k by v . Note, the requirement that variables be distinct does not effect the expressive power of the language.

R4 - If e is an expression of type 1 and $v \in V$ and v is not bound in e , then $\forall v e$ is an expression of type 1. The expression e is said to be within the scope of V and variables occurring bound in e are bound in $\forall v e$.

Any variable which is not bound in an expression is said to be free in that expression. An expression which contains no free variables is said to be a closed expression.

Well formed parts of an expression are certain character substrings of the expression and will be used to define subexpression.

well formed part (See also expression definitions R1-R4.)

1. R1 expressions have no well formed parts.
2. R2 expressions have the well formed parts e, e_1, \dots, e_n .
3. R3 expressions have the well formed parts $\lambda, x_1, \dots, x_n, \bullet, e$.
4. R4 expressions have the well formed parts V, v, e .

subexpressions R1-R4 specify means by which expressions may be combined to form other expressions. Any expression which enters at some stage into the construction of another is said to be a subexpression of that expression. Precisely stated: An expression e_1 is a subexpression of an expression e if:

1. e_1 is a well formed part of e
- or 2. there exists an expression e_2 such that
 - a) e_2 is a well formed part of e
 - and b) e_1 is a subexpression of e_2 .

We note that in general the well formed parts λ , \forall , \bigoplus , and \bullet are not subexpressions. (Notice that the reason they do not satisfy the above definition is they are not expressions.)

We define an atomic expression as follows: An expression e is atomic if

1. it has type 1 and its first characters is not \bigoplus , \forall or λ and
2. it is not a subexpression of any other expression which also satisfies 1.
3. only expressions satisfying 1 and 2 are atomic.

This syntactic definition is motivated by the semantic consideration that an atomic subexpression of an expression e is a type 1 expression which is truth functionally unanalyzable. For those accustomed to working only with first order logics, it might seem more reasonable to define an atomic expression as a type 1 expression which has no type 1 subexpressions. Unfortunately, the nature of the type language U_L makes such a definition unsuitable. Consider, for example, the pe $\bigoplus \forall M \forall P \forall Q M \bigoplus P Q$ where P, Q have type 1 and M type (1:1). Under this latter definition, P and Q would be atomic expressions. However, the expression $M \bigoplus P Q$ is unanalysable in the sense that knowing it is false tells us nothing about the truth or falsity of P or Q . Under the former definition, $M \bigoplus P Q$ is the only atomic subexpression of $\bigoplus \forall M \forall P \forall Q M \bigoplus P Q$.

Expressions formed according to the rules given above provide the raw materials from which the sentences of U are formed. The

sentences of U are called polarized expressions and are syntactical forms which may be thought of as asserting or denying a state of affairs in some domain⁵ of discourse.

POL - polarized expression (pe) If e is a closed expression of type 1 which contains no elements of S , then:

1. $\textcircled{T}e$ is a polarized expression. \textcircled{T} and e are its well formed parts. \textcircled{T} is called the prefix and e the suffix.
2. $\textcircled{F}e$ is a polarized expression. \textcircled{F} and e are its well formed parts. \textcircled{F} is called the prefix and e the suffix.

Polarized expressions are not assigned types and thus can not further combine under R1-R4. The conjugate of a polarized expression $\textcircled{T}e$ ($\textcircled{F}e$) is the polarized expression $\textcircled{F}e$ ($\textcircled{T}e$). The subexpressions of a polarized expression are the subexpressions of its suffix.

Expressions formed according to R1-R4 have several desirable properties which make parsing possible. If e_1 and e_2 are expressions, then

⁵Later, we shall develop the notion of semantic evaluation. Semantic evaluation provides a mapping from the set of expressions into a set of objects called a derived domain. Under this mapping, polarized expressions are mapped into the subdomain $\{T, F\}$. The elements T and F may be thought of as truth and falsity. If D is an arbitrary non-empty set of objects and D^* is the derived domain corresponding to D , then we say that a pe is true relative to D if it is mapped into $T \in D^*$ and false if it is mapped into $F \in D^*$.

1. they only have a finite number of subexpressions
- and
2. If e_1 and e_2 have the same type and are both R_i expressions (for the same i), then it is decidable if their well formed parts correspond. If they do, there exists a unique correspondence (1 \leftrightarrow 1) and this correspondence can be computed. (In particular, the well formed parts of e_1 and e_2 will correspond if and only if they agree in number, type and order in the representing string, i.e. the i^{th} well formed part of e_1 must have the same type as the i^{th} well formed part of e_2 .)

It follows as the direct result of these properties that any polarized expression has a unique derivation in terms of R_1 , R_2 , R_3 , R_4 and POL . This and the fact that subexpressions of corresponding expressions correspond makes a unification⁶ algorithm for U possible as well as allowing the kind of decompositional inference rules which are presented in 2.1.3.

A polarized expression containing quantifiers may impose complex dependencies among its variables which must be taken into account any time an inference rule involving instantiation of a variable is applied to the polarized expression. As an extremely

⁶If x_1, \dots, x_n are variables contained in the expressions e_1 and e_2 and if a_1, \dots, a_n are expressions which when substituted for the occurrences of x_1, \dots, x_n in e_1 and e_2 produce identical expressions, then we say that e_1 and e_2 are unifiable. (Unification and substitution are considered in detail in section 2.1.2.)

simple example of dependency in the first order predicate calculus we have

$$\forall x \exists y Axy$$

in which the choice of an a_2 for which Aa_1a_2 is true will in general depend on the choice of a_1 . A similar phenomenon occurs in U .

Since variable dependencies are implicit in a polarized expression and since determination of substitutions which make distinct expressions identical is an important operation in both U and E and is closely tied to variable dependencies, it is desirable to have an alternate representation for polarized expressions which explicitly exhibits dependencies among the variables of an expression. Furthermore, in addition to simplifying determination of unifying substitutions the appropriate choice of an explicit variable dependency representation can facilitate the unabbreviation operation, which plays an important role in the extension E .

The occurrence of an abbreviation (a particular kind of subexpression of the E equivalent of a polarized expression) implies certain dependencies among the variables occurring in it and in expressions of which it is a subexpression. Since an abbreviation is considered to be an unanalyzable object⁷ of E until it is explicitly unabbreviated, these dependencies can not be accounted for until un-

⁷In much the same way as an atom is considered unanalyzable.

abbreviation has occurred. It is thus necessary that the particular representation chosen to exhibit variable dependencies allow dependencies which are implicit in an abbreviation, to be linked to abbreviation-external dependencies any time an abbreviation is unabbreviated.

The requirements imposed by unification and unabbreviation upon the form of dependency information representation are somewhat at odds. The representation of variable dependencies for the purposes of unification is best accomplished using Skolem functions, which are unsuitable for the purposes of unabbreviation. On the other hand, dependency forests, which are suitable for the purposes of unabbreviation, are extremely cumbersome for the purposes of unification. It is thus necessary to adopt a redundant representation which involves both.

Skolemized polarized expressions (spe)

The representation adopted for exhibiting the nonquantificational information previously contained in a given polarized expression as well as exhibiting explicit variable dependencies consists of a modified pe called an spe and a forest associated with the spe called a dependency forest. The spe results from removing all occurrences of " $\forall x$ " ($x \in V$) in the original pe and replacing certain variable occurrences with Skolem functions. The associated forest (dpe) explicitly exhibits the variable dependencies present in the original pe .

The combination of spe and dependency forest is redundant to the extent that the Skolem functions occurring in the spe contain the same information as contained in the dpf. However, as mentioned above, the redundant representation is necessary.

Skolem functions for a given spe are obtained directly from the dependency forest which in turn is derived from the structural representation of the original pe. The structural representation of a polarized expression is essentially the tree that would be obtained by parsing the formula to the level of atomic expressions. We shall begin by specifying an algorithm which computes the structural representation of a type 1 expression.

Algorithm A (Structural representation, $T(e)$ for type 1 expression, e)

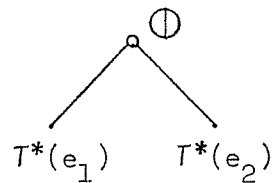
Let e be a type 1 expression. Then $T(e)$ is an ordered tree with contents which is determined as follows:

A1. If e is an atom or is of type 1 and of the form

$\lambda x_1 \dots x_n \bullet e_1 a_1 \dots a_n$ then $T(e)$ is a tree consisting of a single node whose content is e^{γ} .

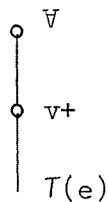
A2. If $e = \bigoplus e_1 e_2$ then $T(\bigoplus e_1 e_2)$ is

⁸The structural representation for e computed by algorithm A will contain nodes having contents of the form a^{γ} where a is a subexpression of e and γ is $+$ or $-$. The suffix γ indicates the parity of a in e , i.e. it indicates whether in e , a occurs in the scope of an even ($+$) or an odd ($-$) number of occurrences of \bigoplus .



where $T^*(e)$ is the complement of $T(e)$ obtained by changing all occurrences of $+$ to $-$ and $-$ to $+$ in $T(e)$.

A3. If $e = \forall v e$ then $T(\forall v e)$ is



The structural representation of a pe is obtained as follows:

1. $T(\oplus e)$ is $T(e)$
2. $T(\otimes e)$ is $T^*(e)$

We may now compute the dependency forest for a type 1 expression. This is accomplished by algorithm B.

Algorithm B (Dependency forest of a type 1 expression)

Let e be a type 1 expression

- B1. Using algorithm A compute the structural representation of e . Let this be $T(e)$.
- B2. Compute the set \bar{N} where \bar{N} is the set consisting of the leaves of $T(e)$, and the nodes of $T(e)$ which have content $v+$ or $v-$ where $v \in V$.

B3. The dependency forest for e is the forest determined by $T(e)$ and \bar{N} (cf. page 19.)

example $\bigoplus \forall x P_x \forall y Q_y$ has the dpf



The complement of the dependency forest for the type 1 expression e is just the dependency forest determined by the tree $T^*(e)$ and \bar{N} . The dependency forest for a pe is defined in an analogous fashion.

Dependency of one variable upon another in the dependency forest F

A variable y is possibly dependent upon a variable x if there is a tree $t \in F$ and nodes n_1, n_2 in t such that $C_t(n_1) = x^+$, $C_t(n_2) = y^-$ and $n_1 <_t n_2$. A variable y is dependent upon a variable x if y is possibly dependent upon x and x occurs in the contents of at least one leaf which has y as an ancestor. A variable y is dependent if either there exists at least one variable x such that y is dependent upon x or y is a negative variable (i.e. there exists node k such that $C_t(k) = y^-$) which has no positive variable ancestors.

Let us once again consider an example from the first order predicate calculus using only the connectives $\neg, \vee, \&$. If a variable x possibly depends upon a variable y then the following situation holds in the formula containing x and y . (We assume

that all variables of quantification are distinct.)

$$\dots \forall x (\dots (\exists y M) \dots) \dots$$

where $\forall x (\dots (\exists y M) \dots)$ and $(\exists y M)$ are both in the scope of an even number of negation symbols.

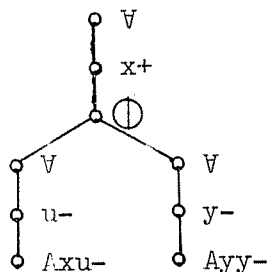
Now suppose M contains no occurrence of x ; then the occurrences of y in M do not depend in any way upon x . (Note that y can occur only in M due to the distinct-variable-of-quantification restriction imposed above.) Thus, treating y as if it did depend upon x would introduce spurious dependencies.

The definition of "dependent" ensures that spurious dependencies of this sort are not introduced; computing efficiency demands that they not be.

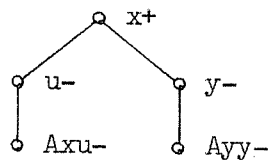
The phenomena illustrated above have their analogs in U_L . Thus \oplus behaves somewhat like negation and $\oplus \forall x \oplus \text{AxAx} \forall x \oplus \text{AxAx}$ like $\exists x \text{Ax}$.

As an illustration of the notions of structural representation, dependency forest, possibly dependent variables and dependent variables, we have the following example:

example Let $e = \forall x \oplus \forall u \text{Axu} \forall y \text{Ayy}$ then $T(e)$ is



and $\text{dpf}(e)$ is then



The variables y and u both possibly depend upon x , however, only the variable u depends upon x .

Skolemization of closed type 1 expressions via their dependency forests

Given the dependency forest for a polarized expression, it is possible to utilize the forest's dependency information to aid in the determination of the corresponding spe. We shall begin by specifying how the Skolem replacement for a negative variable which occurs in a dpf, can be computed. Algorithm C, which computes this replacement, can then be used to Skolemize any type 1 expression.

Algorithm C (Calculating Skolem replacements from a dpf)

Let n be a node (but not a leaf) of the dpf F . Furthermore, let the content of n be v^- . The Skolem replacement for v is calculated as follows:

case 1 v does not depend upon any variable.

C1.1 Let v have type τ and let f have type τ , belong to S and not have been used previously ^{δ_a} . The Skolem replacement for v is f .

^{δ_a} A Skolem function f has been used previously if either it occurs in the dpf or it is part of the Skolem replacement for some negative variable of F other than v .

case 2 v depends upon the positive variables v_1, \dots, v_n

C2.1 Let v have type τ and v_1, \dots, v_n types τ_1, \dots, τ_n respectively. Let f have type $(\tau_1, \dots, \tau_n : \tau)$ where $f \in S$ and has not been previously used. The Skolem replacement for v is $fv_1 \dots v_n$ where the ordering of the v_i s is determined by the linear order induced by the dpf ^{8b}.

Let e be a closed type 1 expression. Skolemization of e is accomplished as follows:

Algorithm D (Skolemization of a type-1 expression)

- D1. Calculate the dependency forest for e using algorithm B.
- D2. Remove all character strings of the form Vv from e (where $v \in V$) which are not within the scope of some " λ ".
- D3. Replace all occurrences of negative variables¹⁰ both in e and $\text{dpf}(e)$ with their Skolem replacements which are calculated by algorithm C.

A polarized expression may be Skolemized in exactly the same way as a closed type 1 expression. As in the case of the closed type 1 expression, the dependency forest for the pe is used to calculate the appropriate Skolem replacements for negative variables

^{8b}Any arbitrary ordering will do, though.

in the modified pe . We note that occurrences of negative variables in the dpe , as well as in the pe , are replaced.

In practice, spe 's are obtained in the manner just described. However, any given spe can also be constructed from elements of TSS using formation rules R1-R3 and POL if, in POL, we relax the restriction that the type 1 expression not contain elements of S . It is thus possible to extend the notions of well formed parts, subexpressions, expressions, suffix and prefix to cover arbitrary spe 's. Unless otherwise indicated, references to these notions should be construed to mean the extended sense of the notion. The language component of U which consists of spe s and their dependency forests will be denoted U_L^* .

In succeeding sections, we define certain computations on dependency forests. These computations make use of the notion of leaf expansion which in turn makes use of the notion of algebraically replacing a dpe leaf with another dpe .

Given two dpe 's F and H and a leaf of H , we can compute a dpe H' using algorithm E. Algorithm E simply replaces n with either F or F^* ⁹ depending upon whether n is positive¹⁰ or negative.

⁹Recall that F^* is obtained from F by replacing occurrences of $+$ with $-$ and $-$ with $+$.

¹⁰The node n is positive (negative) if its content has the form $e+$ ($e-$) where e is an expression.

Algorithm E (Algebraically replacing a dpf leaf with another dpf)

Let F and H be dpf's and let n be a leaf of H . The dpf H' which results from algebraically replacing the leaf n with the dpf F is determined as follows:

$$E1. \text{ Calculate the dpf } G = \begin{cases} F & \text{if } n \text{ is positive}^{10} \\ F * G & \text{if } n \text{ is negative} \end{cases}$$

E2. Calculate H' as follows:

$$N_{H'} = (N_H - \{n\}) \cup N_G$$

$$P_{H'} = (P_H - \{\langle m, n \rangle\}) \cup \{\langle m, m_1 \rangle, \dots, \langle m, m_p \rangle\}$$

where m_i 's are origins of all trees comprising G .

$$\theta_{H'} = (\theta_H - \{\langle m, \langle k_1, \dots, n, \dots, k_s \rangle \rangle\}) \cup \{\langle m, \langle k_1, \dots, m_1, \dots, m_p, \dots, k_s \rangle \rangle\}$$

$$C_{H'} = (C_H - \{\langle n, C(n) \rangle\}) \cup C_G$$

We may now use algorithm E to compute a dpf which results from algebraically replacing a leaf in a specified dpf with the dpf for the content of the leaf.

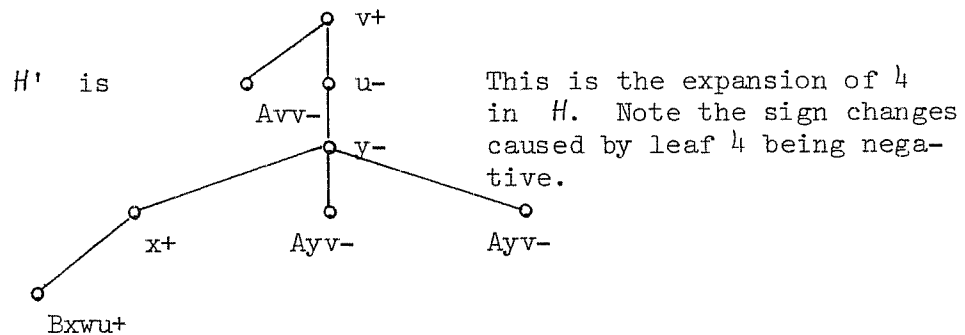
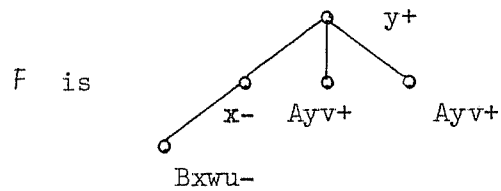
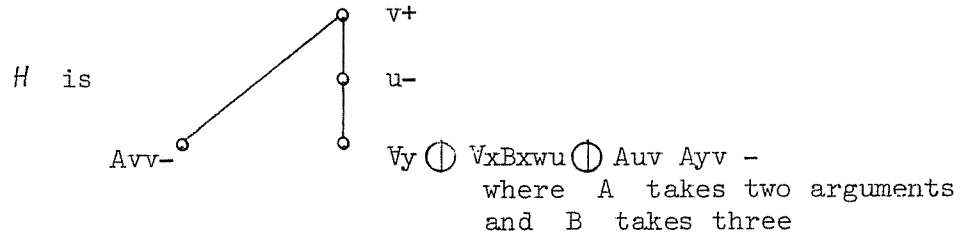
Algorithm F (Leaf expansion)

Let H be a dependency forest which contains the leaf n and let the content of n be $e\gamma$ where γ is either $+$ or $-$.

F1. Calculate the dpf for e using algorithm B. Let F be the result.

F2. Using algorithm E, algebraically replace n with F .

example Application of algorithm E to the leaf 4 of the dpf H.



2.1.2 Substitution, instantiation and unification

unification of spe's The possibility of proving theorems in U ultimately depends upon the possibility of determining under what substitutions for positive variables (if any) two expressions of

... for arbitrary expressions in U , this can be done mechanically in a finite number of steps.

In this section, the concepts of substitution and instantiation (for U) will be presented. We assume that the spe representation of U is used and that subexpression, well formed part, etc. are extended to cover spe's and their parts. The development is essentially that given by Robinson [34] modified where necessary for special characteristics of U .

substitution components (sc) An sc is a pair $\langle e, v \rangle$ where v is a variable of arbitrary type and e is an expression¹ distinct from v but of the same type. If c is an sc, then c_1 will denote the first member (expression) and c_2 will denote the second member (variable).

substitutions A substitution α is a finite set of substitution components with the property that, if $a, d \in \alpha$ and $c \neq d$, then $c_2 \neq a_1$. If E denotes the set of expressions which can occur in subfixes of spe's and $a \in \alpha$ implies $c_1 \in E$ we say α is a substitution over E . We shall use ϕ to denote the empty substitution (i.e. the substitution containing no substitution components).

instantiation The instantiation of the expression e by the substitution α over E (denoted $e\alpha$) is defined as:

¹Note that subexpressions of an spe can not be R4 expres-

1. If e is an R1 expression, then $e\alpha = e$ if $e \in S$
 $= e$ if $e \in V$ and for no $c \in \alpha$
 is it the case that $c_2 = e$
 $= a$ otherwise (where $\langle a, e \rangle \in \alpha$)
2. If e is an R2 expression and of the form $ae_1 \dots e_n$
 then $e\alpha = a\alpha e_1 \alpha \dots e_n \alpha$. That is, $e\alpha$ is obtained by
 forming an R2 expression from the expressions $a\alpha^2$,
 $e_1 \alpha, \dots, e_n \alpha$
3. If e is an R3 expression and of the form $\lambda x_1 \dots x_n \bullet e_1$
 then $e\alpha = \lambda x_1 \dots x_n \bullet e_1 \alpha^3$

It follows by induction on the number of subexpressions of an expression that the type of an expression is invariant with respect to instantiation. In fact, this is necessary if the results of 2 and 3 (above) are to be well defined.⁴

By instantiation of a prefixed formula e we mean the prefixed formula obtained by instantiation of the suffix of e .

unifiable We shall say that two expressions $e_1, e_2 \in E$ are unifiable if there exists a substitution σ over E such that $e_1 \sigma = e_2 \sigma$. The substitution σ is called a unifying substitution.

²If a is the character \bigoplus then $a\alpha$ is \bigoplus .

³It is necessary, in this case, that α not contain any component whose first element is x_i ($i = 1, \dots, n$).

⁴For example if e was type 1 but $e_1 \alpha$ was not, then $\bigoplus e_1 \alpha e_2 \alpha$ would not be well formed. (In particular, it would not be an R2 expression.) Thus, rule 2 above would not define $\bigoplus e_k e_2 \alpha$.

composition of substitutions The composition of two substitutions ρ and α is defined to be the substitution $\rho\alpha$ given below.

$$\rho\alpha = \{ \langle e, v \rangle : e\alpha \neq v \text{ and } \langle e, v \rangle \in \rho \} \cup \{ c : (c \in \alpha) \ \& \ \neg \exists d ((d \in \rho) \ \& \ (c_2 = d_2)) \}^5$$

Robinson established the following properties of substitutions in his formulation of first order logic. They may similarly be established for expressions of \mathcal{E} . (Let $\rho, \alpha, \mu, \sigma$, be arbitrary except for constraints explicitly indicated.)

1. $\phi\rho = \rho\phi = \rho$
2. $(\rho\alpha)\mu = \rho(\alpha\mu)$ (associativity)
3. If e is any expression and $\sigma = \rho\alpha$ then $e\sigma = e\rho\alpha$
4. If for all expressions e , $e\alpha = e\rho$ then $\alpha = \rho$
5. If e is any expression then $(e\sigma)\alpha = e(\sigma\alpha)$

substitution in trees with contents If t is a tree with contents over the set of spe's and σ is a substitution over \mathcal{E} , then $t\sigma$ is the tree obtained by replacing $C(n)$ with $C(n)\sigma$ for each $n \in \mathbb{N}_t$.

2.1.3 Language (semantics)

In this section, we present the model theoretic foundations for the formal system \mathcal{U} . Although the system \mathcal{U} is syntactical in nature, its inference operators are closely linked to the semantic

⁵Note that if $\langle t_1, x \rangle \in \rho$ and $\langle t_2, x \rangle \in \alpha$ then $\rho\alpha$ contains the component $\langle t_1, x \rangle$ but not the component $\langle t_2, x \rangle$.

notion of truth-value analysis. In fact, given the intended method of interpreting the formal objects of U , the inference operators presented in the next section may be viewed as truth function analyzers which determine what truth value assignment to the subexpressions of the given expression could account for the value of the expression.

In introducing the following concepts, we are necessarily vague. The discussion is meant to motivate rather than define the concepts, which will presently be precisely defined.

U is a refutation system in which the syntactical notion of proving a sentence amounts to demonstrating that the negation of the sentence can never be true. In practice, the closed type 1 expression e , which is given in quantificational form, is presented in the polarized form $\textcircled{F}e$. This polarized expression is translated into its associated spe . If processing the spe with the inference operators produces a distinguished object called the empty tree, then the spe can never be true. This implies that $\textcircled{F}e$ can never be true. It then follows that e must always be true.

In this section, we are mainly concerned with the concept of true expression. Intuitively, given an expression of type 1 and a set of objects about which meaningful statements may be made, we may think of the expression as being interpretable as a statement about the objects of the set. If the statement can be verified for elements of the given set, then the formal expression giving rise to the statement may be thought of as being true for the particular set relative to the interpretation which relates it to the statement and the objects.

Rather than interpret expressions of U_L in exactly this manner, we specify an evaluation procedure which maps expressions into a set of objects derived from the given set. The derived set contains possible values for each expression type. In particular, type 1 expressions map into either of the elements T or F which both belong to the derived set. We say that an expression is true under an evaluation if its image under the evaluation is T . We now present the evaluation procedure and related notation.

derived domain D^* Given the non-empty set of individuals D whose elements will be arbitrarily designated as type 0, we can define a related set D^* which contains elements of each type allowable in U_L . D^* will be the range of the valuation mapping which provides values for arbitrary U_L -expressions by mapping a given expression into an element of D^* . D^* is defined recursively as follows:

1. $D \cup \{T, F\} \subset D^*$ where T and F are distinct elements not occurring in any D .
2. Let S be any set of the form $\{ \langle x_1, \dots, x_n \rangle : x_i \in D \text{ and has type } \tau_i \}$ and let y be a function whose domain is S and whose range is included in D^* and contains only entities of type τ . If $Q = \{ \langle x, y(x) \rangle : x \in S \}$ then Q is an element of D^* and has type $(\tau_1, \dots, \tau_n : \tau)$.¹

¹ D^* may be generated as follows:

1. Apply rule 1 to obtain the initial current D^* (basis)
2. Using the current constituency of D^* form a set S which satisfies the stated requirements.
3. Choose any non-empty subset A of the current D^* whose elements are all of the same type. Let this type be τ .

value assignment Let M be a subset of $V \cup S$. A value assignment for the elements of M relative to the domain D is a mapping of M into D^* with the property that an element of M and its image in D^* have the same type. A U_L -expression containing free variables may be evaluated according to rules which follow if the free variables are fixed for purposes of evaluation by specifying a value assignment for them. In this case the expression being evaluated is said to be evaluated for (or at) the specified value assignment. Evaluation of a set of U_L -expressions at a value assignment for their collective free variables gives rise to a valuation mapping².

valuation of U_L -expressions If E is a set of U_L -expressions, D an arbitrary non-empty set and p a value assignment for the free variables occurring in elements of E , then a valuation for E relative to D at the value assignment p is a mapping from E into D^* which satisfies the following recursive definition:

For $e \in E$ and e

E_2 - an R_2 expression, then e has that value which the value of the operator of e coordinates with the n -tuple of values

¹(continued)

4. Choose any function y which maps S onto A .
5. Let the new D^* be $D^* \cup \{ \langle x, y(x) \rangle : x \in S \}$.
- *6. Go back to step 2.

*Note that we assume there is some exhaustive method of enumerating all triples $\langle S, A, y \rangle$.

²This simply means that if R is a set of U_L -expressions containing the free variables v_1, \dots, v_n then the valuation mapping is a mapping of R into D^* which preserves type. In order that

determined by the argument expressions³.

- E1 - $\textcircled{1}$ is assigned the value $\{\langle\langle F, F \rangle, T \rangle, \langle\langle F, T \rangle, T \rangle, \langle\langle T, F \rangle, T \rangle, \langle\langle T, T \rangle, F \rangle\}$
- E3 - an R3 expression of the form $\lambda x_1 \dots x_n \bullet e_1$, then e has the value $Q = \{\langle x, e_{1_x} \rangle : x \in S\}$. (S was given in step 2 under the definition of D^* and e_{1_x} is the value of e_1 at the values of x_1, \dots, x_n .⁴) We note in particular that Q is obtained by varying over all possible value assignments for x_1, \dots, x_n .
- E4 - an R4 expression of the form $\forall v e_1$, then e has the value T if the value of e_1 at each value of v is T . Otherwise, e has the value F .
- E5 - a polarized expression with prefix $\textcircled{T}(\textcircled{F})$ and suffix e_1 , then e has the value $T(F)$ if e_1 has the value T . Otherwise, e has the value F .

²(continued) it be possible to define such a mapping, it is necessary to first fix the values (D^* images) of the variables v_1, \dots, v_n . This is accomplished by specifying a value assignment for the variables, i.e., a type preserving mapping from $\{v_1, \dots, v_n\}$ into D^* .

³We assume that D^* has been generated. This means that D^* contains elements of every allowable type, (regardless of the choice of D). Since valuation of E relative to D is recursively defined, we assume that the operator of the R2 expression as well as its arguments, has already been evaluated using E1-E5. We note that any variable may be assigned any value in D^* of the correct type. This and the fact that rules E2-E5 require evaluation of smaller expressions than the given one insures that the recursion terminates.

⁴Note that x_1, \dots, x_n can receive values either by a value assignment or if $\lambda x_1 \dots x_n \bullet e$ is applied to the arguments a_1, \dots, a_n in which case their values are the values of a_1, \dots, a_n .

Valuation as just defined has the property that given an arbitrary set E of U_L -expressions, a domain D and value assignments for the free variables in members of E , the rules E1-E5 produce a correspondence between E and D^* which assigns to each element of E exactly one element of D^* (i.e. valuation is a function).

valuation of U_L -expressions Expressions of U_L^* involving Skolem functions may also be evaluated using rules E1-E5 if E4 is replaced by E4' below.

E4' - Let e contain the variables v_1, \dots, v_n and the special constants f_1, \dots, f_m then e has value T if and only if there exists a value assignment f for $\{f_1, \dots, f_m\}$ such that for every value assignment v for $\{v_1, \dots, v_n\}$ e is true at f and v .

As in the case of U_L -expressions, the evaluation process just given may be used to define a valuation mapping from arbitrary sets of spe's into D^* . We note, however, that E4' must be applied to the set of spe's as a whole. That is, there must exist a value assignment for all Skolem function symbols occurring in spe's of the set which works for all value assignments to all the variables occurring in the set.

example Evaluation of the spe $\textcircled{F} \textcircled{\bigcirc} \text{hfxxhxx}$ relative to the domain $D = \{a, b\}$

<u>Type</u>	<u>object</u>
0	$x \in V$
(0:0)	$f \in S$
(0,0:1)	$h \in S$

Part of D^* follows

type	objects			
0	a,b			
1	T,F			
(0:0)	f_1	f_2	f_3	f_4
	$\langle a,a \rangle$ $\langle b,b \rangle$	$\langle a,a \rangle$ $\langle b,a \rangle$	$\langle a,b \rangle$ $\langle b,b \rangle$	$\langle a,b \rangle$ $\langle b,a \rangle$
(0,0:1)	h_1	h_2	...	h_{16}
	$\langle \langle a,a \rangle, T \rangle$ $\langle \langle a,b \rangle, F \rangle$ $\langle \langle b,a \rangle, F \rangle$ $\langle \langle b,b \rangle, T \rangle$	$\langle \langle a,a \rangle, F \rangle$ $\langle \langle a,b \rangle, T \rangle$ $\langle \langle b,a \rangle, T \rangle$ $\langle \langle b,b \rangle, F \rangle$		
⋮				...

We apply rule E_4' and find that the value assignment to $\{h,f\}$ required to make $\textcircled{F} \textcircled{\ominus} hfxxhxx$ true is $\{h_1, f_1\}$ i.e., $\textcircled{F} \textcircled{\ominus} hfxxhxx$ is true for $x = a$ and $x = b$ at this value assignment. That is:

$$\begin{aligned}
 f_1 a &= a & h_1 aa &= T & \textcircled{\ominus} h_1 f_1 aah_1 aa &= F & \textcircled{F} \textcircled{\ominus} h_1 f_1 aah_1 aa &= T \\
 f_1 b &= b & h_1 bb &= T & \textcircled{\ominus} h_1 f_1 bbh_1 bb &= F & \textcircled{F} \textcircled{\ominus} h_1 f_1 bbh_1 bb &= T
 \end{aligned}$$

The definition given for valuation makes precise the intuitive notion truth-of-an-expression. Other intuitive notions previously

discussed such as sometimes-true-expression (satisfiable), always-true-expression (valid) and never-true-expression (unsatisfiable) are defined as follows:

satisfiability A polarized expression e (or an spe) is said to be satisfiable if there exists a non-empty domain D and a valuation relative to D under which e has the value T .

validity A polarized expression e (or spe) is said to be valid if for every non-empty domain D and every valuation relative to D the value of e is T .

unsatisfiability A polarized expression e (or spe) is unsatisfiable if it is not satisfiable.

remark 1 The polarized expression $\textcircled{P} e$ is unsatisfiable if and only if e is valid.

remark 2 If e is a polarized expression and \bar{e} is the spe obtained from e , then e is satisfiable if and only if \bar{e} is satisfiable⁵.

The properties set forth in remarks 1 and 2 lie at the core of the operation of U . As mentioned earlier, the quantified expression e to be proved is presented in the pe form $\textcircled{P} e$ and is translated into its spe associate. The inferential operation of U

⁵There is a second kind of Skolemization, the dual of that used in this paper, which preserves validity [5].

attempts to show the unsatisfiability of this associate. By remark 2, this means that $\textcircled{F}e$ is unsatisfiable and thence by remark 1 that e is valid. These aspects of U will be considered in greater detail in the next section.

2.2 Inference operators of U

Intuitively, the inference operators of U provide a means of analyzing the truth-functional structure of a given spe. Commencing with the spe representation of the negation of the expression to be proved, analysis of the spe proceeds under the assumption that the spe is satisfiable. The nature of valuation is such that only certain valuations for the atomic subexpressions of this initial spe could account for the assumed value of T .

At any given stage of analysis, the truth-functional structure of the initial spe thus far uncovered is represented in tree form. Such a representation explicitly exhibits which valuations for those subexpressions of the spe reached at that stage can account for the assumed satisfiability of the initial spe. The goal of analysis is to produce a truth-functional tree which is manifestly unsatisfiable, i.e. one which explicitly demonstrates the impossibility of any valuation for the subexpressions of the initial spe producing the assumed satisfiability.

The truth-functional structure of an initial spe is represented at any given stage of analysis by an ungrounded tree.

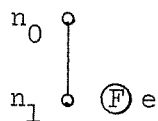
ungrounded tree An ordered tree whose nodes (excluding the origin) have contents in E^1 is called an ungrounded tree. An ungrounded tree is just an ordered tree with contents in E whose content function is undefined at the origin.

An initial tree, which is related to the initial spe, is defined in the following manner:

¹ E is the set of spe's.

initial tree If t is an ungrounded tree of the form $N_t = \{n_0, n_1\}$, $P_t = \{\langle n_0, \langle n_1 \rangle \rangle\}$ $C_t = \{\langle n_1, \textcircled{F}e \rangle\}$ $\textcircled{F}e \in \Xi$ then t is said to be an initial tree². If $\textcircled{F}\bar{e}$ is the polarized expression for which $\textcircled{F}e$ is the associated spe, then \bar{e} is called the initial expression for t and $\textcircled{F}e$ is called the initial spe for t .

Diagrammatically, if $\textcircled{F}e$ is an initial spe, then the initial tree for it is simply



The semantic concept of satisfiability can be extended to ungrounded trees. We say that a tree is satisfiable if there exists a valuation (relative to some domain) for the set of all spe's contained in the tree having the property that some branch is true under the valuation. A branch is true under the valuation v if all its contents have value T under v . It is satisfiable if there exists a valuation v under which it is true.

A branch is said to be closed if it contains a pair of conjugate³ spe's. A tree (ungrounded) is said to be closed if each of its branches is closed. It follows immediately that a closed branch cannot be true under any valuation and a closed tree is unsatisfiable.

²We shall denote initial trees by Θ . This should not be confused with the ordering function for the tree t , i.e. θ_t . The correct meaning will always be clear from the context.

³Two spe's are said to be conjugates if their suffixes are identical and one has prefix \textcircled{T} while the other has prefix \textcircled{F} .

The inference operator O_4 of U removes closed branches from ungrounded trees. If this operator is applied to a closed tree, the resulting object is arbitrarily designated as the empty tree and is denoted as ϕ . Since a closed tree can never be satisfiable, we agree that ϕ is unsatisfiable.

basic inference operators The inference operators of U formalize the process of determining, for the set of atomic^{3a} subexpressions of the initial spe, valuations which are implied by the assumed value of the spe. The possibility of such operators is the direct consequence of the semantic properties developed in the last section.

The starting point for inference operation in U is the initial tree Θ . With this tree we associate the dpf for the initial spe. The U -inference operators, presented in the following pages, operate upon ungrounded trees, which represent some stages of truth-function analysis of the initial spe, and produce new trees which represent a deeper level of analysis. Each inference operator, in addition to producing a truth-function analysis tree, produces a dpf which is associated with the tree. This dpf is computed from the dpf(s) associated with the operand tree(s) and reflects the variable dependencies associated with the content expressions of the associated tree(s)⁴. Part of the definition of each inference operator specifies how this dpf is calculated.

^{3a}See page 27.

⁴The dpf's associated with ungrounded trees are required by both the λ -reduction operator (O_1) and the unabbreviation operator (O_{11}).

Let us call any type 1 subexpression whose first character is λ and which does not occur within the scope of any λ , a maximal λ -expression. Any type 1 expression may be considered to be constructed from maximal λ -expressions and atoms which are not contained in any maximal λ -expression using rules R2 and R3. We shall refer to the set of such subexpressions of an expression as the pseudo atoms of the expression.

Given an initial tree θ and its associated dependency forest, the method of determining the dpf from the structural representation of the initial spe provides a natural correspondence between the pseudo atoms of the initial spe and the leaves of $\text{dpf}(\theta)$.

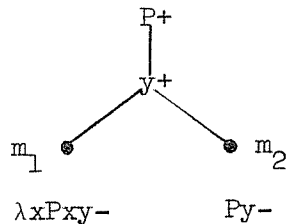
As the inference operators of U are applied to θ and its descendants, the initial spe is decomposed into its constituent subexpressions until, at some stage, the pseudo atoms of the initial spe begin to appear as suffixes of the content of some nodes. For any tree, thus obtained, we may define a correspondence between these nodes and the leaves of the dpf associated with the initial tree. This correspondence (let us call it ζ) will have the following properties:

1. It is a function on a subset of the nodes of the given tree into the set of leaves of the dpf associated with the tree.
2. It is 1-1.
3. Nodes for which the mapping is defined contain spe's whose suffixes are either atoms or whose first character is λ .

example 1 ζ not defined for any node of the given tree

Let the given tree be the initial tree θ where θ is

n_0 \circ
 n_1 \circ $\oplus \ominus \lambda x P x y P y$ and $\text{dpf}(\theta)$ is



The pseudo atoms $\lambda x P x y$ and $P y$ at the leaves m_1 and m_2 are proper subexpressions of the initial spe. Thus, $\zeta(n_1)$ is not defined.

example 2 ζ defined on the initial tree

Suppose θ is n_1 \circ $\oplus \ominus \lambda x \ominus P x P y y$ where $\text{dpf}(\theta)$ is y^+
 m_1
 $\lambda x \ominus P x P y y^+$

then the only pseudo atom of the initial spe is $\lambda x \ominus P x P y y$ which corresponds to leaf m_1 . We thus take $\zeta(n_1)$ to be m_1 .

The operators of U (and of E) provide a means of associating new dpf 's with the new trees which they produce. It is also possible to calculate a correspondence between a subset of the nodes of the new tree and the leaves of its associated dpf . (This is described in detail in appendix D.) This correspondence is motivated by considerations discussed above. It is the same as ζ (in fact we shall denote it by ζ) except that it fails to be 1-1 in general. The

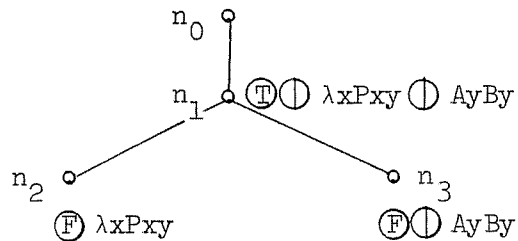
reason for this will become apparent in section 3. Given the function ζ , we may calculate the set $\zeta^{-1}(m)$ as follows:

$$\zeta^{-1}(m) = \{n : n \in N_t \text{ and } \zeta(n) = m\}$$

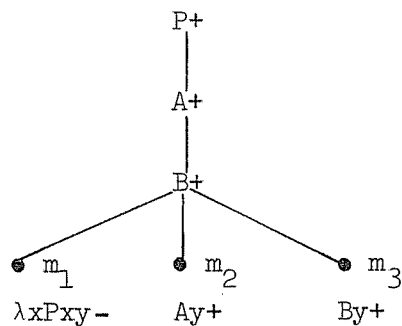
where m is a leaf of $\text{dpf}(t)$. i.e. $\zeta^{-1}(m)$ is the set of nodes which map into m under ζ .

example

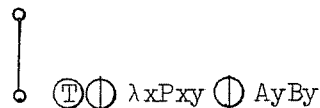
The tree t



and its associated dpf

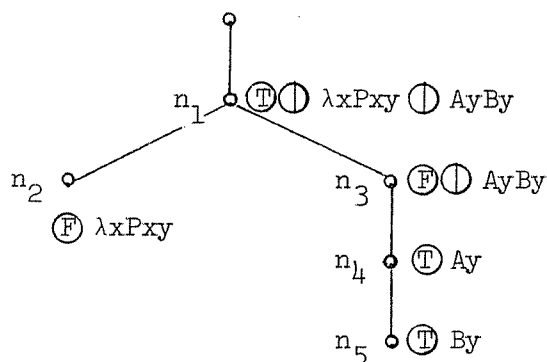


may be derived from the initial tree



ζ is defined on the node subset $\{n_2\}$; $\zeta(n_2) = m_1$ (in this case $\zeta^{-1}(m_1) = n_2$).

We note that ζ does not map a node subset of N_t onto the leaves of $\text{dpf}(t)$. However, if we apply the O_2 inference operator (to be defined presently) to t we obtain the tree



with the same dpf as given above. Since $\zeta(n_2) = m_1$, $\zeta(n_4) = m_2$, $\zeta(n_5) = m_3$, ζ maps $\{n_2, n_4, n_5\}$ onto the leaves of its associated dpf .

From a purely logical point of view, certain operators introduced below are redundant in the sense that the class of U -theorems would not be reduced by their removal. However, from an overall point of view, which takes into account the purpose of the logic U , their inclusion (and later that of other inference operators) is justified on the grounds that they allow more efficient (and thus in a sense more powerful) strategies to be written.

The inferential portion of E consists of twelve inference operators (O_1, \dots, O_{12}) . Operators O_1, \dots, O_{10} belong to U .

Functionally O_1, O_2, O_3 and O_4 determine the class of expressions which are provable in U . Operators O_5 , and O_6 are editing operations; and O_7 and O_8 provide a lemma utilization capability. Finally, O_9 and O_{10} provide for synthesis of results stemming from applications of O_7 .

Algorithmic Expansion

Let a_1 be a node of the ungrounded tree τ and suppose that a_1 contains as subexpression a specified occurrence of an expression a of the form $\lambda x_1 \dots x_n \cdot d \ e_1 \dots e_n$. Furthermore, assume that the content of a_1 has the form δe where δ is either \textcircled{T} or \textcircled{F} and let $m = \zeta(a_1)$.

case 1 a is e

1.1 dpf calculation In this case it can be shown that each $x \in \zeta^{-1}(m)$ as well as m has content $\lambda x_1 \dots x_n \cdot d \ e_1 \dots e_n$ where d has type 1.⁵

1.1.1 Let $d_1 = d\sigma$ where $\sigma = \{\dots, \langle e_i, x_i \rangle, \dots\}$ ⁶

1.1.2 Let H be the dpf which results from replacing the content of m with d_1 and the same suffix as originally contained in m .

1.1.3 Let H_1 be the result of expanding the leaf m in H as prescribed in algorithm F of section 2.1.1.

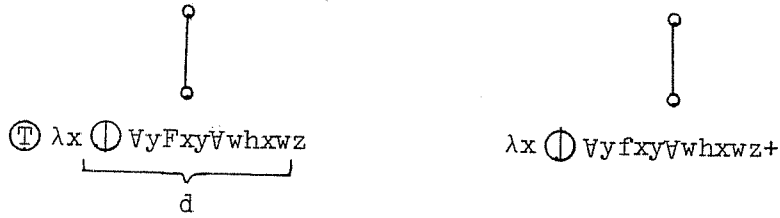
1.1.4 Let d_2 and H_2 be the result of Skolemizing d_1 and H_1 from the dpf H_1 as prescribed in algorithm D of section 2.1.1.

1.2 tree calculation For each $x \in \zeta^{-1}(m)$, replace the suffix of $C_t(x)$ with d_2 as calculated in 1.1.

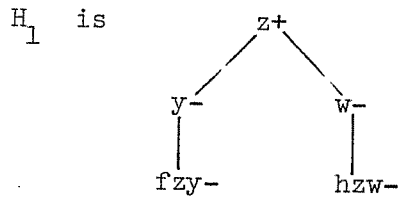
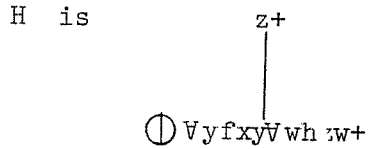
⁵This follows inductively from the definition of dpf. We note that for $x \in \zeta^{-1}(m)$ the content also includes a prefix which is either \textcircled{T} or \textcircled{F} whereas m contains a suffix which is either + or - the suffix (+ if $C_t(x) = \textcircled{T} \lambda x_1 \dots x_n \cdot d \ e_1 \dots e_n$ and - otherwise).

⁶The formation rules for expressions of U_{T^*} insure that "e_i is free for x_i in d." (See Kleene [18], page 79, for the meaning

example original tree and associated dpf are

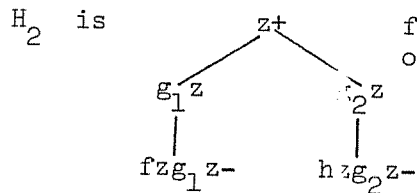


$$d_1 = \oplus VyfzyVwhzw \quad (\sigma = \{<z, x>\})$$

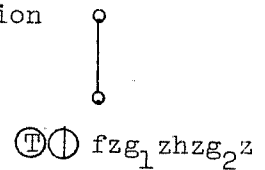


$$d_2 = \oplus fzg_1zhzg_2z$$

where g_1z and g_2z are the Skolem replacements for y and w respectively.



finally, application of 1.2 produces →



⁶(continued) of the quoted expression.)

case 2 a is a proper subexpression of e . (We assume a has the form $\lambda x_1 \dots x_n \bullet d e_1 \dots e_n$ and that a is not type 1.)

2.1.1 dpf calculation Replace the subexpression a occurring in the content of m with $d\sigma$ where $\sigma = \{\dots, \langle e_i, x_i \rangle, \dots\}$

2.1.2 tree calculation Replace the subexpression a occurring in content of $C_t(x)$ with $d\sigma$. ($x \in \zeta^{-1}(m)$.)

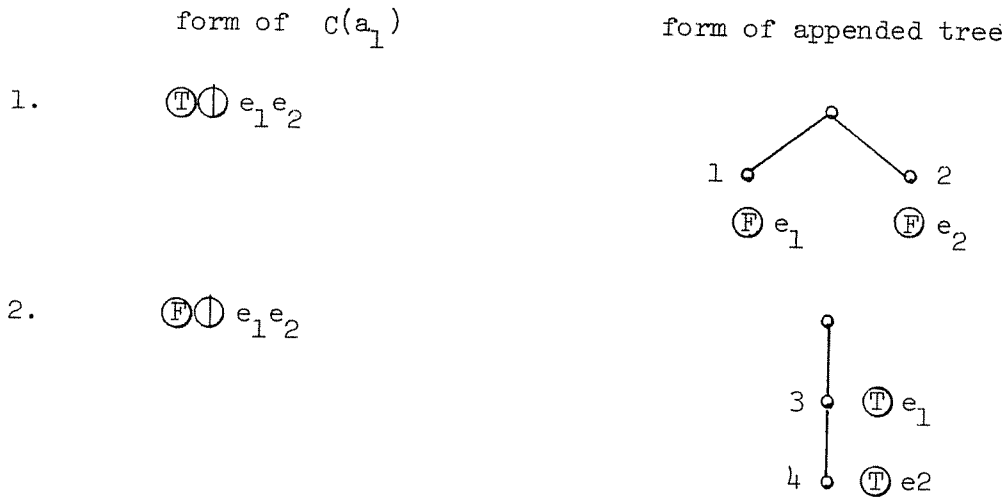
(We note that cases 1 and 2 fail to exhaust all possibilities. In particular, the case where a is type 1 and a proper subexpression. Such a case, however, may be reduced to case 1 by applying the other inference operators of U .)

The complexity of case 1 is necessitated by the fact that O_1 must calculate the dpf for the type 1 expression d which was previously inaccessible due to its occurrence within the scope of λ . Had λ been eliminated from the initial expression as a preprocessing step, then the dpf for the initial spe and that calculated here would be the same⁷.

O_2 - the operators $T \oplus$ and $F \oplus$

Let a_1 be a node and a_2 a leaf of the ungrounded tree t ($a_1 \prec_t a_2$). The tree $O_2(a_1, a_2)$ is the tree which results from appending the appropriate tree (given below) to the leaf a_2 .

⁷This assumes that no other R3 expressions occurring in the initial expression have been processed at this stage.



The dpf associated with $O_2(a_1, a_2)$ is the same as that associated with the tree t . (Two nodes are called disjoint brothers if they arise from the same application of O_2 and occur on different branches in the resulting tree.)

example

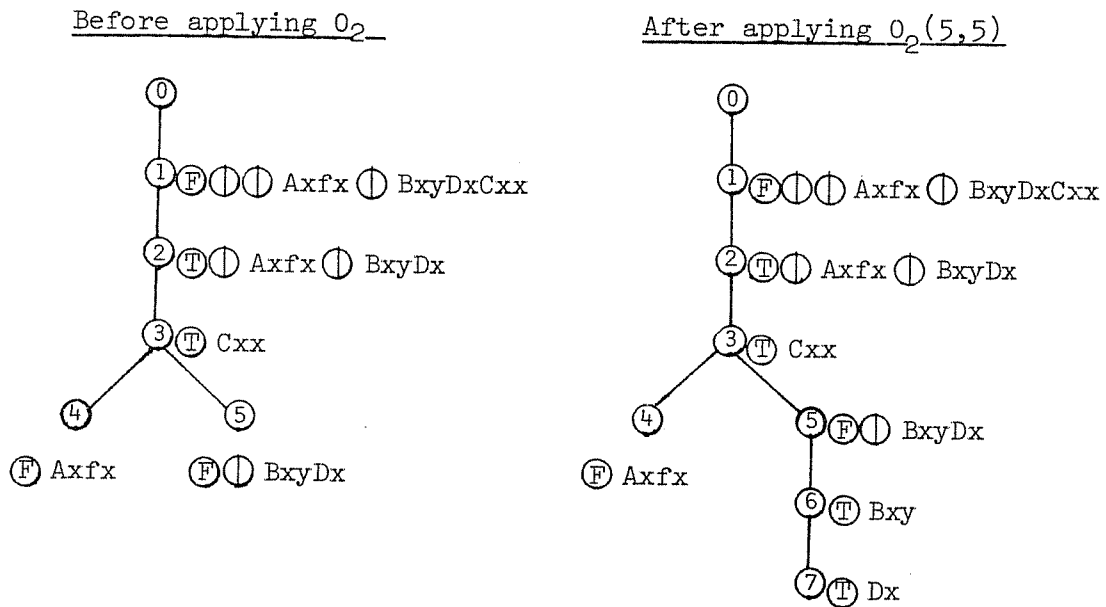


Figure 1.

The symbol " \oplus " denotes Sheffer stroke. In Russell-Whitehead [47] notation, infix Sheffer stroke is definable in terms of \neg and \vee , i.e. $A \oplus B$ is truth-functionally equivalent to $\neg A \vee \neg B$. Thus, $A \oplus B$ is true if and only if A is false or B is false. This corresponds to the rule for $\textcircled{T} \oplus e_1 e_2$ given above. $A \oplus B$ is false if and only if both A and B are true. This corresponds to the rule for $\textcircled{F} \oplus e_1 e_2$ given above.

The U -operators tree substitution and multiple closed branch removal operate on ungrounded trees and produce other ungrounded trees as follows:

O_3 - the operator tree substitution

Let t be an ungrounded tree and σ a substitution over E (the set of spe's). $O_3(t, \sigma)$ is the ungrounded tree which results from applying σ to the content of each node of t .

The dpf associated with the tree $O_3(t, \sigma)$ is obtained by applying σ to the content of each node of the dpf for t .

The operator O_3 may be applied to the tree illustrated in Figure 2. If σ is the substitution $\{ \langle a, x \rangle, \langle b, y \rangle \}$ we obtain the example of Figure 3:

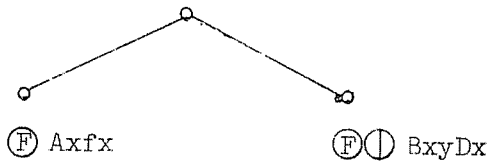


Figure 2

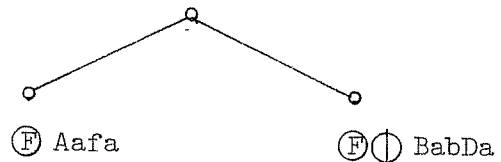
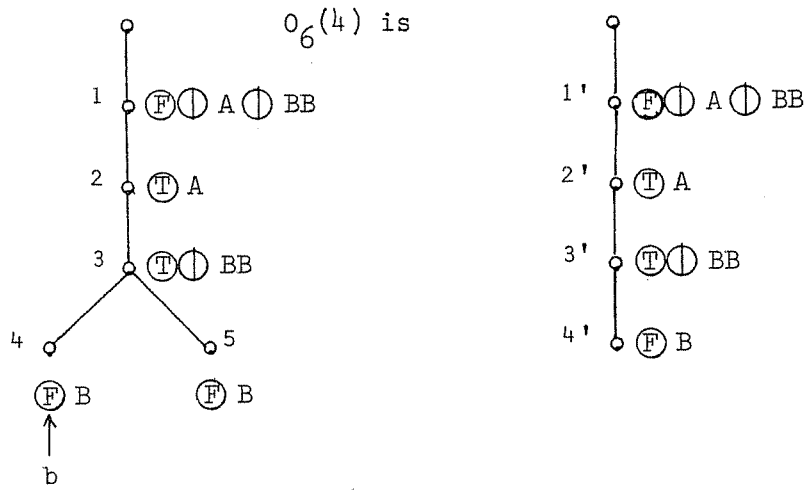


Figure 3

example



Proving initial expression using the operators $\{O_1, O_2, O_3, O_4, O_5, O_6\}$

The inference operators thus far presented provide the necessary mechanism for analyzing the truth-functional structure of any initial expression. Commencing with an initial tree Θ , which represents the negation of the initial expression e , the inference operators of U may be applied to Θ and Θ 's descendants in an attempt to demonstrate the unsatisfiability of Θ and thus the validity of e . Since each of the inference operators preserves truth, it suffices to demonstrate that the above procedure produces a tree which is manifestly unsatisfiable. We note, in particular, that any ungrounded tree which contains only closed branches is unsatisfiable.

The preceding semantic considerations motivate the syntactical notion of proving an initial expression given below. This notion (relative to the current set of inference operators) depends on the notion of linear extension.

linear extension

We write $s \xrightarrow{L} t$ and say that t is a linear extension of s if there exist ungrounded trees t_0, \dots, t_N such that $t_0 = s$, $t_N = t$ and t_i is obtained by applying O_k to t_{i-1} where $1 \leq i \leq N$ and $O_k \in L = \{O_1, \dots, O_6\}$. Any ungrounded tree is considered to be a linear extension of itself.

proving an initial expression e

Suppose e is an initial expression and Θ is the initial tree which represents the negation of e . If $\Theta \xrightarrow{L} \phi$ (where ϕ is the

empty tree) then we say that e is provable relative to the operator set L .

We note that since $\Theta \neq \phi$, asserting $\Theta \xrightarrow{L} \phi$ implies that there exists a tree t containing only closed branches such that $\phi = O_4(t)$ ⁸. Since t is manifestly unsatisfiable we see (by a simple inductive argument) that Θ is unsatisfiable. Thus demonstrating $\Theta \xrightarrow{L} \phi$ ensures that the initial expression is valid.

The operators O_5 and O_6 belong to a class of operators whose inclusion in U is motivated by extra-logical considerations. Both operators perform an editing function thus reducing storage requirements for ungrounded trees. In addition, under certain conditions they allow strategies which utilize U to be simpler than would be possible if they were not included in U . The operators O_7, O_8, O_9 , and O_{10} (introduced later) also share this latter property. As with O_5 and O_6 the class of provable initial expressions would not be reduced by their exclusion from U .

operators which deal with subproblems

In attempting to demonstrate that a given initial tree Θ is unsatisfiable, we produce a sequence of ungrounded trees Θ, t_1, t_2, \dots . We can demonstrate that Θ is unsatisfiable if we can demonstrate that t_1 is unsatisfiable; that t_1 is unsatisfiable if t_2 is unsatisfiable and so on. Our goal, of course, is to

⁸The parameters of O_4 are branches. However, in certain cases where the particular branches are irrelevant, we shall schematically indicate an O_4 application in this manner.

produce the tree ϕ . In pursuing this goal, we must deal with the problem of demonstrating that an arbitrary linear extension of θ is unsatisfiable. The inference operators O_7 and O_8 introduced in this section allow this problem to be split into subproblems.

Loosely speaking, the operator O_7 allows us to generate ungrounded trees which are derived from certain subforest (called proper subforests) of any ungrounded tree. If it can be demonstrated that the generated tree is unsatisfiable, then O_8 allows the proper subforest from which the tree was derived to be deleted from its parent tree. The parent tree is then unsatisfiable if the tree produced by applying O_8 to it is unsatisfiable.

The operators O_7 and O_8 require the introduction of several new notions. These notions are now presented.

linear descendant

It is useful to have a notion of descendancy which holds between nodes of trees related by linear extension. We say that a node m occurring in the ungrounded tree t is a linear descendant of the node n occurring in the ungrounded tree s if

1. $m = n$ (and thus $t = s$)

or 2. there exist ungrounded trees t_1 and t_2 , nodes n_1 and n_2 belonging to t_1 and t_2 respectively such that:

$$a) \quad s \xrightarrow{L} t_1 \xrightarrow{L} t_2$$

b) t results from applying O_2 to t_2

c) n_1 is a linear descendant of n

- d) n_2 is the image of n_1 in t_2 (see appendix C)
- e) m is one of the nodes (but not the origin) in the appended tree associated with $O_2(n_2, k)$. (Here k is some leaf in t_2 .)

If the node m is a linear descendant of the node n , then the content of m (excluding the prefix) is essentially a subexpression of the content of n . The rather involved definition given above is necessary in order that we properly account for the heredity of the subexpression which must be traced through the sequence of ungrounded trees commencing with s and terminating with t .

The notion of linear descendancy is necessary in order to talk about the logical heredity of an expression which occurs as the content of a given node. The successor relationship of the ungrounded tree is inadequate for this purpose since two successors are not necessarily related by logical descendancy.

proper subforest

The forests, from which O_7 generates ungrounded trees, must obey certain constraints if the consistency of U is not to be violated. In particular, we require the following:

Let t be an ungrounded tree and $R \subseteq N_t - \{\text{org}(t)\}$. Further, let F be the subforest determined by R and suppose that x is a positive variable occurring in F , $n \in N_t - N_F$ and $C_t(n)$ contains x . If it is not the case that there exist nodes $m, m_1, n_1 \in N_t$ such that $m \in N_F$, $C_t(m)$ contains x , m is a linear descendant

of m_1 , n is a linear descendant of n_1 and m_1 , n_1 are disjoint brothers⁹, then F is a proper subforest of t .

To understand why these constraints are required, let us consider the following formulas of the first order predicate calculus.

1. $\forall x (Ax \vee Bx)$
2. $\forall x Ax \vee \forall x Bx$

The basic difference between 1 and 2 is that given arbitrary terms t_1 and t_2 , 2 allows us to deduce $At_1 \vee Bt_2$ whereas 1 does not. Alternatively, $\forall x$ does not distribute over \vee .

Now in the case of U , a similar situation holds with regard as to what deductions can be made from the spe $\textcircled{T} \textcircled{\perp} Ax Bx$. Suppose for example that $\textcircled{F} Bx$ occurs as content of one of the nodes of F . If $\textcircled{F} Bx$ has resulted from an expression such as $\textcircled{T} \textcircled{\perp} Ax Bx$, then it is necessary that the node containing $\textcircled{F} Ax$ also occur in F . Otherwise, the operators of U allow independent instantiation of $\textcircled{F} Ax$ and $\textcircled{F} Bx$ implying that the scope of quantification relative to x was not the entire expression $\textcircled{\perp} Ax Bx$. The constraints, which a proper subforest must satisfy, preclude such an omission while allowing omission from F of nodes containing x but whose content could have been revised in the initial expression.¹⁰

⁹It will be recalled that nodes are disjoint brothers if they arise from the same application of O_2 and occur on different branches in the resulting tree. (See page 62.)

¹⁰This is analogous to the following situation. Suppose we have the formula $\forall x(Ax \& Bx)$. We may replace this with the following equivalent formula $\forall x Ax \& \forall y By$.

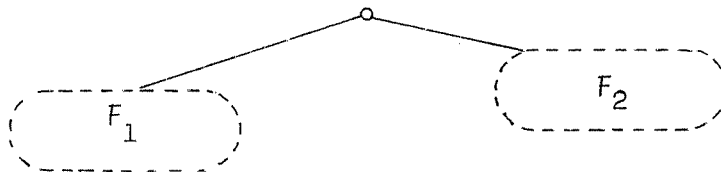
O_7 - the operator proper subforest copy

Let F be a proper forest of the ungrounded tree q . The ungrounded tree, $O_7(F)$ is a tree which is determined in the following manner:

The forest F consists of trees t_1, \dots, t_n which may be ordered (i.e. t_1 precedes t_2 etc.) according to the relative positions of their origins in the tree q containing F . $O_7(F)$ is the ungrounded tree obtained by creating a node n and linking n to each origin of the copies of trees comprising F . Each node of the resulting tree except n , has content.

The dpf associated with $O_7(F)$ is a subforest of that associated with the tree q and consists of the nodes containing any variables occurring in F as well as all leaves dominated by these nodes. (Furthermore, any leaf m not included by the foregoing rules for which there exist a node $n \in N_F$ such that $\zeta(n) = m$ is also to be included.)

The operators of U combined with O_7 , provide a means of breaking down the goal of demonstrating the unsatisfiability of the initial spe into subgoals. Suppose, for example, that the tree q given below has proper subforest F_1 and F_2 .



The task of demonstrating the unsatisfiability of q may be broken into the tasks of demonstrating the unsatisfiability of F_1 and F_2 .

The notion of proper forest thus allows the delineation of subgoals, and O_7 allows any subgoal to be abstracted from the parent tree in which it occurs and dealt with independently. The copy operator is defined in such a way as to be applicable to any proper forest of any ungrounded tree. We may thus abstract subgoals of subgoals. In general, it is not necessary to abstract all subgoals at the same stage or level of a proof attempt. This is a useful property since some strategies, which use the logic of U , may follow several lines of attack simultaneously.

O_8 - the operator proper subforest removal

Suppose that s and t are ungrounded trees, F is a proper subforest of s , $t = O_7(F)$ and $t \xrightarrow{L} \phi$. The ungrounded tree $O_8(s)$ is obtained as follows:

For each branch b occurring in F , remove all branches of s which contain each of the nodes of b .

(The dpf for $O_8(s)$ is the same as the dpf for s .)

We note that if s contains subforests F_1, \dots, F_N such that $O_7(F_1) = O_7(F_2) = \dots = O_7(F_N) = t$, then the above definition allows O_8 to be applied N times to remove F_1, \dots, F_N from s . Furthermore, once it has been demonstrated that $t \xrightarrow{L} \phi$, then O_8 may be applied to any tree containing a proper subforest H such that $O_7(H) = t$. The operator O_8 thus provides U with the capability of utilizing lemmas since $t \xrightarrow{L} \phi$ may be regarded as a proof of the unsatisfiability of t . By the above observation, this subresult may be used as often as desired.

proving initial expressions with the operator set $\{0_1, 0_2, 0_3, 0_4, 0_5, 0_6, 0_7, 0_8\}$

The notions of "linear extension" and "proof of an initial expression" may be extended to include the operators 0_7 and 0_8 . Linear extension, as previously defined, was introduced both as an aid to defining the notion of proof, and as a means of talking about logical descendancy. It involved application of operators chosen from the set $L = \{0_1, 0_2, 0_3, 0_4, 0_5, 0_6\}$.

With the introduction of 0_7 and 0_8 , we need a means of talking about logical descendancy as well as proof of an initial expression.

If we define $L_1 = L \cup \{0_8\}$ and $L_2 = L \cup \{0_7, 0_8\}$ the notation $s \xrightarrow{L_1} t$, $s \xrightarrow{L_2} t$ parallels $s \xrightarrow{L} t$ and has the obvious meaning.

Different notions of linear extension are necessary for several reasons. The notation $s \xrightarrow{L_2} t$ is necessary in order to talk about the heredity of a given tree when the set of operators L_2 is available. However, since $\theta \xrightarrow{L_2} \phi$ does not in general imply that θ is unsatisfiable,¹¹ we need the notation $s \xrightarrow{L_1} t$ as an aid to defining the notion of provable with respect to the operator set L_2 .

If θ is the initial tree for the initial expression e and if $\theta \xrightarrow{L_1} \phi$ then we say that e is provable with respect to the operator set L_2 . (Note that the operator set must be L_2 rather than L_1 since application 0_8 implies previous application of 0_7 .)

¹¹Note that $s \xrightarrow{L_2} t$ implies that 0_7 may have been utilized to generate t . Since application of 0_7 to a proper subforest generates a tree whose unsatisfiability only implies the unsatisfiability of the subforest and not in general the parent tree, we see that $s \xrightarrow{L_2} \phi$ usually does not imply that s is unsatisfiable.

The operators restricted append and unsatisfiable branch removal

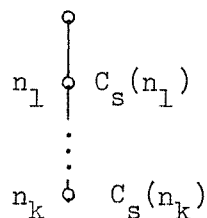
The final U -inference operators to be presented are the restricted append operator O_9 and unsatisfiable branch removal O_{10} . Restricted append takes two ungrounded trees, which satisfy certain conditions, and produces an ungrounded tree which is the append of the two (see page 16). The conditions imposed upon the operand trees ensure that the tree resulting from the restricted append will be satisfiable if the operand trees are simultaneously satisfiable. This requirement is necessary if U is to be consistent.

The conditions which the operand trees must satisfy in order for O_9 to be applicable depend upon the notation of "patriarch." "Patriarch" is another hereditary notion which allows us to talk about the set of expressions necessary and sufficient for the generation of a given ungrounded tree.

The operator O_{10} is used in conjunction with O_9 (in a fashion analogous to the way O_8 is used with O_7) to remove branches containing sets of patriarchs shown to be unsatisfiable via a deduction of ϕ using the operator O_9 .

patriarchs

Suppose s and t are ungrounded trees such that $s \xrightarrow{L_1} t$. Further suppose that s has the form

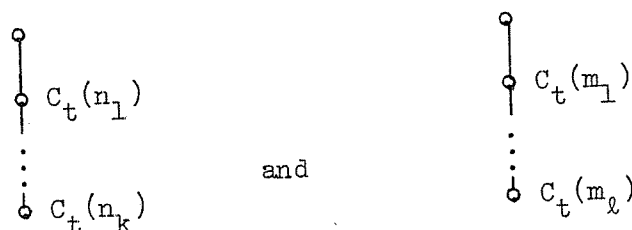


where $i \neq j$ implies n_i is not a linear descendant of n_j . Under these conditions we say that $\{C_s(n_i): 1 \leq i \leq k\}$ is a set of patriarchs for t .

Suppose t_1 and t_2 are ungrounded trees having sets of patriarchs s_1 and s_2 respectively. Let us say that the ungrounded tree t can generate t_1 and t_2 if the following conditions hold:

1. Some branch of t contains (among others) the nodes $n_1, \dots, n_k; m_1, \dots, m_\ell$ which have the property that $\{C_t(n_i): 1 \leq i \leq k\} = s_1$ and $\{C_t(m_i): 1 \leq i \leq \ell\} = s_2$.
2. $\{n_i: 1 \leq i \leq k\}$ and $\{m_i: 1 \leq i \leq \ell\}$ each determine a proper subforest of t .

We note that if t can generate t_1 and t_2 , then O_7 can be applied twice to t to produce



Since $\{C_t(n_i)\}$ and $\{C_t(m_i)\}$ are patriarchs of t_1 and t_2 respectively, we see that $t \xrightarrow{L_2} t_1$ and $t \xrightarrow{L_2} t_2$. Thus if t can generate t_1 and t_2 , then either tree can be deduced from t within the framework of U . Furthermore, since all expressions of s_1 and s_2 are contained on the same branch of t , a tree which is essentially the same as the tree resulting from appending t_1 to some leaf of t_2 can be deduced from t if $s_1 \cap s_2 = \phi$.

Finally, we need to extend the notion of descendancy to cover derivations involving the operator O_g (restricted append) presented below. This notion, "derivable" deals with the heredity of a given ungrounded tree t and is used to relate t to a set of trees from which it may be produced under the full set of operators of U .

We say that an ungrounded tree t is derivable from the set of ungrounded trees Γ and write $\Gamma \Rightarrow t$ if

1. $t \in \Gamma$
- or 2. there exists an ungrounded tree s such that $\Gamma \Rightarrow s$
and $s \xrightarrow{L_2} t$
- or 3. there exist ungrounded trees t_1 and t_2 , sets of ungrounded trees Γ_1 and Γ_2 such that $\Gamma_1 \cup \Gamma_2 \subset \Gamma$,
 $\Gamma_1 \Rightarrow t_1$, $\Gamma_2 \Rightarrow t_2$ and t is the result of (restrictive) appending t_2 to some leaf of t_1 .

the operator O_g

Let t_1 and t_2 be non-empty ungrounded trees with sets of patriarchs s_1 and s_2 and suppose there exists an ungrounded tree t which can generate t_1 and t_2 and such that $\{\emptyset\} \Rightarrow t$. We then say that the restrictive append operator O_g is applicable to t_1 and t_2 and define the restrictive append of t_2 to the leaf n of t_1 as follows:

1. Let $\{x_i : 1 \leq i \leq k\}$ and $\{y_i : 1 \leq i \leq l\}$ be the sets of variables occurring in t_1 and t_2 respectively and let $\{v_i : 1 \leq i \leq k\}$, $\{w_i : 1 \leq i \leq l\}$ consist of variables of the same type as those contained in $\{x_i\}$,

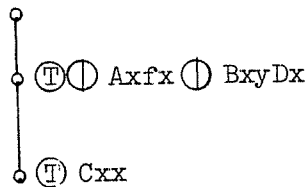
$\{y_i\}$, and be pairwise disjoint and not contain any of the variables contained in $\{x_i\}$, $\{y_i\}$. Define $\sigma = \{\dots, \langle v_i, x_i \rangle, \dots\}$ and $\rho = \{\dots, \langle w_i, y_i \rangle, \dots\}$.

2. Append t_2^ρ to the image of the leaf n in t_1^σ . The result is the ungrounded tree $O_9(t_2, n)$.

(Note that if $t_2 = \phi$ then $O_9(t_2, n)$ is the result of removing the branch determined by n from t_1 .)

If t_1 and t_2 have patriarch sets s_1 and s_2 , then $O_9(t_2, n)$ has the patriarch set $s_1 \cup s_2$. The dpf associated with $O_9(t_2, n)$ is the union¹² of the dpf's associated with t_1^σ and t_2^ρ .

example Referring to figure 1 (page 62) the forest determined by nodes 2 and 3 is proper. Thus, $O_7(F)$ is the ungrounded tree

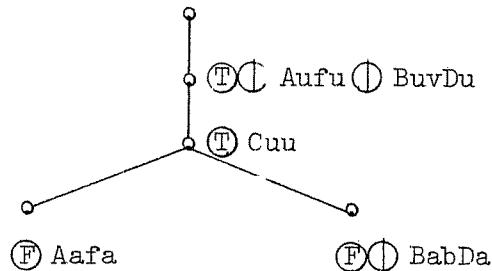


(where F is determined by the node set $\{2,3\}$.)

Furthermore, $\{\textcircled{T} \textcircled{A} \textcircled{Axfx} \textcircled{B} \textcircled{BxyDx} \textcircled{T} \textcircled{Cxx}\}$ is the patriarch set for $O_7(F)$. The tree given in figure 3 has the patriarch set $\{\textcircled{T} \textcircled{Axfx} \textcircled{BxyDx}\}$. Thus this tree and the tree $O_7(F)$ given above

¹²The union of two forests is the union of the sets of trees comprising the two forests. We impose an order on these trees as follows: All trees from $\text{dpf}(t_1)$ precede all trees from $\text{dpf}(t_2)$. We assume that the trees of $\text{dpf}(t_1)$ and $\text{dpf}(t_2)$ are already ordered.

can be generated from the tree t given in figure 1 (right hand tree). If we define θ as $\circledR \circledR \circledR \text{Axfx} \circledR \text{BxyDxCxx}$ we see that $\{\theta\} \Rightarrow t$. We may thus apply restricted append to $O_7(F)$ and the tree given in figure 3 to produce



which has the patriarch set $\{\circledR \circledR \text{Axfx} \circledR \text{BxyDxCxx}, \circledR \circledR \circledR \text{Axfx} \circledR \text{BxyDxCxx}\}$

O_{10} - unsatisfiable branch removal

Suppose that by using the operators at hand we can demonstrate $\{\theta\} \Rightarrow \phi$. Can we then conclude that θ is unsatisfiable? In general, the answer is no for the derivation of ϕ from θ may include applications of the proper subforest copy operator.¹³

Fortunately, if by previously given rules, a set of patriarchs has been associated with ϕ , we may take advantage of the fact that ϕ has been derived to remove branches from any trees which possess branches containing all elements of this set. This is accomplished using the O_{10} operator.

Suppose $\{\theta\} \Rightarrow \phi$ and that π is a set of patriarchs for this particular occurrence of ϕ . Further, let t be any ungrounded tree.

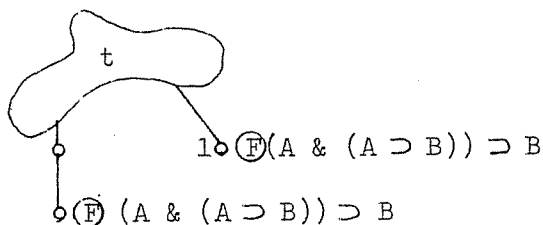
¹³It will be recalled that if F is a proper subforest of s , $t = O_7(F)$ and t is unsatisfiable, then we can conclude only that F is unsatisfiable. Thus, if s contains non-closed branches which are not common to F , then the unsatisfiability of t does not imply the unsatisfiability of s .

If t possesses any branch b which contains all expressions of s , then remove b from t . The result is denoted by $O_{10}(t)$. The dpf for $O_{10}(t)$ is the same as that for t .

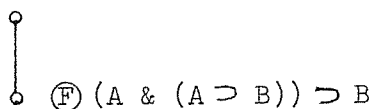
With the introduction of O_{10} (which behaves in a manner similar to O_4) we extend the various hereditary notions previously presented to include O_{10} by redefining L_1 as $L_1 \cup \{O_{10}\}$.

example In this example we shall consider a derivation containing expressions of the propositional logic. This particular example involves the connectives $\&$ and \supset . We note that rules similar to O_2 could be specified for these connectives. (In fact, this is done in [43].) For the purposes of this example, we shall assume them; their structure is fairly obvious.

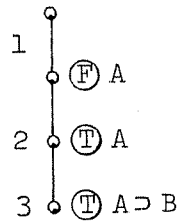
Suppose a tree s of the form



is derivable from some initial tree θ . The operator O_7 applied to the proper subforest determined by node 1 produces the tree



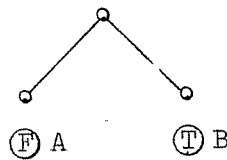
which linearly extends¹⁴ to the tree t_0



the operator O_7 may be applied to the proper subforests determined by $\{1,2\}$ and $\{3\}$ to produce the respective trees

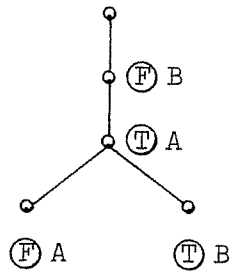


The latter linearly extends to the tree t_2

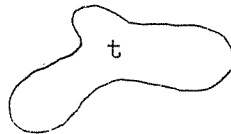


Now $\{(F) B, (T) A\}$ is a partriarch set for t_1 and $\{(T) A \supset B\}$ for t_2 . Since t_0 possesses a branch (its only branch) which contains each of the exhibited expressions, we may append t_2 to t_1 using the O_9 operator. This produces the tree t_3

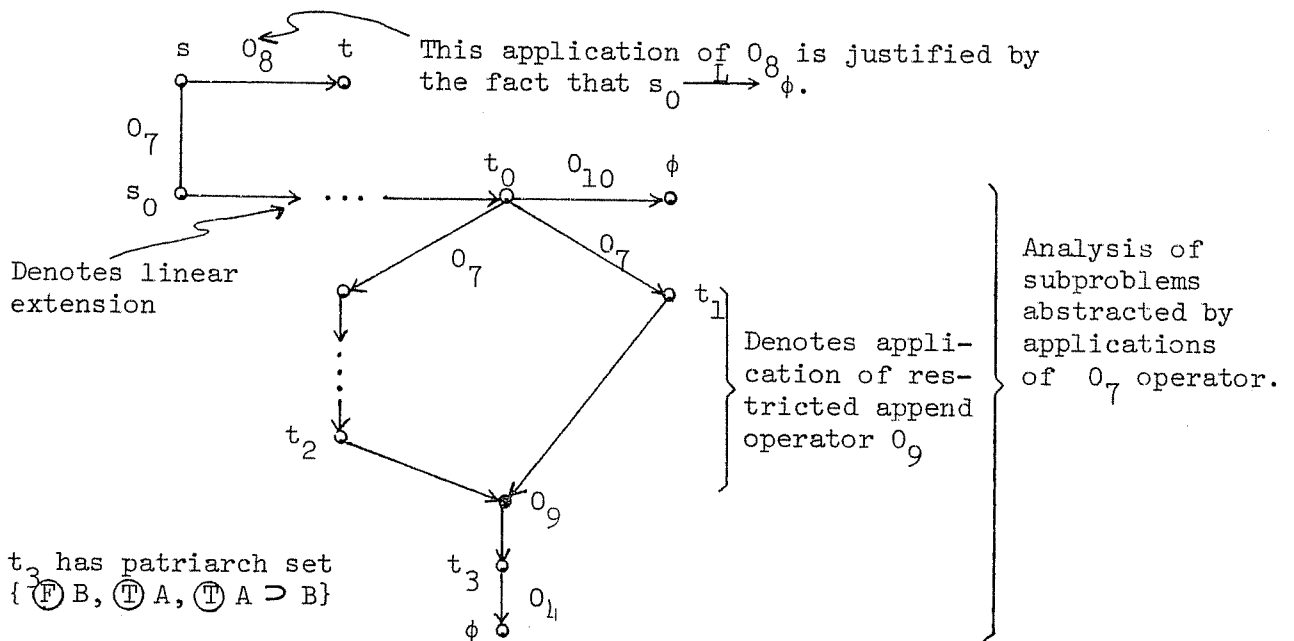
¹⁴The operators, which could be specified for the connectives $\&$ and \supset , behave in a manner similar to O_7 . Thus linear extension could be extended to these in the obvious manner.



which linearly extends (using one application of O_4) to the empty tree ϕ . By the preceding remarks, this particular occurrence of ϕ has associated the patriarch set $\{(F) B, (T) A, (T) A \supset B\}$. Thus O_{10} may be applied to the single branch of t_0 to produce the empty tree ϕ . The operator O_8 may then be applied twice to the tree s to produce t .



This entire example may be summarized in the following diagram:



the current attempted proof graph for an initial tree Θ

The schematic diagram given above provides an example of a structure called an attempted proof graph. An attempted proof graph is a particular kind of directed graph used to explicitly exhibit the heredity of the various ungrounded trees generated in the process of attempting to prove an initial expression. Actually there is not just one attempted proof graph, but a sequence of such graphs. These record the history of any attempted proof and provide a record which is useful when attempting to ascertain whether or not certain inference operators apply. In addition, attempted proof graphs are utilized by the system described in appendix A.

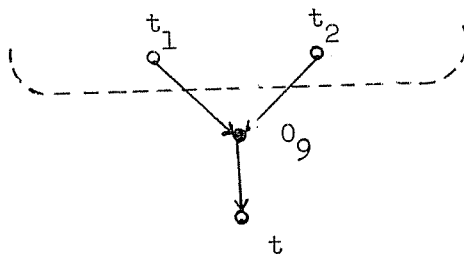
For the purposes of the logic U (and also E) it is only necessary to record the most recent attempted proof graph since previous graphs appear as subgraphs of the most recent graph. In fact, if we take the graph consisting of a single node whose label is Θ as the initial attempted proof graph, succeeding graphs are produced constructively by rules given below.

The graph depicted above is actually only part of an attempted proof graph since the heredity of the tree s is not indicated and certain linear subgraphs are only suggested. In general, an attempted proof graph consists of nodes labelled with the names of ungrounded trees and edges labelled with the names of inference operators. (The treatment of restricted append is an exception to this rule and is described below.)

Let the initial attempted proof graph be as described above and

suppose that the current attempted proof graph T_N ($N = 1, 2, \dots$) is defined. (T_1 is the initial attempted proof graph.) T_{N+1} is constructed as follows:

1. If s is an ungrounded tree occurring as a label of some node of T_N and application of the operator $O_k \in \{O_1, \dots, O_{10}\} - \{O_9\}$ to s produces an occurrence of the ungrounded t then not in T_N , add a single node to T_N labelled with t and add a directed edge labelled with O_k from the node labelled with s to the node just added.
2. If t_1 and t_2 are ungrounded trees occurring as labels of two nodes of T_N and application of O_9 (restricted append) produces an occurrence of the ungrounded tree t then not in T_N add the following structure to T_N



i.e. add the auxilliary node (shaded) with label O_9 , the node labelled with t and the three indicated directed edges.

The following example illustrates some of the notions presented in this section. It represents a \mathcal{U} -proof of e where e is the \mathcal{U} -equivalent of the given first order formula.

example of a \mathcal{U} -proof The spe $\textcircled{F} \textcircled{\perp} Afy \textcircled{\perp} AvgAwh$ is the \mathcal{U} version of the first order predicate calculus formula $\neg (\exists x \forall y Axy \supset \forall z \exists w Azw \ \& \ \forall u \exists v Auv)$. f, g and h are Skolem functions of zero arguments and x, y, z, u, v, w are positive variables.

The initial tree θ is $\textcircled{F} \textcircled{\perp} Afy \textcircled{\perp} AvgAwh$

Linear extension utilizing the \mathcal{O}_2 operators $\textcircled{F} \textcircled{\perp}$ and $\textcircled{T} \textcircled{\perp}$ produces the tree t_3 .

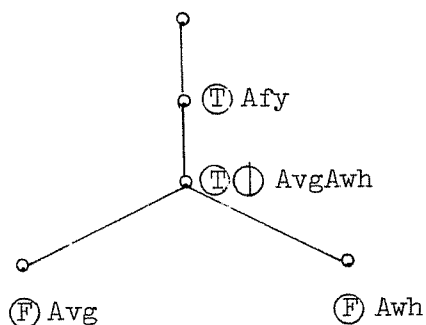
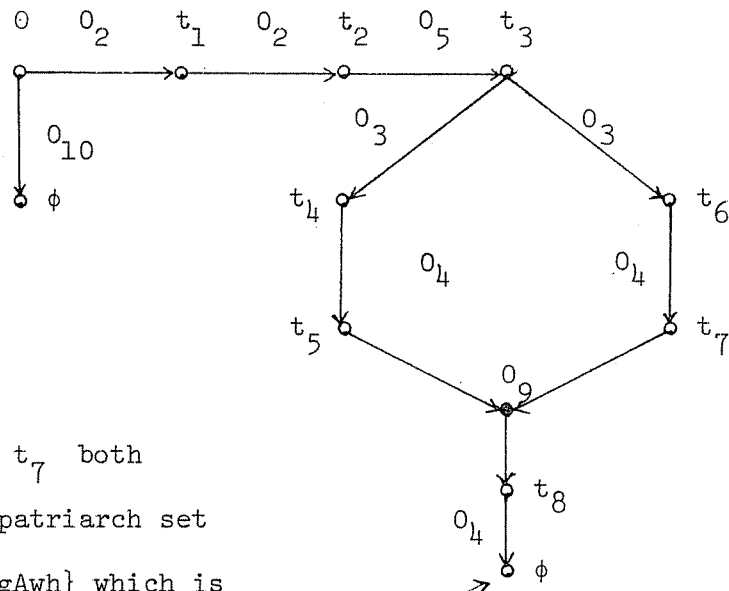


Figure 6

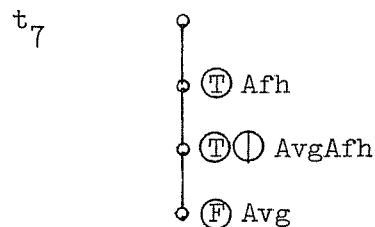
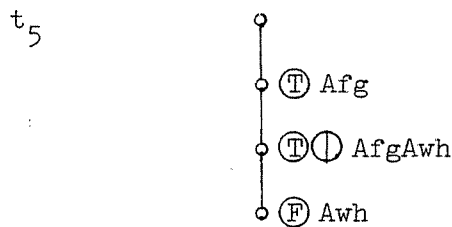
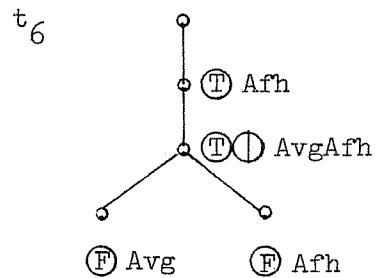
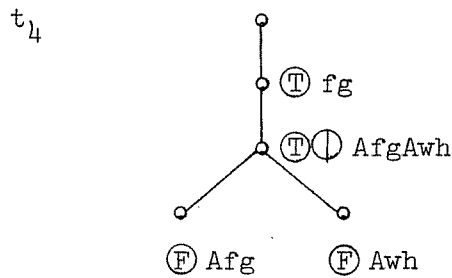
An attempted proof graph is given below.

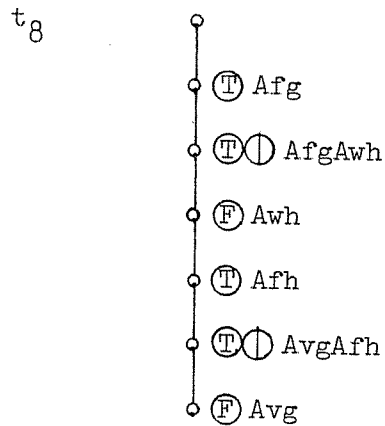


Note t_5 and t_7 both have the same patriarch set $\{\textcircled{F}\textcircled{\perp} \text{Afy} \textcircled{\perp} \text{AvgAwh}\}$ which is therefore the patriarch set for

This justifies the application of 0_{10} to θ .

The various ungrounded trees occurring in the attempted proof graph are as follows:





Since $\theta \xrightarrow{L_1} \phi$ (i.e. $\theta \xrightarrow{\{O_{10}\}} \phi$) we see that the initial spe $(F)(I)Afy(I)AvgAwh$ is unsatisfiable. If e had been the initial expression producing this initial spe, then e would be provable, and, therefore, valid.

The syntactical idea of U -provable expression is related to the semantic ideas of unsatisfiable set and valid expression. These relations are developed here for the operator set $\{O_1, \dots, O_{10}\}$. They extend to the operator set $\{O_1, \dots, O_{12}\}$, (where O_{11} and O_{12} are inference operators presented in section 3.)

The formal system U with operators O_1, \dots, O_{10} is consistent in the sense that if a closed type 1 expression e is provable then it is valid. The consistency of U can be demonstrated using the following lemmas:

lemma 1 If t is true under some valuation v and s results from the application of some operator of $M = \{O_2, O_3, O_4, O_5, O_6, O_7, O_8, O_9, O_{10}\}$ to t , then s is true under v . If t is satisfiable and s results from the application of O_1 to t , then s is satisfiable.

proof (See appendix I.) Note that if t is true under v , then t is satisfiable.

lemma 2 If t is satisfiable and $t \xrightarrow{L_1} s$, then s is satisfiable.

proof (Induction on the number of trees n occurring in the deduction $t \xrightarrow{L_1} s$.)

If $n = 2$ then lemma 2 follows by lemma 1. Thus, assume that lemma 2 holds for any deduction $t_1 \xrightarrow{L_1} s$ of length n and let $t \xrightarrow{L_1} s$ be a deduction of length $n + 1$. There exists an ungrounded tree t_1 such that $t \xrightarrow{L_1} t_1$ and s is obtained from t_1 by application of an operator from $M \cup \{O_1\}$. Thus t_1 is satisfiable by the induction hypothesis and s is satisfiable by lemma 1. Therefore, lemma 2 holds for an arbitrary deduction $t \xrightarrow{L_1} s$.

Theorem (consistency) If $\theta \xrightarrow{L_1} \phi$, then θ is unsatisfiable.

proof This result follows immediately from lemma 2 and the fact that ϕ is unsatisfiable.

The development of the inference operators of U has been motivated by the semantic idea of truth-function analysis. The operators and the associated notions of linear extension and derivability embody these ideas and provide a syntactical mechanism for establishing the provability of initial expressions. The fact that the formal system U is consistent simply reaffirms the tie between the formal system and its semantic foundations.

3.0 The extended system E

U_L -expressions representing interesting relationships have one striking feature in common - they tend to be unreadable. Moreover, as one gets further from pure logic into areas of application, this tendency increases since complicated expressions, which represent concepts central to the area, tend to reoccur. The predicates of equality and transitivity provide a simple but apt illustration of this point.

Loosely speaking, we may say that two entities x and y are equal if anything we may predicate of one may be predicated of the other. This may be represented in U_L by the expression

$$(1) \quad \begin{aligned} & \forall F \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc Fx Fx \bigcirc Fy Fy \bigcirc Fx Fx \bigcirc Fy Fy \bigcirc \bigcirc \bigcirc \bigcirc Fy Fy \bigcirc Fx Fx \bigcirc \\ & Fy Fy \bigcirc Fx Fx \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc Fx Fx \bigcirc Fy Fy \bigcirc Fx Fx \bigcirc Fy Fy \bigcirc \bigcirc \bigcirc \bigcirc Fy Fy \bigcirc \\ & Fx Fx \bigcirc Fy Fy \bigcirc Fx Fx \end{aligned}$$

Let us denote this expression by $G(x,y)$. Now suppose we wish to represent the fact that equality is transitive, i.e. for arbitrary u, v and w if u equals v and v equals w then u equals w . Schematically we may represent this as:

$$(2) \quad \forall u \forall v \forall w \ G(u,v) \text{ and } G(v,w) \text{ implies } G(u,w)$$

The U_L representations of "and" and "implies" are not particularly complicated, but the expression which results from replacing the three occurrences of equality in the above expression is.

The extension of U , presented in this section, provides a way around this difficulty by providing for the introduction of defined constants. These defined constants may be nested, thus allowing concise formulation of concepts which in U_L would be unrecognizable. For example, in E we might define "=" as follows¹

$$\lambda xy \bullet \forall F (Fx \equiv Fy)$$

and Trans (transitivity) as

$$\lambda H \bullet \forall u \forall v \forall w ((Huv \ \& \ Hvw) \supset Hwv)$$

Thus allowing (2) to be represented as

$$(3) \quad \text{Trans}(=)$$

The inference rules of E allow any expression such as (3) to be replaced with an equivalent expression in which prescribed defined constants have been replaced with primitive symbols or other defined constants. (We note that in these illustrations, \equiv , $\&$ and \supset are defined constants also.) The degree of unabbreviation is completely flexible and in the example just given could range from one step of unabbreviation applied to Trans in (3) yielding

$$\forall u \forall v \forall w (((u = v) \ \& \ (v = w)) \supset (u = w))$$

to the expression which would be obtained by replacing all occurrences of $G(x,y)$, "and" and "implies" in (2) with their U_L -representations.

¹For the purpose of illustration we use infix notation rather than the actual prefix notation of E .

We note that the complexity of a given non-primitive concept depends heavily upon our choice of primitive logical concepts. Thus, the complexity of (1) is due to our choice of \oplus as a logical primitive. The choice of some other set of primitives does not change matters though, since as we precede into arbitrary areas of application, any fixed set of primitives will sooner or later prove as cumbersome as \oplus did in this example.

What is needed is not some large set of predetermined logical primitives but rather the facility of introducing appropriate constants as a given area of application makes their need apparent. This facility, combined with a mechanism which allows newly introduced constants to function as if they had originally been included among the primitives of U is what gives the extended system E its power.

The extended system E is obtained from U in two stages. The first stage consists of a minor modification to the language U_L . The second stage, which is more extensive, consists of adding several inference operators to the set $\{O_1, \dots, O_{10}\}$. We begin the extension of U with a discussion of unabbreviation.

3.1 Definitional extension of an object language

In the following discussion we shall be considering three distinct languages: OL the given object language, EOL the extension of OL containing formulas with abbreviations, and DL the language in which definitions are written.

The general framework in which we intend to set definitional extension does not depend upon the particular object language being extended nor the form chosen for definitions. In order that this framework be presented in a familiar context, we begin the discussion of definitional extension with an example in the language of the first order predicate calculus found in Kleene's I.M. [18]. This language which we shall call H , consists of well formed formulas constructed with the operators \neg , \vee , $\&$, \supset , \exists , \forall .

Consider the H -formula-like expression \hat{F}

$$\hat{F} = \forall y(M(y) \& \underbrace{\exists! x(P(x,y) \vee \exists zQ(x,z))}_A)$$

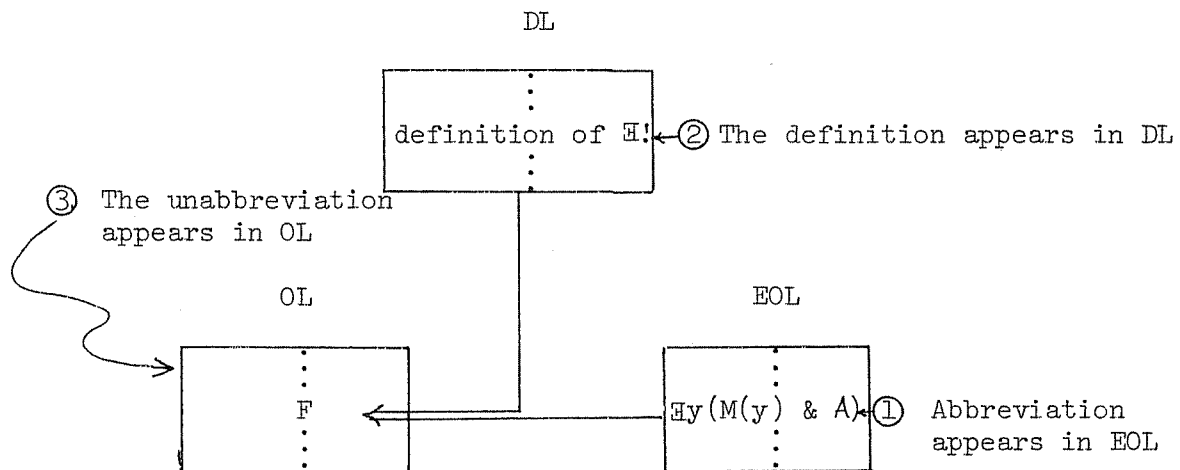
which involves the abbreviation¹ A and unabbreviates² to the H -formula F

$$F = \forall y(M(y) \& \exists x(P(x,y) \vee \exists zQ(x,z)) \& \forall w \forall u((P(w,y) \vee \exists zQ(w,z)) \& (P(u,y) \vee \exists zQ(u,z))))$$

Strictly speaking, \hat{F} is not a formula of H since it contains a symbol ($\exists!$) not in the vocabulary of H . It is, however, structurally similar to formulas of H . \hat{F} will be considered to be a formula of a new language (EOL for H denoted \hat{H}) which allows OL-like formulas over an extension of the OL-vocabulary.

¹The term abbreviation will be precisely defined in 3.2

²We shall take as the definition of $\exists! zM(z)$ the expression $\exists xM(x) \& \forall y \forall z(M(y) \& M(z) \supset y = z)$ although we do not specify the mechanics of unabbreviation at this time.



Abbreviations provide a concise representation for lengthy object language expressions. Total unabbreviation³ provides a means of getting from an EOL expression to some OL expression. Thus, unabbreviation ultimately produces a formula of the object language. This suggests that a reasonable way to extend the OL is by modifying the definition of OL formula to include abbreviations. We note that prior to unabbreviation an abbreviation such as A will appear as an unanalyzed object of the EOL in much the same way as an atomic formula appears as an unanalyzed object of the OL. This suggests that an appropriate formula modification can be obtained by specifying a set of abbreviation formation rules and extending the OL notion of atomic formula to include abbreviations.

In the case of arbitrary OLs, the framework for definitional extension will consist of three languages (OL, EOL, DL) bound together in the manner indicated by the example. In its most rudimen-

³Replacement of all defined constants with appropriate expressions which contain only primitive symbols of the logic.

tary form, definitional extension consists of extending the OL by introduction of a single new constant c_1 . (E! was such a constant in the above example.) The EOL for this extension is structurally similar to OL but contains formulas constructed with the help of c_1 . The DL will consist of a single expression which is the definition of c_1 . Such an extension will be called unit extension.

In general, the OL will be extended by introducing a sequence of defined constants c_1, c_2, \dots . Such an extension may be thought of as arising from a sequence of unit extensions⁴ where the OL for the introduction of c_i is the EOL for the introduction of c_{i-1} .

F is an abbreviation if F is a subformula of a formula⁵ F' which contains a specified occurrence of some c_i and F is the smallest subformula of F' containing this occurrence of c_i . (In general, any subformula which contains a specified occurrence of a symbol and is the smallest subformula containing the occurrence of the symbol is called the completion of the symbol.)

As noted earlier, abbreviations and atomic formulae are equivalent with respect to the manner in which they enter into EOL-formula construction. Thus, differences between an OL and its EOL arise solely from differences in the formation rules for abbreviations and

⁴This assumes that the abbreviation formation rules are independent of the particular constants used in the extension.

⁵Throughout this section, the term "formula" will be used in general discussions to denote a syntactical object of some specified language (OL, EOL, etc.) whose semantic interpretation is either "truth" or "falsity". In specific cases, the precise meaning will be clear from the context. Thus, in the case $OL=H$ then "formula" will have the meaning given in I.M. [13] and in the case $OL=L$, "formula" will be taken to mean type 1 expression. Similar remarks also hold for "subformula".

atomic formulae. Just how different abbreviations and atomic formulae are will depend upon the particular characteristics of the given OL, EOL, and DL.

Unabbreviation of a specified abbreviation F occurring in a formula G and containing c_i is a form of subexpression replacement in which F is replaced by an expression which is determined by the definition of c_i , the subexpressions of F , and under certain conditions, the G subexpressions which contain F .

In U , the replacement performed in unabbreviating a type 1 expression containing defined constants has the properties of either logical equivalence or equality depending on the type of the defined constant. This will be considered in greater detail under "semantic considerations." (cf. 3.2).

3.2 The language E_L

The general considerations of the last section may be applied to the special case of extending the type theory U . The extension obtained will be denoted by E . In the terminology of the last section, U_L is the OL, E_L the EOL, and D_L the DL.

the extended object language The language extension of U_L (i.e. E_L) results from a simple modification of U . The vocabulary of U is extended so as to contain constants, which enter into expression formation in exactly the same way as do elements of V and S . In practice, any such constants occurring in an expression will also occur in the DL as definientes, in which case they are called defined constants.

The extension of U is obtained by modifying the U -vocabulary rule as follows:

1. For any type - a denumerable set of
 - a) variables (V)
 - b) Skolem function symbols (S)
 - c) constants (C)

where V , S and C are pairwise disjoint and TSS is taken to mean $V \cup S \cup C$.

If c is a defined constant (i.e. $c \in C$ and is defined in \mathcal{D}_L) occurring in a type 1 expression (or a pe or spe) G , then the smallest type 1 expression F containing c is called an abbreviation. The constant c is said to be an associate of F . An abbreviation F occurring in G is said to be maximal (with respect to G) if it is not a subexpression of any other abbreviation occurring in G . Note that we could have defined maximal λ -expressions of a type 1 expression $G \in U_L$ in an analogous manner.¹

Dependency forest calculations, for type 1 expressions of U_L , treated atoms and maximal λ -expressions alike. Dependency forest calculations, for type 1 expressions of E_L , treat atoms, maximal λ -expressions and maximal abbreviations alike. We shall thus refer to these collectively as pseudo atoms.

¹Let us define a λ -abbreviation as the smallest type 1 expression containing a particular occurrence of an expression of the form $\lambda x_1 \dots x_n \bullet e a_1 \dots a_n$. A maximal λ -expression can then be defined as a λ -abbreviation which is not contained in any other λ -abbreviation. (All expressions are considered relative to some expression G of which they are subexpressions.)

The dpf for a type 1 expression e of E_L is calculated from the structural representation of e in exactly the same way as was done for type 1 expressions of U_L . The structural representation of e is determined by applying algorithm A of section 2.1.1 with step A1 modified as follows:

A'1.1. If e is a pseudo atom, then $T(e)$ is set .

Excepting occurrences of abbreviations, E_L is the same as U_L . The E_L definitions of syntactical categories such as closed expression, type 1 expression, atomic expression, pe and spe are the same as given for U_L .

The structural representation of an expression, which is defined for any pe or type 1 expression, determines a mapping from the pseudo atoms of the expression onto the set of leaves of the dependency forest. Because duplicate branches are removed during calculation of the dependency forest, this mapping will not be 1:1 in general. (cf. p. 56). In what follows, it will be assumed that given an spe e , the following hold:

1. The associated dependency forest of e can be determined.
2. Given a pseudo atom of e , a unique leaf of $\text{dpf}(e)$ is determined.
3. Given a leaf of $\text{dpf}(e)$, a set of pseudo atoms of e is determined and distinct leaves determine distinct sets of pseudo atoms.

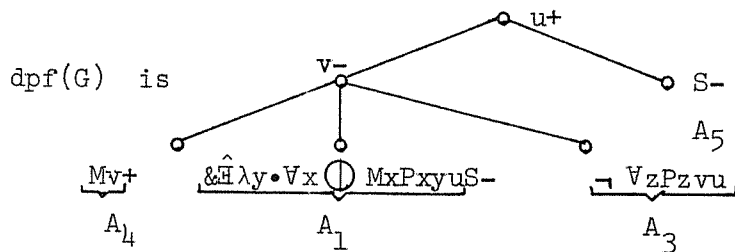
example 1

The closed type 1 expression G illustrates many of the concepts given above.

$$G = \forall u \circlearrowleft \forall v \circlearrowleft \underbrace{Mv \circlearrowleft}_{A_4} \& \underbrace{\hat{\exists} \lambda y \bullet \forall x \circlearrowleft MxPxyuS}_{A_2} \underbrace{\neg \forall zPzvuS}_{A_3} \underbrace{S}_{A_5}$$

A_1

A_1 , A_2 and A_3 are abbreviations with associates $\&$, $\hat{\exists}$ and \neg . A_1 and A_3 are maximal abbreviations. A_4 and A_5 are atoms. S has type 1, M type (0:1), P type (0,0,0:1), $\hat{\exists}$ type ((0:1):1), \neg type (1:1), $\&$ type (1,1:1) and u, v, x, y, z type 0. The defined constants \neg , $\&$, and $\hat{\exists}$ appear in the DL as will be seen presently.



Note that v depends upon u and that u occurs in A_3 , G is Skolemized with the aid of $\text{dpf}(G)$ yielding G_s .

$$G_s = \circlearrowleft \circlearrowleft Mfu \circlearrowleft \underbrace{\& \hat{\exists} \lambda y \bullet \forall x \circlearrowleft MxPxyuS}_{A_1} \underbrace{\neg \forall zPzfuS}_{A_3}$$

Note that both occurrences of v have been replaced by fu . However, quantifiers appearing within an abbreviation are left intact.

example 2

$$G_1 = \textcircled{T} \vee y \ \& \ M y \ \hat{E}! \ \lambda x. \vee \ P x y \ \hat{E} \ \lambda z. \ Q x z$$

is an example of a pe in E_L .

Here we have assumed that P, Q have type $(0,0:1)$, \vee has type $(1,1:1)$ and $\hat{E}!$ has type $((0:1):1)$. Note that $\&$ and \vee are elements of C and occur in subexpressions of G by rule R2a. This polarized expression is the EOL version of the formula \hat{F} given at the beginning of this chapter. Close inspection will reveal that \hat{E} is not the same as E and $\hat{E}!$ is not the same as $E!$. Skolemization of G_1 gives:

$$T(G_1) \text{ is } \begin{array}{c} \circ \quad \vee \\ | \\ \circ \quad y+ \\ | \\ \circ \end{array} \quad \text{dpf}(G_1) \text{ is } \begin{array}{c} \circ \quad y+ \\ | \\ \circ \end{array}$$

$$\& \ M y \ \hat{E}! \ \lambda x. \vee \ P x y \ \hat{E} \ \lambda x. \ Q x z + \quad \& \ M y \ \hat{E}! \ \lambda x. \vee \ P x y \ \hat{E} \ \lambda z. \ Q x z +$$

$$G_s = \& \ M y \ \hat{E}! \ \lambda x. \vee \ P x y \ \hat{E} \ \lambda z. \ Q x z$$

the definition language The definition language D_L for the extended system E is based upon the components of E_L . We begin by singling out a subset of E_L -closed expressions which we shall call pseudo type 1-expressions. These are of interest since they occur in definitions and may contain implicit dependency information. As in the case of type 1 expressions, a useful notion of dpf may be defined for pseudo type 1-expressions.

Pseudo type 1-expressions are closed expressions of the form $\lambda x_1^1 \dots x_{n_1}^1 \bullet \dots \lambda x_1^k \dots x_{n_k}^k \bullet e$ in which e is an expression of type 1 whose first character is not λ . The character string $\lambda x_1^1 \dots x_{n_1}^1 \bullet \dots \lambda x_1^k \dots x_{n_k}^k \bullet$ is called the prefix, the substrings $\lambda x_1^i \dots x_{n_i}^i \bullet$ segments and e the kernel. In particular, we allow pseudo type 1-expressions with empty prefixes (i.e. in which the prefix is the empty string).

A pseudo type 1-expression e_1 , of the form $\lambda x_1^1 \dots x_{n_1}^1 \bullet \dots \lambda x_1^k \dots x_{n_k}^k \bullet e$ can occur in a given pe in various ways. We are particularly interested in the case where e_1 occurs as an operator (cf R2) in a type 1 subexpression $e_1 a_1^1 \dots a_{n_1}^1 \dots a_1^k \dots a_{n_k}^k$. In this case, repeated applications of λ -reduction produce $e_1 \sigma$ where $\sigma = \{\dots, \langle a_{j_i}^i, x_{j_i}^i \rangle, \dots\}$ ($i = 1, \dots, k$ and $j_i = 1, \dots, n_i$). In general, such an occurrence of e_1 will cause implicit dependencies between the variables occurring in e which are bound in e and those bound outside of e (excluding $x_1^1, \dots, x_{n_k}^k$). The λ -rule (O_1), as defined for U , takes such dependencies into account, however, some of the dpf's necessary to perform λ -reduction are calculated at the time λ -reduction is applied. Thus, in λ -reducing several variant² subexpressions of an expression, the same dpf³ will be calculated more than once.

²Two subexpressions of the forms $\lambda x_1^1 \dots x_{n_1}^1 \bullet \dots \lambda x_1^k \dots x_{n_k}^k \bullet e_1$ and $\lambda y_1^1 \dots y_{n_1}^1 \bullet \dots \lambda y_1^k \dots y_{n_k}^k \bullet e_2$ are said to be variants of each other. e_1 and e_2 are such that there exists a substitution σ of the form $\{\dots, \langle x_i, y_i \rangle, \dots\}$ where $x_i, y_i \in V$ and $e_1 \sigma = e_2 \sigma$.

³Same up to substitution of expressions for the variables occurring in the subexpression's prefix.

In the case of a pseudo type 1-expression e , it is possible to calculate a dpf for e prior to λ -reduction and then use the dpf for any λ -reduction involving a variant of e . The unabbreviation operation presented in section 3.3 and the λ -reduction operator already presented take advantage of this property by providing a more efficient means of calculating dependency forests than would be possible if dependency forests for pseudo type 1-expressions could not be pre-calculated.

definition in \mathcal{D}_L Elements of \mathcal{D}_L , which are called definitions, may contain pseudo type 1-expressions. A definition is a syntactical object of the form $c \longleftrightarrow f$ where \longleftrightarrow is a new symbol, $c \in C$ and f is obtained from the closed expression e (which has the same type as c) in the following manner: If e is not a pseudo type 1-expression, then $f = e$. Otherwise, we associate the dpf for the kernel of e with c and form f by removing from e all occurrences of $\forall v$ not occurring in a pseudo atom of the kernel. The defined constant c is called the definiens and the expression f the definiendum^{3.5}. To avoid multiple definition, we require that each definiens is distinct from any other definiens.

^{3.5}A closed expression e is an element of E_L . The logic of E deals with expressions of E_{L*} (i.e. elements of E_L which have been processed by the Skolemization algorithm). Rather than take the closed expression e as the definiendum of c , and then apply the Skolemization algorithm (including dpf calculation) each time c is unabbreviated, we precalculate the dpf and remove occurrences of " $\forall v$ " once - at the time the constant is first defined. The result is taken as the definition

Circularity can be avoided by the simple expedient of ordering the definientes and requiring that any newly introduced definitions be defined only in terms of primitive symbols and previously introduced definientes. Unfortunately, this precludes recursive definitions. In order to avoid such a limitation, we shall leave the general question of circularity open and not further constrain \mathcal{D}_L . Thus, in general, we have no guarantee that any finite sequence of unabbreviation operations applied to an arbitrary element of E_L will produce an expression of U_L . (For particular \mathcal{D}_L s it is often possible to prove that arbitrary E_L expressions unabbreviate to U_L expressions. All \mathcal{D}_L s considered in this paper will possess this property.)

example The pseudo type 1-expressions displayed below determine the definitions $d_1 - d_8$.

1. $\lambda P \cdot \bigcirc PP$
2. $\lambda PQ \cdot \neg \bigcirc PQ$ P and Q have type 1
3. $\lambda PQ \cdot \bigcirc \neg P \neg Q$
4. $\lambda PQ \cdot \vee \neg PQ$ P and Q have type (0:1)
5. $\lambda PQ \cdot \& \supset PQ \supset QP$
6. $\lambda xy \cdot \forall F \equiv Fx Fy$ x, y and z have type 0
7. $\lambda R \cdot \neg \forall x \neg Rx$ ^{3.6}
8. $\lambda R \cdot \hat{\&} \lambda x Rx \forall y \forall z \supset \& Py Pz = yz$

^{3.6}We note that \mathcal{D}_L does not allow direct definition of operators such as $\exists x$. Fortunately, we may define operators which have the same effect. Thus d_8 defines $\hat{\exists}$. We note that $\hat{\exists} \lambda x$ is then notationally equivalent to $\exists x$ in the sense that $\exists x Px$ and $\hat{\exists} \lambda x Px$ are semantically equivalent (assuming the usual interpretation for $\exists x$).

d_1	$\neg \longleftrightarrow$	$\lambda P. \bigoplus PP$	$\{oP - \}^{3.7}$
d_2	$\& \longleftrightarrow$	$\lambda PQ. \neg \bigoplus PQ$	$\{o \neg \bigoplus PQ + \}$
d_3	$\vee \longleftrightarrow$	$\lambda PQ. \bigoplus \neg P \neg Q$	$\{o \neg P-, o \neg Q-\}$
d_4	$\supset \longleftrightarrow$	$\lambda PQ. \vee \neg PQ$	$\{o \neg PQ + \}$
d_5	$\equiv \longleftrightarrow$	$\lambda PQ. \& \supset PQ \supset QP$	$\{o \& \supset PQ \supset QP+\}$
d_6	$= \longleftrightarrow$	$\lambda xy. = Fx Fy$	$\left\{ \begin{array}{l} \circ F+ \\ \circ = Fx Fy+ \end{array} \right\}$
d_7	$\hat{H} \longleftrightarrow$	$\lambda R. Rx$	$\left\{ \begin{array}{l} \circ x- \\ \circ Rx+ \end{array} \right\}$
d_8	$\hat{H}! \longleftrightarrow$	$\lambda R. \& \hat{H} \lambda x. Rx \forall y \forall z \supset \& Ry Rz = yz$	$\{o \lambda R. \& \hat{H} \lambda x. Rx \forall y \forall z \supset \& Ry Rz = yz + \}$

Figure 8

semantic considerations in E_L The valuation procedure for expressions of U_L and U_{L*} can be extended to the expressions of E_L . The extension is accomplished by requiring that a defined constant have the same value as its definiendum. This means that any defined constant which can be evaluated is associated with a fixed element of each derived domain whose specific nature is determined by the definiendum. Thus, defined constants are treated in the same manner as the primitive symbol \bigoplus . This is necessary if the

^{3.7}The dependency forest for the associated pseudo type-1 expression $\lambda P \bigoplus PP$ would be $\{o P-, o P-\}$. However, the duplicate branch provides no additional information and is thus removed. (cf algorithm B section 2.1.1)

association between definiens and definiendum introduced at the syntactical level is to extend to the semantic interpretation.

example Consider the definition d_L given previously. P has type 1. This determines the type of the definiendum and thus the type of \neg (i.e. type (1:1)). If \neg is to have the properties of negation, there is clearly only one acceptable value for \neg in any D^* and that value is $\{\langle\langle T \rangle, F \rangle, \langle\langle F \rangle, T \rangle\}$. The evaluation procedure produces this value as follows:

① is assigned the value $\{\langle\langle T, T \rangle, F \rangle, \langle\langle T, F \rangle, T \rangle, \langle\langle F, T \rangle, T \rangle, \langle\langle F, F \rangle, T \rangle\}$. Using valuation rule E_3 (for valuation of u_L -expressions) we obtain $S = \{\langle P \rangle, \langle F \rangle\}$ and $Q = \{\langle\langle T \rangle, F \rangle, \langle\langle F \rangle, T \rangle\}$ since ① PP has value F when P is assigned the value T and T when P is assigned the value F .

Thus, the evaluation procedure assigns a value to \neg which is consistent with our intended meaning for \neg .

We note immediately that expressions of E_L containing either undefined constants (i.e. elements of C which do not occur as definiens of elements of D_L), circularly defined constants or non-primitively defined constants: (see below) can not be evaluated.

We noted in section 2.1 3 that valuation relative to a domain D produced a function from u_L into the derived domain D^* . Valuation for E_L relative to a domain D produces a function from a subset of E_L into the derived domain D^* .

If D is a given non-empty domain, e an expression of E_L for which valuation is defined, and p is a value assignment for the tree variables occurring in e , then we shall denote the value of e in D^* relative to the value assignment p by $v_D^p(e)$, (or where D and p are irrelevant to the discussion at hand, by $v(e)$). Valuation (both for U_L and E_L) has the important property that if A_1 and A_2 are expressions, e_1 and e_2 are subexpressions of A_1 and A_2 respectively and A_2 is obtained by replacing e_1 by e_2 in A_1 and if $v(e_1)$, $v(e_2)$, $v(A_1)$, $v(A_2)$ are defined then $v(e_1) = v(e_2)$ implies $v(A_1) = v(A_2)$.

The pseudo type 1 expression $\lambda PQ \bullet \neg \bigoplus PQ$ which determined the definiendum of $\&$ in Figure 8, contains an occurrence of the defined constant \neg . This can be eliminated if \neg is replaced with the pseudo type 1 expression $\lambda P \bullet \bigoplus PP$. In general, we could replace more than one such occurrence of a defined constant in an expression by the defined constant's definiendum. It is convenient to have notation which expresses the relation between the original definiens and the expression obtained from it in this manner.

We write $c \iff e$ if

1. $c \iff f \in \mathcal{D}_L$ and f is obtained from the expression e as indicated under the formation rules for elements of \mathcal{D}_L .
- or
2. There exists an expression d such that $c \iff d$ and e is obtained from d by replacing some defined constant occurring in d with its definiendum.

If e is an expression of U_L and c a defined constant such that $c \iff e$ we say c is primitively definable. It can be shown that $c \iff e$ implies $v(c) = v(e)$. Thus, it follows that if c is primitively definable by e and by f then $v(e) = v(f)$.

Primitively definable defined constants may always be evaluated using the procedure previously set forth. Defined constants may fail to be primitively definable for several reasons. They may be circularly defined, i.e. there exists an expression $e \ni c \iff e$ and e contains occurrences of c , or for any e such that $c \iff e$ it may be the case that e contains constants which are not defined.

Because of the close relationship between semantic evaluation and truth-function analysis (as embodied in U_L and E_L inference operators) the logic of E is mainly concerned with expressions which contain primitively definable constants. The occurrence of non-primitively definable constants in E_L -expressions, while acceptable in certain cases⁴, will, in general, cause the expression to be unprovable⁵ in E . This is in keeping with the semantic nature of E (and U) which requires that type 1 expressions of E_L be meaningful in any arbitrary domain (i.e. they evaluate to either T or F).

⁴Suppose $c \in C$ but is not defined. The expression $\forall c \neg c$ ($c \forall \neg c$ in infix) is provable in E since the nature of c does not bear on the validity of $\forall c \neg c$.

⁵Not to be confused with unsatisfiable.

3.3 Inference operators of E

The inferential notions developed for U , including inference operators and attempted proof graph, may be applied without modification to E . Given an spe of E_L , the inference operators of U then provide a means of determining what value assignments for pseudo atoms of the spe could account for the assumed satisfiability of the spe. Truth functional analysis may continue if the pseudo atom is λ -reducible¹, however, if the pseudo atom is either an atom or a maximal abbreviation, there is presently no mechanism which allows further analysis. This is certainly reasonable in the case of an atom, since a truth-value assignment to an atom implies nothing about truth-value assignments for its type-1 subexpressions. Truth-value assignments to maximal abbreviations, on the other hand, do imply truth-value assignments for some of the abbreviation's type-1 subexpressions. Furthermore, if the first character of the abbreviation is a defined constant which is primitively definable, the semantic rules for E then allow evaluation of the abbreviation. It should thus be possible to define an inference operator within the current framework which produces the desired analysis since the truth-functional synthesis performed during semantic evaluation is the inverse of truth-functional analysis.

The inference operator O_{11} produces the required analysis by replacing a specified occurrence of a defined constant occurring in a tree t with its definiendum. Dependencies caused by the context

¹An expression is said to be λ -reducible if the λ -reduction rule O_1 can be applied.

of the constant's occurrence and the particular nature of the constant are explicitly exhibited in the dpf derived from $\text{dpf}(t)$. The following algorithm specifies how this derived dpf and the tree resulting from application of O_{11} are obtained.

Algorithm A (unit unabbreviation)

Let a_1 be a node of the tree t and suppose that the content of a_1 contains a specified occurrence of the defined constant c . Furthermore, assume that the content of a_1 has the form δe where δ is either \textcircled{T} or \textcircled{F} and let $m = \zeta(a_1)$.

case 1 The specified constant is the first character of e and has a pseudo type-1 expression for its definiendum.

Al.1 tree calculation For each $x \in \zeta^{-1}(m)$, replace the first character of the suffix of $C_t(x)$ with the definiendum of $c^{1a,2}$.

Al.2 dpf calculation Algebraically replace leaf m with the dpf associated with c^2 .

^{1a}We shall say that two dpfs conflict if they have any variables in common. Now suppose that G and H are conflicting dpfs and that x_1, \dots, x_n is an enumeration of all distinct variables common to both. Let z_1, \dots, z_n be distinct variables such that z_i has the same type as x_i and such that z_i occurs neither in G nor in H . We shall say that H has been revised relative to G if z_i is substituted for each occurrence of x_i in H . If H has been revised relative to G , then G and H no longer conflict. (If H is the dpf for the tree t , then we revise t whenever we revise H by calculating $t\sigma$ where $\sigma = \{\dots, \langle z_i, x_i \rangle, \dots\}$.)

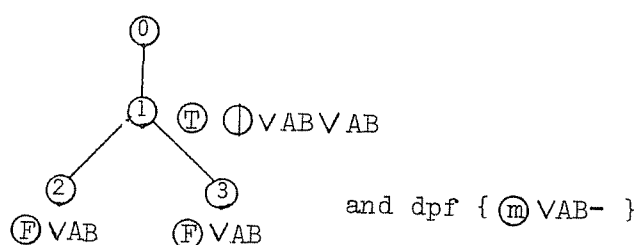
²It is assumed that the dpf associated with c has been revised prior to carrying out Al.1 and Al.2.

case 2 c does not have a pseudo type 1 definiendum

A2.1 tree calculation Replace the specified occurrence of c in $C_t(x)$ with the definiendum for c . Here $x \in \zeta^{-1}(m)$.

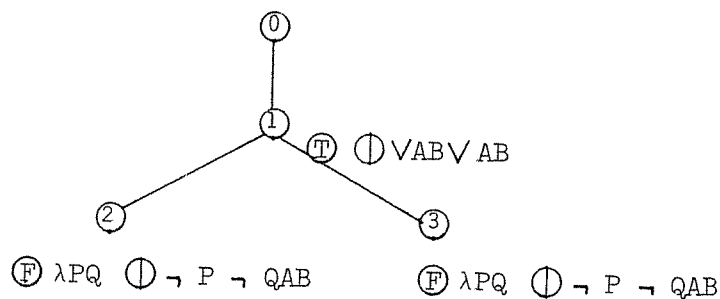
A2.2 dpf calculation Replace the specified occurrence of c in the content of leaf m^4 .

example Suppose we have the following tree



We apply O_{11} to node 3 with \vee the specified occurrence.

Step A1.1 produces



³Note that $x, y \in \zeta^{-1}(m)$ implies $C_t(x) = C_t(y)$ thus specifying an occurrence of c in $C_t(a_1)$ determines occurrences of c in each $C_t(x)$ where $x \in \zeta^{-1}(m)$.

⁴The content of m is the same expression as the suffix of $C_t(a_1)$.

Step A1.2 produces $\{o \neg P+, o \neg Q+\}$ (where $o \neg P+$ is a tree with one node whose content is $\neg P+$). Here we have used the dpf associated with v . (See page 103.) Note sign changes caused by leaf m being negative.

total vs. partial unabbreviation

There are two ways one can handle unabbreviation in a system such as E . The first is to remove all defined constants at the outset by preprocessing the initial expression. The second is to provide an incremental unabbreviation capability by including operators such as O_1 and O_{11} .

The former approach has the advantage of allowing us to work with expressions in which all dependencies can be explicitly represented by Skolem functions. This renders dependency forests and associated paraphernalia unnecessary, but at the expense of total unabbreviation.

Total unabbreviation is not always desirable. For, while it is true that some provable expressions require total unabbreviation in their proofs, there are many cases where total unabbreviation is not only unnecessary, but needlessly complicates the proof. The trivial example $c \vee \neg c$ was given previously. Not so trivial is the theorem $(a \neq b \ \& \ (\exists!Px \ \& \ Pa)) \supset \neg Pb$, which can be proved without unabbreviating " $=$ ". In general, certain basic theorems for each new concept are proved using unabbreviation; the further development of the concept uses the theorems and unabbreviation is not further needed.

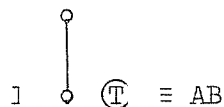
Unabbreviation generally leads to a marked increase in the complexity of the truth-function trees occurring in the attempted proof graph. This is due to the dependency of a tree's complexity on the number of connectives occurring in the initial spe. Unabbreviation has the property that it generally increases the number of such connectives. It is thus desirable to unabbreviate as little as possible.

The operators of \bar{E} allow partial unabbreviation, thus allowing possibly irrelevant structure in an expression to be suppressed until it is actually required. The hierarchical structure created by the inclusion of abbreviations in E_L can thus be examined in whatever detail is necessary.

3.4 Derived rules of inference and theorem utilization in \bar{E}

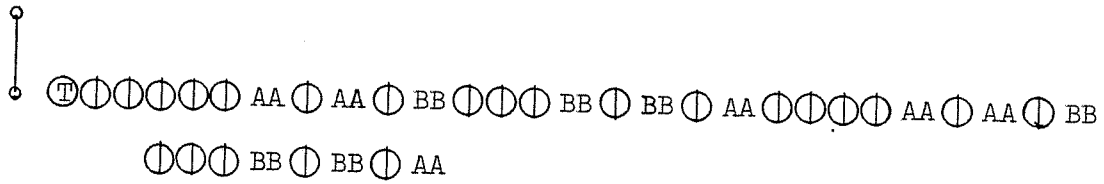
Introduction

The operators thus far presented allow the complete (if tedious) truth-function analysis of all primitively definable constants which may occur in any ungrounded tree with contents over spe's of E_L° . Consider, for example, the occurrence of \equiv in the ungrounded tree t_0 .

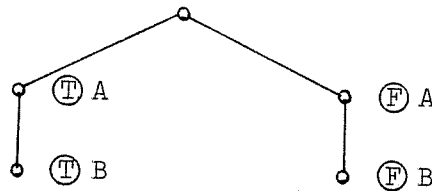


Thirteen applications of O_{11} (unit unabbreviation) followed by O_1 (λ -reduction) (i.e. twenty six operations in all) produces the tree

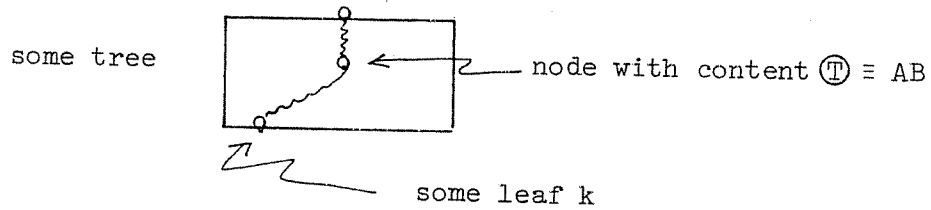
^oSkolemized expressions of E_L .



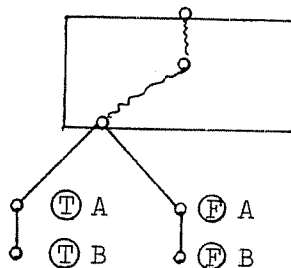
Finally, many applications of O_2 followed by duplicate branch removal, multiple closed branch removal, and node removal, produces the tree t_1



which is satisfiable if and only if t_0 is satisfiable. We note that had the expression $\oplus \equiv AB$ occurred in a more complex context such as



then the same sequence of operations would have produced



where the leaf k plays the role of leaf 1 of t_0 .

We make the following observations: 1) The form of t_1 depends only upon the definition of \equiv and the prefix \textcircled{T} ; 2) Total analysis of \equiv in arbitrary context, as discussed above, produces a tree which could have been obtained by appending t_1 to leaf k ; 3) If \equiv occurs frequently, total analysis of each occurrence by the sequence of operations used to obtain t_1 , will result in an excessive amount of processing.

Observations 1 and 2 suggest that given a defined constant, whose completion is a type 1 expression, we may pre-analyze the constant by assuming a prefix and dummy arguments and applying the operations of E to produce an ungrounded tree such as t_1 . The analysis is performed twice; once for an assumed \textcircled{T} prefix and once for an assumed \textcircled{F} prefix. The resulting trees are called truth-function analysis trees and are considered to be associated with \textcircled{T} and \textcircled{F} occurrences of the parent constant respectively.

Once the truth-function analysis trees (tfat's) associated with \textcircled{T} and \textcircled{F} occurrences of a constant are calculated, total unabbreviation of an arbitrary occurrence of that constant becomes an operation essentially the same as O_2 . Thus, suppose that some node n of the tree t contains $\delta c a_1 \dots a_n$ and that the tfat's for c have previously been calculated. If k is some leaf of t , ($n < \cdot_t k$) we simply substitute a_i for x_i in the tfat associated with a δ occurrence of c (where δ is either \textcircled{T} or \textcircled{F}) and append the result to k .

In the general case, unabbreviation is somewhat more complicated than might be assumed from the above discussion. Complexity enters in the form of implicit dependencies arising from the context of a defined constant coupled with the occurrence of quantified variables in the definition of the constant. As long as a constant contains no implicit quantified variables¹ we may proceed as above. However, once implicit quantified variables are present, unabbreviation should provide explicit representation of any dependencies uncovered by the unabbreviation operation.

In the following sub-section we present an algorithm for computing tfat's for defined constants from their definitions. Associated with each such tfat will be a dependency forest which explicitly represents the internal dependencies of the associated constant which result solely from the nature of the particular constant. These dependency forests provide the necessary mechanism for determining dependencies among implicit variables occurring in a constant's definition and variables in whose scope a particular occurrence of the constant appears.

The operator O_2 is extended to cover defined constants having associated tfat's. Its operation is essentially the same as before

¹ \bar{E} ! is an example of a constant which contains implicit quantified variables. Although such variables may be positive in the definition (i.e. represent universal quantification) they will be negative when \bar{E} ! is unabbreviated if the particular occurrence of \bar{E} ! being unabbreviated is in the scope of an odd number of occurrences of \textcircled{O} if prefix is \textcircled{T} or an even number of occurrences if prefix is \textcircled{F} . Thus, unabbreviation may uncover dependencies, since there is always the possibility that a negative variable will be within the scope of a positive variable.


except that dependencies arising from unabbreviation are explicitly represented in a dpf associated with the new tree $O_2(n,k)$ and as Skolem functions occurring in the contents of $O_2(n,k)$ and associated dpf.

In any system such as E , it is desirable to be able to take advantage of previously proven results. The operator O_{12} discussed in sub-section 3.4.2 provides E with a primitive capability for utilizing t_{fat}'s calculated for expressions known to be valid.

3.4.1 Calculation of truth-function analysis trees for defined constants

Basic method

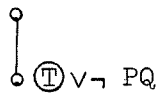
Suppose that the following are true:

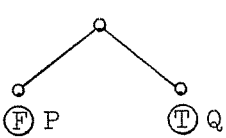
1. $c \longleftrightarrow \lambda x_1^1 \dots x_{n_1}^1 \bullet \dots \lambda x_1^k \dots x_{n_k}^k \bullet e$ is a definition with associated dependency forest F .
2. The first character of e is not λ .
3. t is the ungrounded tree  δ_e with associated dependency forest F (given in 1) and δ is either \textcircled{T} or \textcircled{F} .

Given t as just described and assuming that e contains no circularly defined constants, the procedure presented below produces a tree \hat{t} which possesses the following properties:

1. \hat{t} is an ungrounded tree (with associated dpf).
2. $\delta c x_1^1 \dots x_{n_1}^1 \dots x_{n_k}^k$ is true if and only if \hat{t} is satisfiable.
3. If b is the suffix of the content of any node of \hat{t} , then b is either an atom or an abbreviation whose first character is an undefined constant.

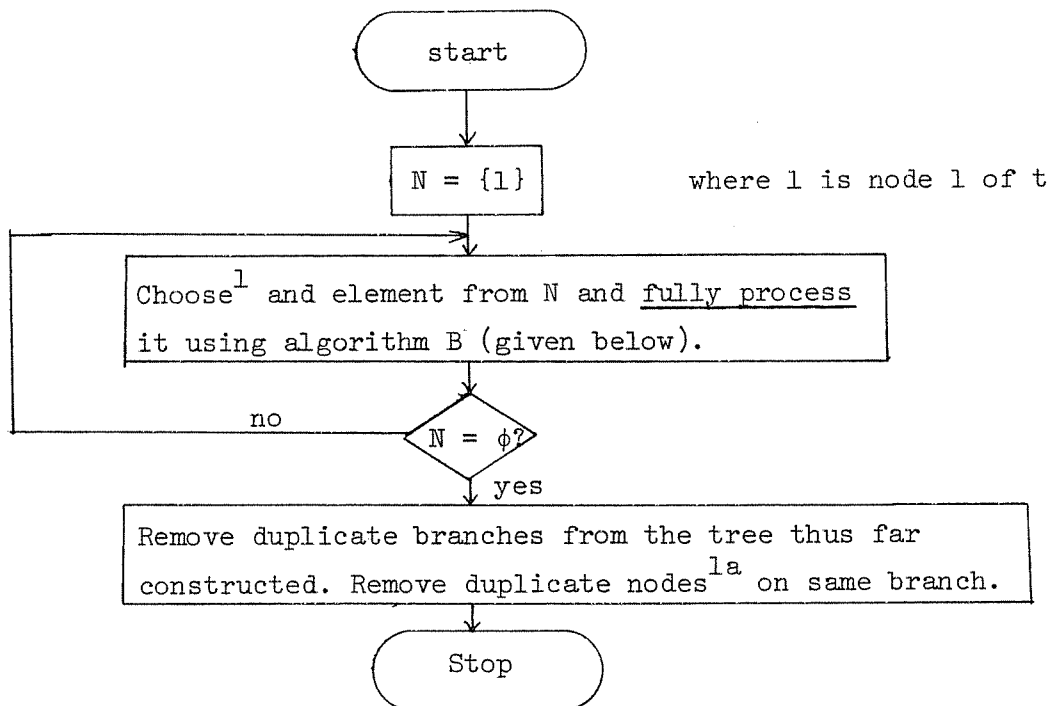
example $(d_4) \supset \leftrightarrow \lambda PQ. \vee \neg PQ$ $\{o \vee \neg PQ+\}^*$

t is the tree  with associated dpf $\{o \vee \neg PQ+\}$

\hat{t} is  with associated dpf $\{oP-, oQ+\}$

Algorithm A (basic calculation of tfat's)

The tree \hat{t} associated with $\delta c x_1^1 \dots x_{n_1}^1 \dots x_{n_k}^k$ is calculated as follows:



* Where oe is a tree having one node with content e .

¹We shall assume that some ordering for N is generated which allows exhaustive enumeration of its elements.

^{1a}Remove node furthest from the origin.

Algorithm B (fully processing a node)

Let $C_t(n) = \delta e$ where $n \neq \text{org}(t)$ and $n \in N \subset N_t$. The processing of node n depends on the nature of the first character of e . Let this character² be denoted by γ .

case 1 (γ is the character λ or a defined constant.)

B1.1 Apply modified³ λ -reduction (O_1) or unabbreviation (O_{11}) whichever is applicable

B1.2 Do not remove n from N

case 2 (γ is the character \oplus .)

B2.1 Apply $O_2(n,k)$ for each leaf k such that $n <_t k$.

B2.2 Remove n from t

B2.3 Remove n from N

B2.4 Add to N those nodes created in B2.1 (i.e. by applications of O_2).

case 3 ($\gamma \in V \cup S$ or γ is an undefined constant.)

B3.1 Remove n from N

²Any E_{L^*} -expression of the form $\oplus e$ or $\oplus e$ has the property that the first character of e must be \oplus , λ or an element of $V \cup S \cup C$. Thus, the given case analysis is exhaustive.

³Skolem functions are introduced by the λ -reduction operation. The sole purpose of introducing these functions is to expedite the determination of unifying substitutions. However, unifying substitutions are not sought during the calculation of truth-function analysis trees, thus making introduction of Skolem functions unnecessary at this stage. In addition, the manner in which dpf's for truth-function analysis trees will be used precludes any reasonable usage of Skolem functions which might be introduced at this point. Modified λ -reduction is λ -reduction without Skolem function introduction.

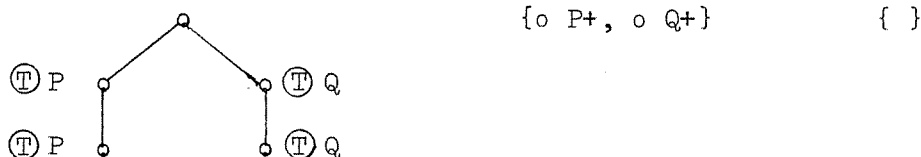
example calculation of the tfat associated with $\textcircled{T} \vee PQ$

case	truth-function analysis tree	dependency forest	N
		{o ¬ P-, o ¬ Q-}*	{1}
2		same	{2,3}
1 use 0_{11}		{o M+, o ¬ Q-}	{2,3}
1 use λ-red.		{o P+, o ¬ Q-}	{2,3}
2		{o P+, o ¬ Q-}	{4,5,3}

Nodes 4 and 5 are removed by step B3.1 (case 3). Further processing of 3 (which is identical to that of 2) yields

* Where oe represents a tree having one node with content e.

3



There are no duplicate branches, and duplicate node removal yields the final tree t given below:



which is associated with $\textcircled{T} \vee PQ$.

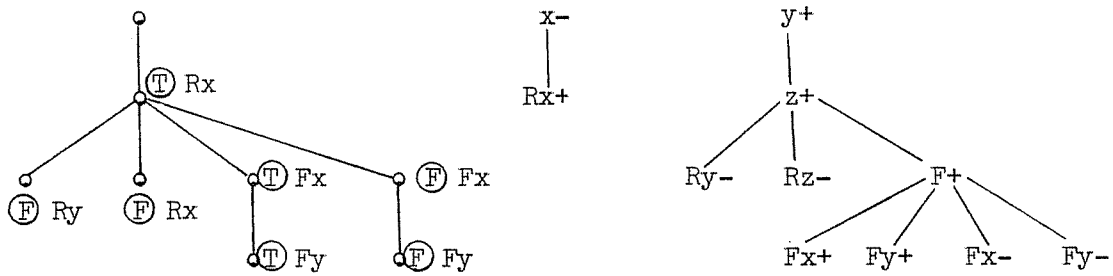
Controlling the depth of analysis afforded by a truth-function tree

In certain case, it may be desirable to limit the depth of truth-functional analysis afforded by algorithm A. This can be accomplished by treating specified defined constants⁴ as if they were undefined. Thus, defined constants are partitioned into two sets: those to be analyzed and those not to be analyzed. The former set are processed by case 1 of algorithm B, the latter by case 3.

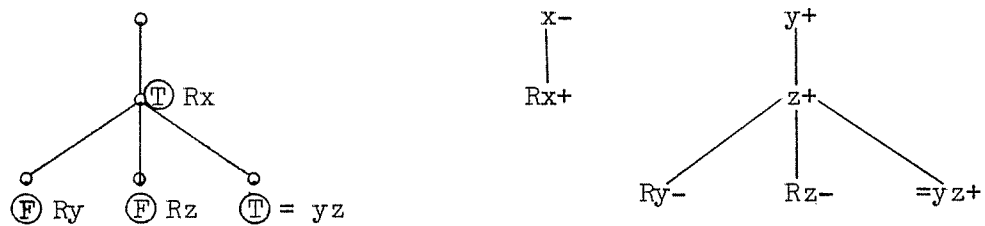
example - limited truth-function tree

If algorithm A is applied to the defined constant $\bar{E}!$, which is introduced on page 103, the following tree associated with $\textcircled{T} \bar{E}!R$ is obtained:

⁴We might even wish to treat specified defined constant occurrences as undefined. That is, rather than consider all occurrences of a specified defined constant as defined, consider only certain occurrences as defined.



If, on the other hand, = is treated as if it were undefined, we obtain:



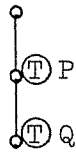
We note that the level of analysis afforded by the above tree and its associated dpf is sufficient to deal with the theorem $(a \neq b \ \& \ (\exists!Px \ \& \ Pa)) \supset \neg Pb$.

Examples of truth-function trees for selected defined constants

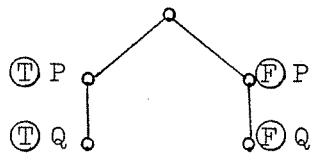
We have already considered the truth-function analysis trees associated with $(\mathbb{T})\forall$, $(\mathbb{T})\supset$, $(\mathbb{T})\neg$ and $(\mathbb{T})\bar{\exists}$!. The truth-function analysis trees associated with the remainder of the constants introduced on page 103 are given below. All implicit constants are analyzed.

⊤ occurrence

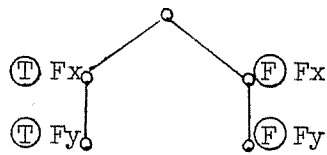
⊤ & PQ



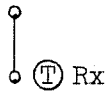
⊤ ≡ PQ



⊤ = xy

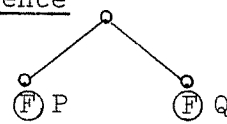


⊤ \bar{E} R



⊥ occurrence

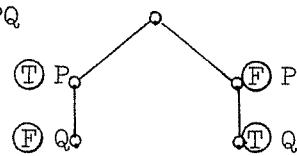
⊥ & PQ



dpf

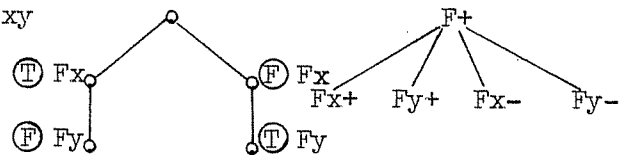
{oP+, oQ+}

⊥ ≡ PQ



{oP+, oQ+, oP-, oQ-}

⊥ = xy



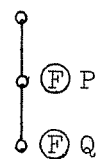
⊥ \bar{E} R



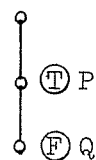
⊥ \neg P



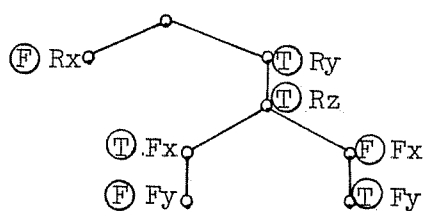
⊥ \vee PQ



⊥ \supset PQ



⊥ \bar{E} !R



Utilizing truth-function analysis trees in E (the extended operator O_2)

At the time inference operators were first introduced, it was observed that they provided a means of investigating the ramifications of assuming a given spe to be satisfiable. The inference operator O_2 , for example, covered spes of the forms $\mathbb{T} \circledast e_1 e_2$ and $\mathbb{F} \circledast e_1 e_2$. Associated with each form was an ungrounded tree⁵ which represented the possible truth value assignments which could account for the presumed satisfiability of the particular spe. The operator O_2 simply appended the appropriate tree to some specified leaf of the tree containing the original spe subject to the condition that the node containing the spe was an ancestor of the leaf.

The truth-function analysis trees, which we have been considering, allow exactly the same sort of analysis as do the trees associated with $\mathbb{T} \circledast e_1 e_2$ and $\mathbb{F} \circledast e_1 e_2$. It is thus reasonable to extend the operator O_2 to apply to any spe of the form $\delta c a_1 \dots a_n$ where δ is either \mathbb{T} or \mathbb{F} and c is a defined constant which has associated truth-function analysis trees for $\mathbb{T} c x_1 \dots x_n$ and $\mathbb{F} c x_1 \dots x_n$. If we now agree that $\mathbb{T} \circledast e_1 e_2$ and $\mathbb{F} \circledast e_1 e_2$ have associated truth-function analysis trees, then O_2 may be extended to cover constants such as c as well as \circledast .

Algorithm C (calculation of $O_2(n,k)$ for the extended operator O_2)

Let c be a defined constant and suppose that $\mathbb{T} c x_1 \dots x_n$ and $\mathbb{F} c x_1, \dots, x_n$ have been processed by algorithm A (i.e. they

⁵This was called the appended tree (see page 62).

have associated truth-function analysis trees). Further suppose that $C_t(n) = \delta ca_1 \dots a_n$ where δ is either \textcircled{T} and \textcircled{F} and that k is a leaf of $t(n \prec_t k)$. Let s be the tfat associated with $\delta cx_1 \dots x_n$ and $\text{dpf}(s)$ its corresponding dpf.

case 1 ($\zeta(n)$ is defined and is the leaf m of $\text{dpf}(t)$)

- Cl.1 preprocessing If $\text{dpf}(t)$ and $\text{dpf}(s)$ conflict, revise $\text{dpf}(s)$ (and s) relative to $\text{dpf}(t)$.
- Cl.2 truth-function tree calculation Calculate $t_1 = s\sigma$ where $\sigma = \{\dots, \langle a_i, z_i \rangle, \dots\}$; z_i is x_i if x_i was not revised in step Cl.1, otherwise z_i is the variable substituted for x_i . Remove all occurrences of $\forall v (v \in V)$ which are not within the scope of some λ or constant. Let the result be t_2 . (Steps Cl.1 and Cl.2 produce a tfat which is specialized to the particular expression $\delta ca_1 \dots a_n$ and which contains no variables which conflict with those already occurring in $\text{dpf}(t)$.)
- Cl.3 dependency forest calculation Let F be the result of expanding each leaf of $\text{dpf}(s)\sigma$ using algorithm F section 2.1.1 (node expansion). Algebraically replace m (leaf of $\text{dpf}(t)$) with F using algorithm E of section 2.1.1 and let the result be G .
- Cl.4 Skolem function introduction Introduce Skolem functions into t_2 (calculated in step Cl.2) and G (calculated in step Cl.3) by replacing all negative variable occurrences

with Skolem functions as prescribed in algorithm D section 2.1.1 step D3. Let r and H be the respective results.

The tree $O_2(n,k)$ is obtained by appending r to the leaf k where $n <^*_t k$. Its associated dpf is H .

case 2 ($\zeta(n)$ is undefined)

For this to be the case, O_2 case 1 above must have previously been applied either to n or to some node of $\zeta^{-1}(m)$ where m was the leaf of $\text{dpf}(t)$ corresponding⁶ to the node n at the time of that case 1 application of O_2 .

Let r be the tree calculated in step C1.4.

C2.1 Append r to the leaf k where $n <^*_t k$. (The $\text{dpf}(t)$ is associated with $O_2(n,k)$.)

case 3 (c is the character (\mathbb{D}))

C3.1 Calculate $O_2(n,k)$ as specified for old version of O_2 (see page 62).

The tfat and dpf associated with c possess several characteristics which should be discussed. The tfat explicitly represents structure which is determined solely by the particular distribution

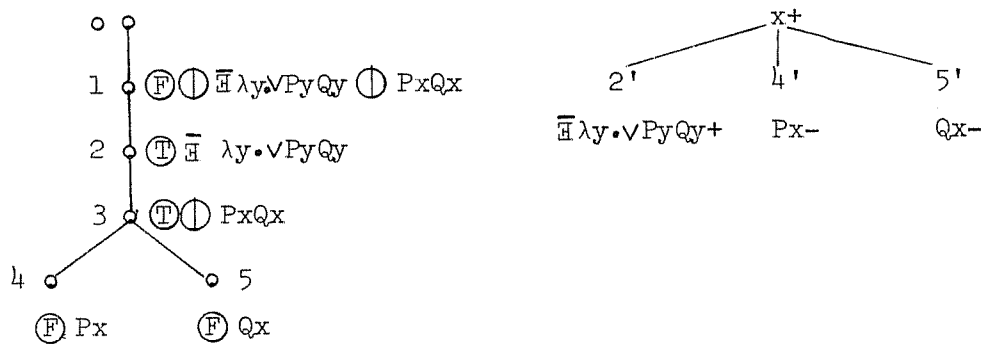
⁶If case 1 is applicable, then there exists a leaf m of $\text{dpf}(t)$ such that $\zeta(n) = m$. Furthermore, $\zeta^{-1}(m)$ is the set of all nodes of t which correspond under ζ to m . In applying O_2 (case 1) to n we destroy this correspondence since in general step C1.3 replaces the leaf m with a forest whose leaves correspond to pseudo atoms of the content of node n . The reference here (in case 2) pertains to the set of nodes corresponding to the leaf m prior to the application of O_2 under case 1, i.e. while $\zeta(n)$ was still defined.

of $\textcircled{1}$ in the primitive definition of c . The quantificational structure, on the other hand, is represented in the dpf of the tfat. If O_2 is applied to a node n occurring in the interior of some tree t , the tfat associated with c may correctly be appended to any leaf $k \cdot > n$. The first application of O_2 to n causes calculation of a dpf (Cl.3 and Cl.4) which is associated with $O_2(n,k)$. Since this dpf explicitly represents all dependencies resulting from the context of the pseudo atom containing c in the intial spe, and since these in no way depend on how many times (after the first) $O_2(n,k)$ is applied, it is unnecessary to recompute the dpf under case 2. In fact, had we wished to do so, the tree r computed in Cl.4 could have been appended to all leaves dominated by n . This, however, is undesirable for the same reason that blind total unabbreviation is undesirable.

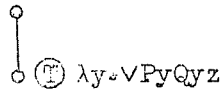
We now present two examples of extended O_2 usage.

example 1 (In this example, we shall use the truth-function analysis tree for $\textcircled{\text{T}} \bar{\text{H}} R$. This is given on page 103.)

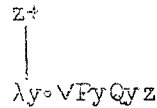
Suppose that at some stage of processing, we have the following tree and dependency forest



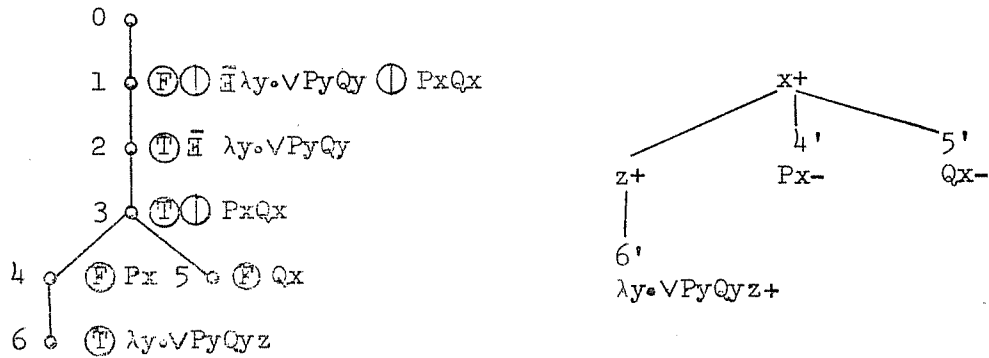
and wish to calculate $O_2(2,4)$ and its dependency forest. Case 1 applies since 2 corresponds to the indicated node in the above dpf. Furthermore, x_1 is R, and a_1 is $\lambda y \cdot \forall PyQy$. The tree r computed in step C1.4 is then



and F is the dpf

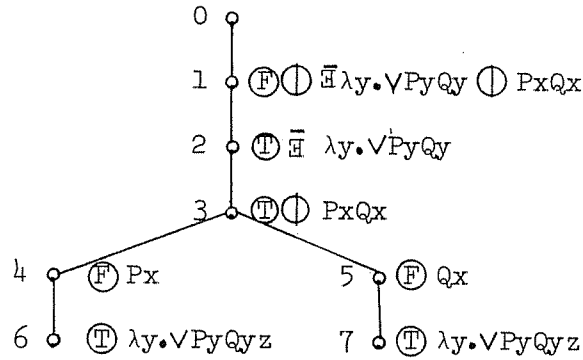


Application of $O_2(2,4)$ thus produces



Note that node 2 no longer corresponds to a leaf on the dependency forest.

If we now wish to apply $O_2(2,5)$, case 2 applies. We thus obtain

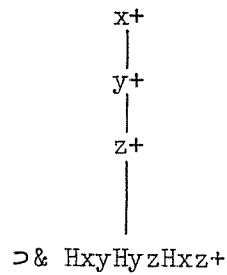


The dependency forest is unchanged. We note in particular that $\zeta(6') = \{6,7\}$ since both 6 and 7 correspond to node 6' of the dependency forest.

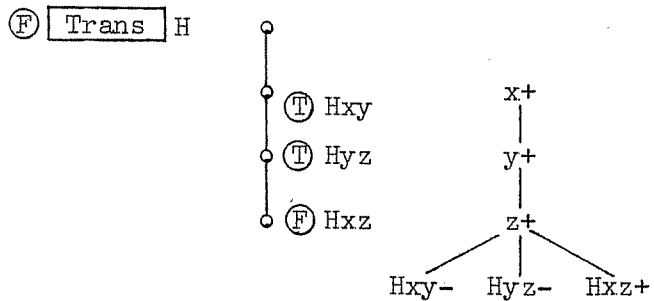
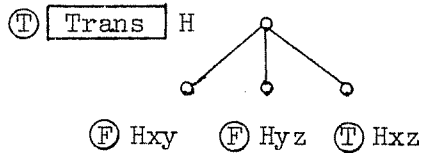
example 2 proof of transitivity of = using extended O_2

Let the following definition be added to those given on page .

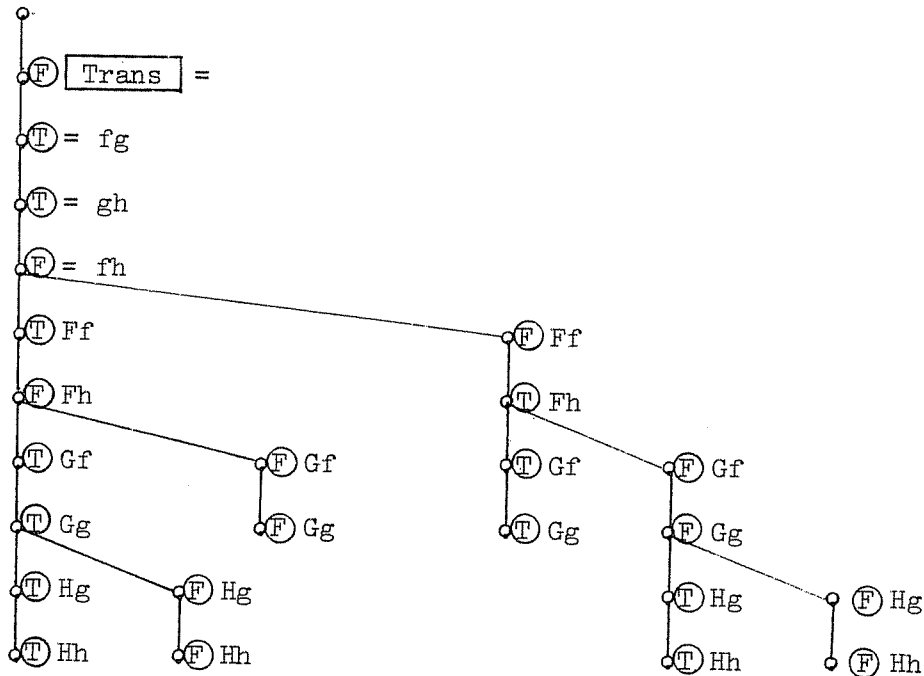
d_9 Trans $\longleftrightarrow \lambda H. \supset \& Hxy Hyz Hxz$



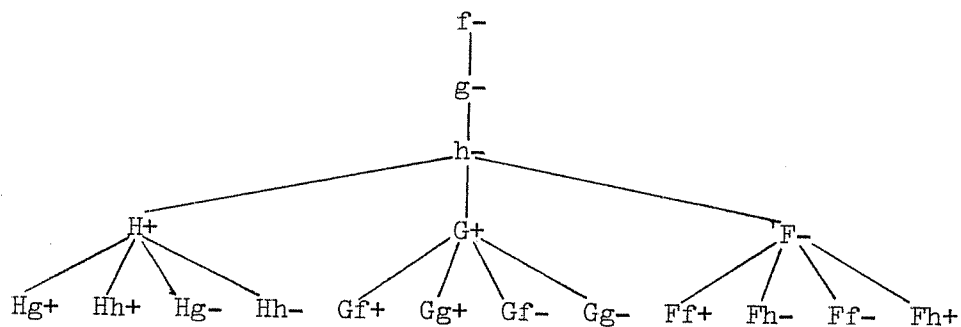
Application of algorithm A produces the truth-function analysis trees and associated dpf



Now suppose the initial tree θ is $1 \circ \textcircled{F} \text{Trans} =$ with associated dpf $\{ \circ \text{Trans} = - \}$. Application of O_2 to Trans followed by five applications of O_2 to various occurrences of "=" yields⁷



The associated dpf is



⁷References to O_2 now refer to the extended version as computed by algorithm C.

Applying tree substitution with $\sigma = \{\langle F, G \rangle, \langle F, H \rangle\}$, and multiple closed branch removal produces the empty tree ϕ . Thus $\theta \xrightarrow{L_1} \phi$ and $\boxed{\text{Trans}} = \text{is}$ therefore provable. i.e. $T_1 \boxed{\text{Trans}} = \text{is}$ a theorem of E .

Modification of the truth-function analysis tree algorithm (algorithm A)

Algorithm A reduces primitively defined constants to their primitive definienda by alternately applying unabbreviation (O_{11}) and λ -reduction (O_1). This can involve a considerable amount of processing if the constant in question has a definiendum which implicitly contains deeply nested⁸ defined constants. Fortunately, if a constant appearing in a definiendum has already been processed by algorithm A and consequently has an associated truth-function analysis tree, it is possible to avoid reprocessing the constant by utilizing the pre-calculated truth-function analysis tree. This modification of algorithm A is obtained simply by altering the method used to fully process defined constants as described in algorithm B. In particular, we define algorithm D and E as follows:

Algorithm D (modified algorithm B)

case 1 (γ is the character λ or a defined constant)

D1.1 same as B1.1 if γ is λ or a defined constant without an associated tfat.

⁸The defined constant "=" is an example. It implicitly contains the defined constant \vee which occurs two levels down.

D1.2 Let n be the node whose content contains c and such that $m = \zeta(n)$. Calculate $O_2(1,k)$ for each l and k such that $l \in \zeta^{-1}(m)$ and $l <_t k$, (use algorithm C) omitting step Cl. +⁹ if case 1 applies and using t_2 instead of r .

D1.2.1 Remove n from t

D1.2.2 Remove n from N

D1.2.3 Add to N those nodes created in D1.2 by applying O_2 (extended).

case 2 (γ is the character \oplus) same as case 2 of algorithm B.

case 3 ($\gamma \in V \cup S$ or γ is an undefined constant) same as case 3 of B.

Algorithm E (tfat calculation utilizing tfats)

Same as algorithm A except use algorithm D in place of algorithm B.

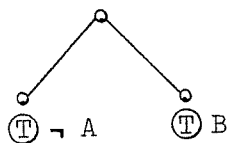
Algorithm E produces the same results as algorithm A but with less processing. Depth control may be included if desired (cf. page 119).

example -- algorithm E applied to $\oplus \triangleright AB$




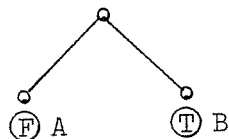
⁹ That is, apply O_2 but suppress introduction of new Skolem functions into the resulting trees.

The tree associated with $\textcircled{T} \vee PQ$ is calculated on page 118. Substituting $\neg A$ for P and B for Q and completing the remaining substeps D1.2 - D1.2.3 yields.



with associated dpf $\{o\neg A+, oB+\}$ which is derived from the original dpf $\{o \vee \neg AB\}$.

Finally, taking the tree associated with $\textcircled{T} \neg R$ to be  and its dpf to be $\{o R-\}$ we obtain



$\{oA-, oB+\}$

Utilization of the precalculated trees for $\textcircled{T} \vee$ and $\textcircled{T} \neg$ results in a relatively small amount of processing compared to the amount of processing required by the unmodified procedure. This becomes clear when we note that in order to calculate $\textcircled{T} \supset$ using the unmodified procedure, it is necessary to calculate $\textcircled{T} \vee$ as a subprocedure. This calculation by itself (see page 118) is longer than the entire calculation by algorithm E of $\textcircled{T} \supset$. Since the examples given are simple, one would expect even greater differences to exist for more complex cases.

3.4.2 Utilizing theorems in E.

Suppose that e is a provable expression with initial spe $\textcircled{F} \bar{e}$. By definition, \bar{e} must always be true. It follows that if

t is any ungrounded tree over E_{L*} , then adding a node containing $\textcircled{T} \bar{e}$ to any branch of t can not affect the satisfiability of t. That is, t will be satisfiable or unsatisfiable whether the node is added or not.

The ability to make such an addition, while not adding to the logical power of E , can significantly reduce the amount of work necessary to close a given tree if the tree contains an expression $\textcircled{F} d$ where d and \bar{e} are unifiable.

example Suppose we have proved the following theorems:

$$T_2 \quad \boxed{\text{Sym}} = \quad (\text{where } \boxed{\text{Sym}} \leftrightarrow \lambda H. \supset HxyHyx \quad \begin{array}{c} x+ \\ | \\ y+ \\ | \\ \supset HxyHyx+ \end{array})$$

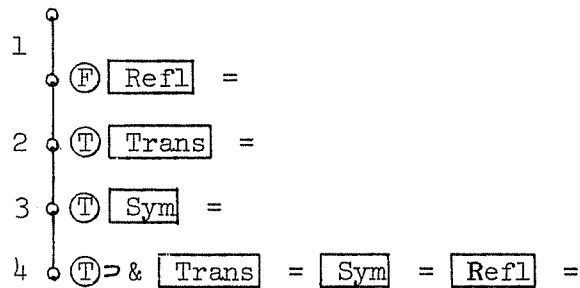
$$T_3 \quad \supset \& \boxed{\text{Trans}} H \boxed{\text{Sym}} H \boxed{\text{Ref1}} H \quad \begin{array}{c} H+ \\ | \\ \supset \& \boxed{\text{Trans}} H \boxed{\text{Sym}} H \boxed{\text{Ref1}} H+ \end{array}$$

$$\text{where}^1 \quad \boxed{\text{Ref1}} \leftrightarrow \lambda H. \underbrace{\supset \& = xy \vee \bar{E} \lambda w. Hxw \bar{E} \lambda z Hzx Hxy}_{\text{Def}} \quad \begin{array}{c} x+ \\ | \\ y+ \\ | \\ \text{Def}+ \end{array}$$

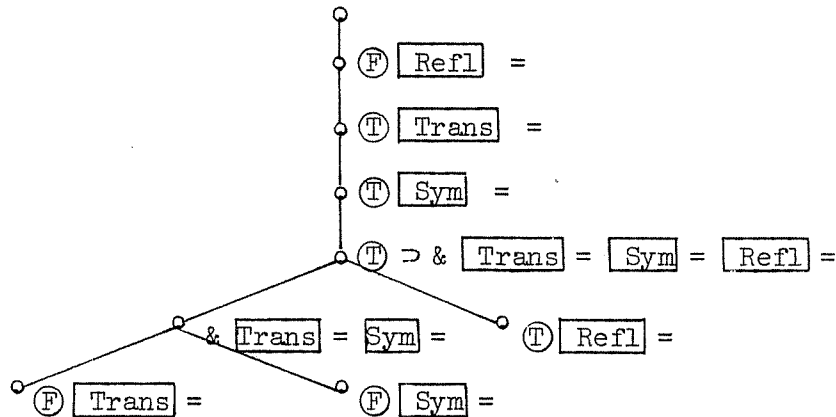
T_3 states that if H is any transitive and symmetric relation, then H is also reflexive (our definition of reflexive).

¹This is essentially the same as the first order predicate calculus formula $\text{Ref1} \equiv \forall x \forall y ((x=y) \& (\exists w Hxw \vee \exists z Hzx) \supset Hxy)$. To aid in parsing expressions containing defined constants such as Ref1, we shall enclose the defined constant in a rectangle.

We wish to prove the simple corollary² $\boxed{\text{Refl}} = .$ We thus take $\{1 \circ \textcircled{F} \boxed{\text{Refl}} = \}$ as the initial tree Θ and append three nodes, obtaining the following:



The last node appended is recognizable as an instance of the general expression previously given. Applying O_2 to \supset and to $\&$ yields

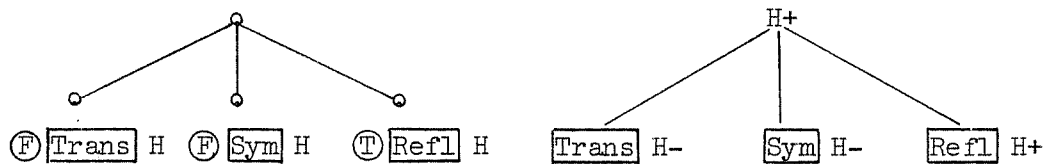


Since each branch has conjugate nodes, we can obtain a proof of $\boxed{\text{Refl}} = .$ by applying multiple closed branch removal.

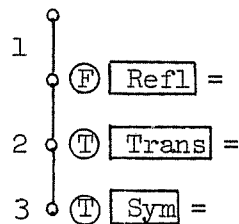
We note that in the above example it was necessary to truth-functionally analyze the expression $\textcircled{T} \supset \& \boxed{\text{Trans}} = \boxed{\text{Sym}} = \boxed{\text{Refl}} = .$

²This can, of course, be proved directly, however, we wish to illustrate the use of previously proved expressions.

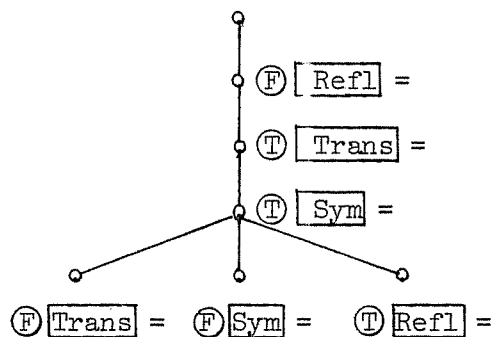
Now, it is possible to calculate a truth-function analysis tree for the expression $\textcircled{T} \supset \text{Trans H Sym H Refl H}$ utilizing either algorithm A or E. Had we done this, we would have obtained:



Here we have assumed that all defined constants, with the exception of $\&$ and \supset , have been treated as undefined³. If we substitute = for H in the above tree and associated dpf and append the result to leaf 3 of the tree



we obtain



³That is depth control has been applied as discussed in section 3.4.1.

Thus, it is more desirable to append a partial truth-function analysis of the theorem than to append the unanalyzed theorem as was done in the first case.

We may systematize the above operations by adding the operator O_{12} to the existing inference operators of E .

O_{12} - theorem utilization

Let t be an ungrounded tree which has been produced by application of the inference operators of E and let k be a leaf of t . Furthermore, let s be a truth-function analysis tree corresponding to $\textcircled{\mathbb{T}} e^4$ which has no variables in common with variables of t^5 . Finally, let σ be any substitution for positive variables of s .

The ungrounded tree which results from applying O_{12} to t , σ and s is obtained by appending $s\sigma$ to k and removing any branches which contain conjugate nodes. Its associated dependency forest is composed of the trees which comprise $\text{dpf}(t)$ and $\text{dpf}(s)$. That is, it is the union of the two forests considered as sets of trees.

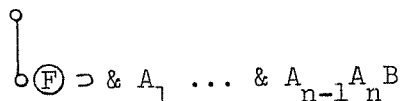
axiom utilization in E

The operator O_{12} may be applied with axiom as well as theorem arguments. Suppose we wish to demonstrate that a closed type 1

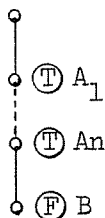
⁴ $\textcircled{\mathbb{P}} e^-$ is the initial spe corresponding to the provable expression e .

⁵This can be arranged either by revising conflicting variables as a preprocessing step or as part of O_{12} .

expression B is valid⁶ in any domain in which the axioms A_1, \dots, A_n are valid. In the system E , this is equivalent to showing that $(F) \supset \& A_1 \dots \& A_{n-1} A_n B$ is unsatisfiable⁷. If we define θ to be the initial tree



we can obtain the following tree by applying the operators of E :



Thus, if we wish to prove that B is valid in the context of the axioms A_1, \dots, A_n , it is only necessary to append the trees

$(T) A_i$ $i = 1, \dots, n$. As in the case of theorems, we may calculate the t for any axiom A by applying algorithm E of section 3.4.1 to $(T)A$. In particular, if we can obtain the empty tree as a linear extension of the initial tree $(F) B$ using O_{12} with the axioms A_1, \dots, A_n (as well as using other operators of E) then $\supset \& A_1 \dots \& A_{n-1} A_n B$ is a theorem of E .

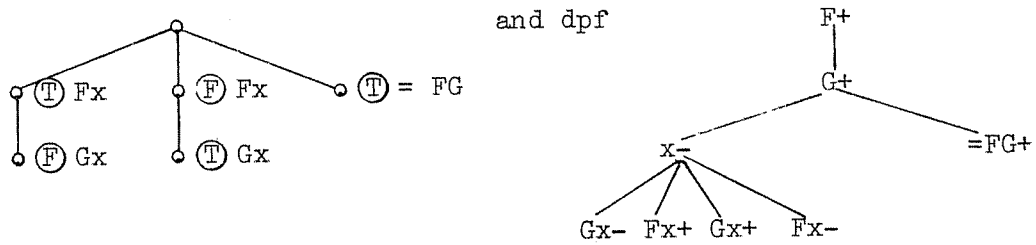
⁶We mean, of course, that B is true for all possible value assignment in each domain in which A_1, \dots, A_n are true (simultaneously) for all value assignments.

⁷In infix notation this would be $(A_1 \& (A_2 \& \dots A_n) \dots) \supset B$.

example Let us define the axiom of extensionality as follows:

$$\text{Ext} \leftrightarrow \supset \equiv FxGx = FG^8 \quad \{o \supset \equiv FxGx = FG+\}$$

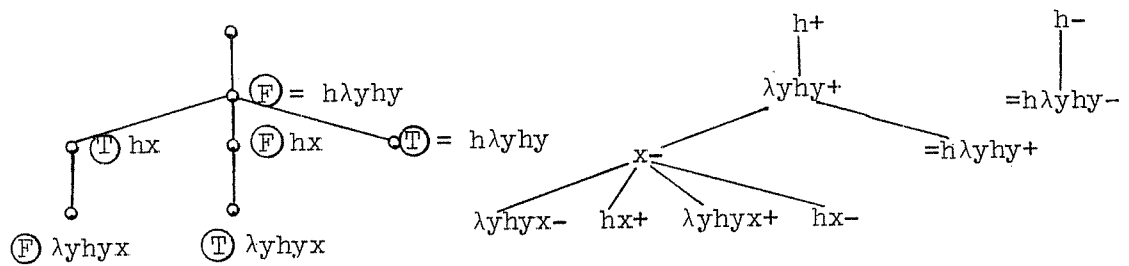
Algorithm E, applied to a \textcircled{T} occurrence of Ext, produces⁹



We shall now outline a proof of the E equivalent of $\text{Ext} \supset \forall P(P = \lambda yPy)$.
The initial tree and associated dpf are



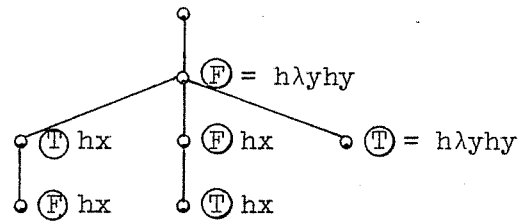
where h is a Skolem function. If O_{12} is applied with Ext (above) and $\sigma = \{ \langle h, F \rangle, \langle \lambda yhy, G \rangle \}$ we obtain:



⁸Semantically, this is the same as $\forall F(\forall G((\forall x(Fx \equiv Gx)) \supset F=G))$.

⁹The constant "=" is treated as undefined for the application of algorithm E as discussed under depth control in section 3.4.1.

Two applications of λ -reduction yields:



Multiple closed branch removal produces ϕ thus demonstrating that

$\Theta \xrightarrow{L_1} \phi$. This completes the proof.

CHAPTER 3 DEFINED RULES OF INFERENCE

1.0 Introduction

In this chapter we wish to introduce the notion of a defined rule of inference and demonstrate how specific instances of this notion may be used to simulate selected proof procedures for the first order predicate calculus. Since the actual form of a defined rule of inference is dependent upon syntactical details¹ which are peripheral to the main considerations of this chapter, we shall confine our treatment to a general discussion of the setting and properties of these rules.

A defined rule of inference is basically a generalization of the inference operators presented in Chapter 2 which combines the operation of one or more inference operators in order to produce system trees possessing prescribed properties. Specifically, a defined rule of inference (DRI) is a well defined procedure which accepts as input a set of ungrounded trees satisfying some prescribed condition² and solely by application of E -inference operators to these trees and their descendants, produces a sequence of ungrounded trees. If this sequence is finite, we then distinguish some subsequence as the output of the DRI. It follows that an output tree is satisfiable if the input trees from which it descends are simultaneously

¹These details are covered in appendix A.

²The E -inference operators given in Chapter 2 are examples of DRIs. Corresponding to each operator is a set of conditions which must be satisfied if the operator is to be applicable to a given input tree. For example, $\textcircled{T} \textcircled{\perp}$ with given parameters n and k (a specific instance of $O_2(n,k)$) applies to the tree t if the following properties hold:

satisfiable. In these respects, a DRI is similar to an \bar{E} -inference operator since an \bar{E} -inference operator accepts a set of ungrounded trees as inputs (either a singleton or pair), produces a set of ungrounded trees (a singleton) as output and preserve satisfiability in the above sense. A DRI differs in general from an \bar{E} -inference operator in that it may produce no output set (i.e. it may not terminate) or an output set consisting of more than one ungrounded tree. The DRI fpn (fully process node) illustrates some of these ideas.

The defined rule of inference fpn

If t is any ungrounded tree which satisfies the property of containing a node n whose major connective γ has an associated t_{γ} , we shall say that O_2 is applicable to t at n . Let $K(n) = \{k: n \prec_t k \text{ and } k \text{ is a leaf}\}$. If O_2 is applicable to t at n , then for any $k \in K(n)$, $O_2(n,k)$ is the result of applying O_2 to n for the leaf k and is the ungrounded tree obtained by appending t_{γ} to the leaf k . Furthermore, if t is satisfiable, then $O_2(n,k)$ is satisfiable. Fully processing n produces the tree $\text{fpn}_1(n)$ which is the result of applying O_2 to n for each $k \in K(n)$. This tree has the property that it is satisfiable if and only if t is satisfiable. It is an intermediate step in the production of $\text{fpn}(n)$.

²(continued)

1. $n \prec_t k$

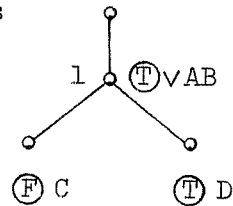
2. k is a leaf

3. $C_t(n) = \textcircled{\text{T}} \textcircled{\text{O}} e_1 e_2$ where e_1 and e_2 are type 1 expressions.

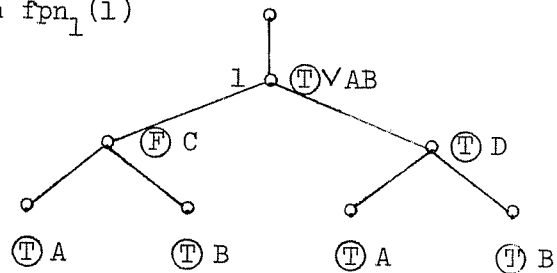
We note in particular that properties 1, 2 and 3 are decidable. In general, we require that the prescribed input properties, which constitute an applicability test for the given DRI, are decidable over the class of input trees.

example:

If t is



then $f_{pn_1}(1)$

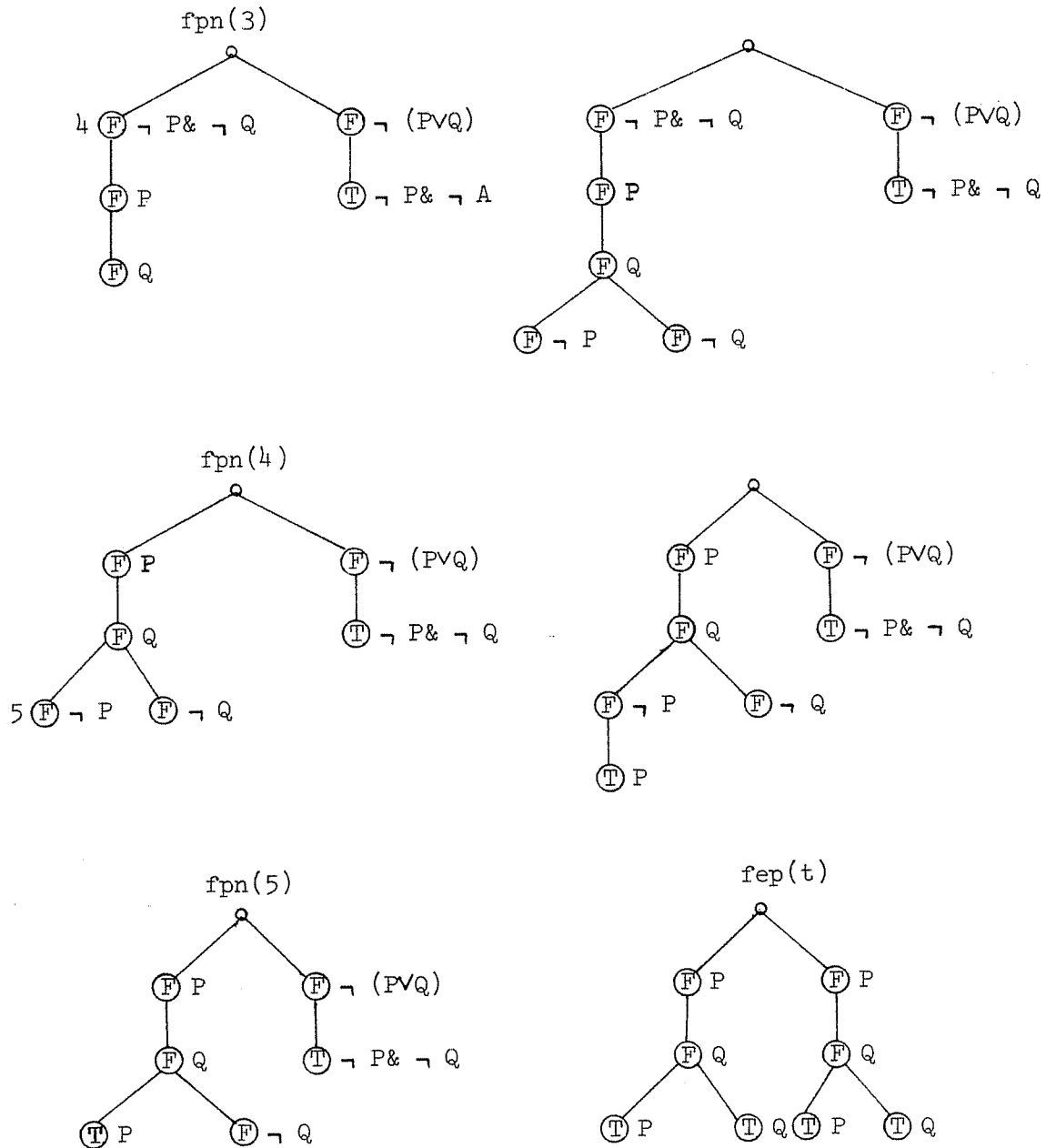


The defined rule of inference f_{pn_1} exhibits the general characteristics of a DRI. It accepts a set of trees satisfying some property. (t satisfies the given property.) It produces a sequence of trees (t and a new tree for each application of O_2 to n for $k \in K(n)$). It clearly terminates and upon termination then produces the last tree of the sequence as output. It preserves satisfiability. Furthermore, the output tree has the property that it is satisfiable if and only if the input tree was satisfiable. This latter property continues to hold even if n is removed. Thus, there is no need to retain r . We may thus specify another DRI f_{pn} which has the same properties as f_{pn_1} and which produces an output which does not contain n .

f_{pn} 1. Apply f_{pn_1} to n .

2. Apply node removal to the image of n in the result.

In the above example $f_{pn}(1)$ is



Continued processing of the remaining tree branches produces the output tree fep(t).

It follows by a simple inductive argument that fep will terminate for any input system tree containing only primitively definable defined constants. Consequently, fep will always produce an output tree under these conditions. In general, if we place no restrictions on the input trees (other than that they be ungrounded trees, we can not even guarantee that fep will terminate. Even if the above input conditions are satisfied, we can still only conclude that fep(t) produces an output which is mutually satisfiable with the input and generally provides a deeper truth-functional analysis than the input. However, it is possible to restrict the class of input trees by strengthening the input applicability test, in such a way as to be able to guarantee that the output tree possess some rather useful properties.

Suppose that t_f's have been computed for \neg , \vee , $\&$, \supset , \equiv , and that P is the set of all type 1 expression of the form $\forall v_1 \dots \forall v_n R(v_1, \dots, v_n)$ where $v_i \in V$ has type 1 and $R(v_1, \dots, v_n)$ is a type 1 expression involving only the variables v_1, \dots, v_n and the defined constants $\neg, \vee, \&, \supset, \equiv$. P is the E -equivalent of the familiar propositional calculus^{3.5}. If we limit initial expressions to elements of P , then all system trees will contain expressions involving only $\neg, \vee, \&, \supset, \equiv$ and the Skolem functions f_1, \dots, f_n which are introduced for the universally quantified variables v_1, \dots, v_n . If we require that input trees for fep be chosen from un-

^{3.5} Thus, in particular, the propositional formula $P \vee Q$ would have the E -counterpart $\forall v_1 \forall v_2 \vee v_1 v_2$. Quantification is necessary in the latter expression since arbitrary formulas may be substituted for the propositional variables P and Q in the former formula.

grounded trees of this class, we observe the following:

1. The output tree represents⁴ a disjunctive normal form for $R(v_1, \dots, v_n)$.
2. The suffix of the content of every node of the output tree other than its origin, is an atom.

Since the output of fep is satisfiable if and only if the input is satisfiable and since a branch containing only atoms is unsatisfiable if and only if it contains conjugate spes, property 2 (above) guarantees that the following simple modification of fep is a decision procedure for P .

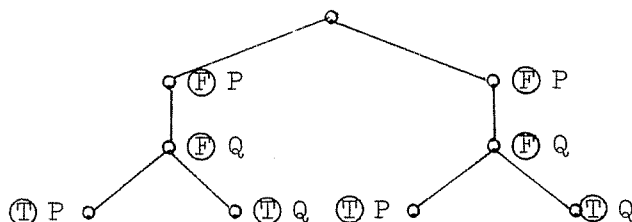
1. Apply fep to the input tree.
2. Apply multiple closed branch removal to the result of step 1.

If we denote the sequence of operations carried out in steps 1-3 by dpp (decision procedure propositional), we observe that dpp is a terminating⁵ DRI. Furthermore, if dpp is restricted to initial trees for $e \in P$, then e is a theorem of P if and only if $dpp(\theta) = \phi$.

⁴The DNF for $R(v_1, \dots, v_n)$ is $(L_1^1 \& \dots \& L_1^1) \vee \dots \vee (L_1^k \& \dots \& L_n^k)$. The literal L_j^i is obtained as follows. If n is the i th node of the j th branch and $C(n) = \delta e$ then $L_j^i = \begin{cases} e & \text{if } \delta = \textcircled{T} \\ \neg e & \text{if } \delta = \textcircled{F} \end{cases}$

⁵Excepting the pathological case where the input tree contains circularly defined constants.

example If we take θ to be $\circlearrowleft (P) \neg (PVQ) \equiv \neg P \& \neg Q$ then step 1 produces the tree



and step 2 produces ϕ . Thus $\text{dpp}(\theta) = \phi$ and $\neg (PVQ) \equiv \neg P \& \neg Q$ is a theorem of the propositional calculus.

Thus far we have considered only DRIs which take one tree as input and produce one tree as output. The following DRIs (multiple append and multiple proper forest copy) accept more than one tree as input and produce one or more trees as output.

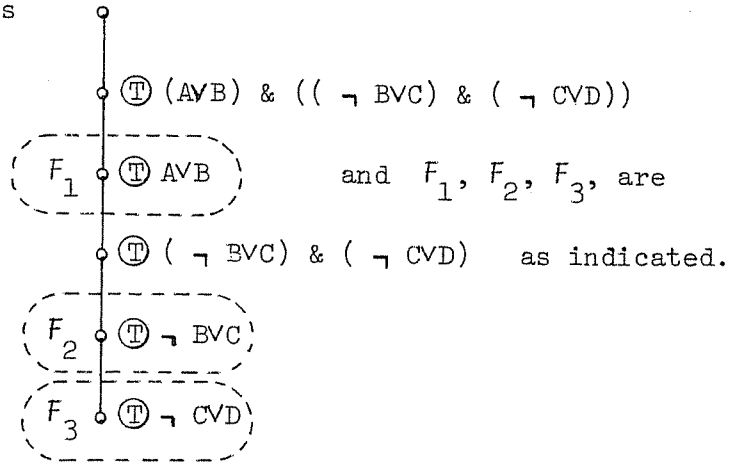
multiple proper forest copy (one in - many out)

If F_1, \dots, F_n are proper subforests in T then $\text{mpfc}(F_1, \dots, F_n) = \{O_7(F_1), \dots, O_7(F_n)\}$. We note that mpfc takes as input a single input tree. (This is implicit in the notation $\text{mpfc}(F_1, \dots, F_n)$.)

multiple append (many in - one out)

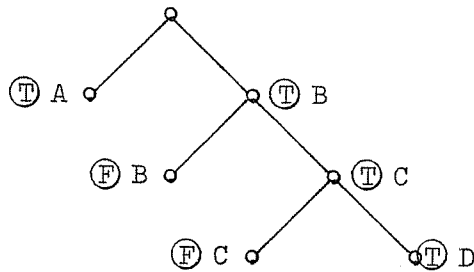
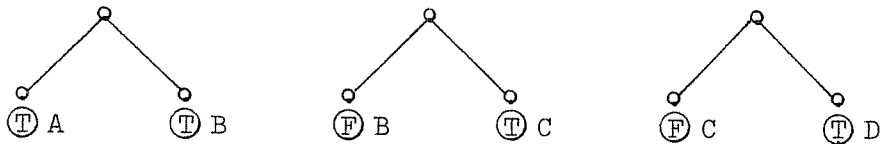
If t_1, \dots, t_n are ungrounded trees, l_1 is a leaf of t_1 , l_2 is a leaf of $O_9(t_1, l_1), \dots$, and l_n is a leaf of $O_9(t_{n-1}, l_{n-1})$, then $\text{ma}(t_1, \dots, t_n) = O_9(t_n, l_n)$.

example Suppose T is

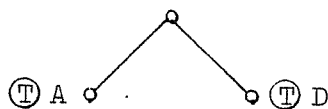


$$\text{mpfc}(F_1, F_2, F_3) = \left\{ \begin{array}{c} \circ \\ | \\ \text{Ⓣ AVB} \end{array}, \begin{array}{c} \circ \\ | \\ \text{Ⓣ } \neg BVC \end{array}, \begin{array}{c} \circ \\ | \\ \text{Ⓣ } \neg CVD \end{array} \right\}$$

If t_1, t_2, t_3 are the trees then $\text{ma}(t_1, t_2, t_3)$, for the images of the indicated leaves, is the tree



Applications of multiple closed branch removal and node removal produce the tree



The DRI test illustrated the fact that the input applicability test effects the properties of output trees. In certain cases, the severity of the input applicability test not only effects properties of the output trees, but also determines whether or not the DRI terminates for all inputs satisfying the test. The following DRI is an example of a defined rule of inference which terminates for certain inputs and not for others.

The DRI bfs (brute force strategy)

Let e be any formula of the first order calculus and $\Theta = \textcircled{P} M(x_1, \dots, x_n)$ its initial tree. (The variables x_1, \dots, x_n are distinct positive variables occurring in the dpf for Θ . They correspond to the existentially quantified variables occurring in e .⁶) Let $\sigma_i = \{\dots, \langle x_{in+j}, x_j \rangle, \dots\}$ $i > 1, 1 \leq j \leq n$ where $k \neq \ell$ implies $x_k \neq x_\ell$. If we define M_i to be the i^{th} variant $M(x_1, \dots, x_n)\sigma_i$, $S_1 = \neg M(x_1, \dots, x_n)$ and $S_i = S_{i-1} \ \& \ \neg M_i$ then the DRI presented below generates a sequence of trees t_1, t_2, \dots which represent disjunctive normal forms of S_1, S_2, \dots and, as each t_i is generated, determines if there exists a substitution for $x_1, x_2, \dots, x_{in+n}$ which causes all branches of t_i to close. Herbrand's theorem⁷ guarantees that such a substitution will exist

⁶Recall that since we work with $\textcircled{P} e$, universal variables of e correspond to $-$ variables in the dpf for $\textcircled{P} e$ and existentially quantified variables of e correspond to $+$ variables.

⁷See [14].

if e is valid. Since t_i contains only atoms, the question of closure can be decided using only the unification algorithm⁸.

- bfs
1. Let $t_1 = \text{fep}(\theta)$, $i = 2$ and $T = t_1$
 2. Apply unify to T and multiple closed branch removal to the result. If the result is ϕ then terminate, otherwise
 3. Let t_i result from appending $O_3(t_1, \sigma_i)$ to each leaf of t_{i-1} , set T to t_i and i to $i + 1$. Go to 2.

Any expression of the first order predicate calculus (H) may be represented in E^9 . If we denote the set of type 1 expressions corresponding to these by Pd then bfs restricted to Pd constitutes a proof procedure for H in the sense that it is a proof procedure for Pd and every element of Pd is provable if and only if it corresponds to a theorem of H . Since the class of input trees for which bfs does not terminate corresponds exactly to those formulas of H which are not theorems we see that it is impossible to sharpen the input test so that it excludes exactly these input trees¹⁰. Generally speaking, bfs is a non-terminating DRI

⁸We define a procedure in the appendix which utilizes the unification algorithm (applying to pairs of atoms) and which always terminates with a substitution which closes the given tree or another value which indicates that no closing substitution exists.

⁹We note that if $e \in H$ is of the form $\exists xP(x)$ then its representation in E is $\exists \lambda xPx$.

¹⁰In fact, even if bfs is restricted to P , it will not terminate for those elements of P which are not theorems since the tree generated at step 3 will never be closed.

since it will not terminate for certain elements of any set S (determined by an applicability test) which contains Pd .

We have characterized a DRI as a well defined procedure which accepts a set of input trees satisfying stated properties, generates a sequence of system trees, and, if terminating, produces a specified set of system trees as output. This rather general characterization may be sharpened.

We note from the preceding examples that a DRI is applied by executing a sequence of operations whose order and nature conditionally depend upon properties of the system trees being processed. Each operation falls into one of two categories.¹¹ In the first category we have the inference operators of E , as well as other DRIs¹². Since these operations are the only operations which can produce new system trees, we call them effector operations. The second category consists of operations which provide the capability of recognizing arbitrary features which may be present in the system trees being processed. These operations will be called perceptor operations and include the applicability tests for the various DRIs as well as tests which control the order of application of the effector operations. A simple example of the latter would be the test for the empty tree.

¹¹For the purposes of this discussion, we exclude certain necessary (but peripheral) notions such as counting, recognizing 0 (zero) and similar arithmetically related operations.

¹²We note in particular that a DRI specification may be recursive. For examples see *fep* in appendix A.

If we designate the E-inference operators as primitive effector operations, we note that any non-primitive effector operation must ultimately reduce to primitive effector operations. For example, f_{pn_1} is defined directly in terms of primitive effector operations and f_{ep} is reducible to primitive effector operations since its constituent effector operations are reducible to primitives.

Analogously, we may specify a set of primitive perceptor operations¹³ and construct non-primitive perceptor operations which are ultimately reducible to these primitives. The perceptor operation "clash" used in the DRI which simulates Resolution, is an example of a non-primitive perceptor operation. Its specification is given in the appendix.

Finally, we note that the exact nature of a DRI depends not only on the above considerations, but also upon the language used to specify the DRI. The procedural nature of DRIs, as well as their conditional dependence on prescribed data features, requires that any formal DRI specification language have the properties usually associated with programming languages. In fact, DRIs may be specified in any of a number of languages¹⁴. Formalization of DRI specifications within a particular programming language is a fairly in-

¹³These are considered further in appendices A and G. We note that the primitive perceptor operations in combination with the language used to specify DRIs must at least allow construction of the applicability tests for the primitive effector operations.

¹⁴In appendix A we consider how DRIs may be specified in a problem oriented language embedded in the LISP meta language.

volved problem in its own right.

Since knowledge of the details associated with such a formalization are not required to understand the remainder of this chapter, we relegate them to appendix A.

2.0 Simulation of selected proof procedures

In this section, we shall demonstrate how DRIs may be used to simulate selected proof procedures of Prawitz, Robinson and Loveland as well as a semi-decision procedure of Friedman. Each of these procedures assumes that input expressions are in prenex normal form. (Friedman's assumes, in addition, that the expression is in Skolem normal form.) DRIs do not deal directly with expressions of the first order predicate calculus, but rather with expressions of *Pd*. It is thus necessary to describe how expressions in prenex normal form are represented and dealt with in *Pd*.

Suppose $\Pi R(z_1, \dots, z_n)$ is an expression of the first order predicate calculus in prenex normal form. Π is the quantifier prefix, $R(z_1, \dots, z_n)$ is the quantifier free matrix and z_1, \dots, z_n are variables occurring in Π . $\Pi R(z_1, \dots, z_n)$ is represented in *Pd* by the type 1 expression $\Pi'R'(z_1, \dots, z_n)$. Π' is obtained from Π by replacing occurrences of $\exists x$ with $\hat{\exists}\lambda x$ and $R'(z_1, \dots, z_n)$ is the prefix (as opposed to infix) equivalent of $R(z_1, \dots, z_n)$.

Suppose e is an initial expression and that $\Pi R(z_1, \dots, z_n)$ is in prenex normal form and equivalent to $\neg e$. If $\Pi = \Pi_1 \Pi_2$ where Π_1 contains only universal quantifiers and Π_2 is a

quantifier prefix beginning with \exists , then the initial tree for the expression e will be $\circ \textcircled{\text{T}} \Pi_2 R'(z_1, \dots, z_n)$. We note that given an initial tree Θ of this form, it is possible to define a terminating DRI in terms of λ -reduction and unabbreviation which takes Θ as input and produces as output a system tree Θ^* which has the form $\textcircled{\text{T}} R'(x_1, \dots, x_k)$ and is satisfiable if and only if Θ is satisfiable. (Where x_1, \dots, x_n are those z_i which correspond to universal quantification.) In defining the DRIs which follow, we shall thus assume that Θ^* rather than Θ is given as input.

2.1 A defined rule of inference which simulates a proof procedure of Prawitz

Given any closed formula E of the 1st order predicate logic, there exists a quantifier-free formula $M(x_1, \dots, x_n)$ such that E is valid iff there exists a finite subset of the set $S = \{M(t_1, \dots, t_n) : t_i \in H_F \text{ } i \leq n\}$ which is unsatisfiable. F is a prenex normal form for $\neg E$ and H_F is the Herbrand universe for F . The question of validity can thus be reduced to the problem of determining whether S contains an unsatisfiable finite subset.

The basic method proposed by Prawitz determines a minimal unsatisfiable subset of S (if E is valid) by sequentially examining the formulae S_0, S_1, \dots where $S_k = M(X_1, \dots, X_n) \& \dots \& M(X_{kn+1}, \dots, X_{kn+n})$. For each $k = 0, 1, \dots$ the procedure determines whether any substitution for the variables X_1, \dots, X_{kn+n} makes S_k inconsistent. If E is valid, there will exist a k for which the answer is "yes."

The feasibility of this procedure hinges on the existence of an efficient means of answering the question, "Does there exist a substitution for the variables X_1, \dots, X_n which makes S_k inconsistent?" Prawitz's original procedure made use of the fact that if S_k is transformed into DNF, then any substitution which makes S_k inconsistent must make contradictory at least one pair of literals (of the form $P(t_1, \dots, t_n)$, $\neg P(u_1, \dots, u_n)$) occurring in each disjunct (i.e. for each disjunct the substitution must simultaneously satisfy the atomic substitution conditions $u_1 = t_1, \dots, u_n = t_n$). A pair of literals for which the substitution condition $u_1 = t_1$ & ... & $u_n = t_n$ is satisfiable is said to be a possible contradiction with corresponding substitution condition $u_1 = t_1$ & ... & $u_n = t_n$. The entire procedure, hereafter called Unmodified Prawitz, is given below.

1. Transform the formula S_k into the disjunctive normal form D_k .
2. For each of the m clauses of the disjunction obtained in step 1, form the condition $\alpha_1^i \vee \dots \vee \alpha_{h_i}^i$ called C_i , where $\alpha_1^i, \dots, \alpha_{h_i}^i$ are the substitution conditions that correspond to possible contradictions among the formulae occurring in the i^{th} clause.
3. Form the conjunction $C_1 \& \dots \& C_m$ of conditions obtained in 2.
4. Decide whether the conjunction formed in 3 is satisfiable. (i.e. whether there is a set of substitution conditions,

one from each C_i , which are mutually compatible.)

A substitution σ which satisfies $C_1 \& \dots \& C_m$ will have the property that for each $1 \leq i \leq m$ σ satisfies α_j^i for some $1 \leq j \leq h_i$. Thus each disjunct of $D_k \sigma$ will contain a contradictory pair of literals and $D_k \sigma$ will be unsatisfiable.

The method given above may be reformulated in such a way as to avoid explicit development of S_k into DNF. This modified formulation, which includes several additional refinements of the original procedure, is the version of Prawitz that will be simulated. The reformulation is as follows:

Let S_k be written in conjunctive normal (CNF) form thus $(A_{11} \vee \dots \vee A_{1n_1}) \& \dots \& (A_{m1} \vee \dots \vee A_{mn_m})$ and represented by the following matrix:

$$A = \left\{ \begin{array}{l} A_{11}, A_{12}, \dots, A_{1n_1} \\ A_{21}, A_{22}, \dots, A_{2n_2} \\ \dots \\ A_{m1}, A_{m2}, \dots, A_{mn_m} \end{array} \right\}$$

A path in this matrix is determined by selecting exactly one literal from each line. The disjunctive normal form for the formula is obtained by forming a disjunction of all conjunctions formed by conjoining literals on paths. The formula S_k is then inconsistent if and only if there exists a set Σ of atomic substitution conditions which has the following properties:

1. Each path of the matrix contains at least one possible contradiction such that the atomic parts of the corresponding substitution condition belong to Σ .
2. The substitution conditions in Σ are simultaneously satisfiable.

The set Σ may be determined in various ways. Prawitz suggests, among others, the following:

Take the paths of the matrix in some order. For each path, as it is encountered, pick out a possible contradiction and form the corresponding substitution condition. If the newly formed substitution condition is simultaneously satisfiable with the current elements of Σ , then the new condition is added to Σ . Otherwise, the procedure backs up to the last path for which an untried substitution condition exists. If no such path exists, then S_k is satisfiable - otherwise all substitution conditions formed for the path and later paths are purged from Σ , the untried substitution condition is added to Σ and the procedure continues with the next path.


In building the set Σ it is not actually necessary to consider all paths of the matrix. In fact, it can be shown that if (A_{ij}, A_{kp}) $i \neq k$ is a possible contradiction with corresponding substitution condition α , then in further building Σ it is only necessary to consider those paths which contain A_{ij} or A_{kp} but not both.

Simulation of Prawitz¹

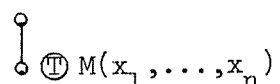
It has been noted previously that full expansion of a propositional formula produces a tree which explicitly represents a DNF for the original formula. Since both versions of Prawitz's method ultimately depend upon the DNF of the input formula, it is possible to simulate the essentials of Prawitz's method without constraining the form of the quantifier-free matrix $M(x_1, \dots, x_n)$. However, in an effort to maintain fairly close contact with the second version of Prawitz, we shall tolerate the inefficiencies introduced by assuming the quantifier-free matrix to be in CNF.

The DRI which simulates the above procedure consists of two major phases: 1) Production of the tree T_k representing the DNF of S_k and 2) Determining if there exists a substitution σ which closes T_k .

1 Production of the DNF tree for S_k

Theoretically, full expansion of the initial tree  produces a DNF tree for S_k . However, due to the particular form assumed for $M(x_1, \dots, x_n)$ and the relation between S_k, S_{k-1} it is possible to produce T_k in a more efficient manner.

CASE 1 $k = 0$

Starting with the tree  T_0 is obtained by applying the operators " $\textcircled{T} \&$ ", "copy", " $\textcircled{T} \vee$ " and

¹A formal description of this DRI is given in appendix A.

" $\textcircled{T} \neg$ " as indicated in "Description of Prawitz"². This produces a tree having the property that each branch contains one atom from each disjunct (clause) of $M(x_1, \dots, x_n)$. Furthermore, if the atom is part of the literal $\neg A_{ij}$ then its prefix in T_0 is " \textcircled{F} " otherwise " \textcircled{T} ".

CASE 2 $k > 0$

Let T be the tree resulting from substituting x_{kn+i} for x_i in T_0 . T_k is obtained by appending T to each leaf of T_{k-1} . This results in the same tree as would have been produced by applying the procedure for case 1 to $\textcircled{T} S_k$, however, it requires less computation.

2 Determining if T_k can be closed

The paths in T_k correspond exactly to the paths obtained from the matrix A . Rather than construct the set Σ of simultaneously satisfiable substitution conditions, we construct the closing substitution σ . The method used to construct σ , however, exactly parallels that used to construct Σ .

Assume that a set of substitutions $\{\sigma_1, \dots, \sigma_n\}$ has been constructed which closes the first n paths of T_k^* where $T_k^* = T_k \sigma_1 \dots \sigma_n$. We consider the $(n+1)^{\text{st}}$ path. If this path contains a possible contradiction (in our case two expressions of the form $\textcircled{T} P(t_1, \dots, t_n)$ and $\textcircled{F} P(u_1, \dots, u_n)$ which are unifiable under the substitution σ_{n+1}) we add σ_{n+1} to this set, and delete from the

²See appendix A.

remaining paths of T_k all those containing neither $\textcircled{T}P(t_1, \dots, t_n)$ nor $\textcircled{F}P(u_1, \dots, u_n)$ or containing both. The process is continued with the images of all remaining paths in $T_k^* \sigma_{k+1}$. If a path is encountered which is not unifiable, then the process is backed up to the last path for which untried unifying substitutions exist. If no such path exists then T_k can not be closed and we must consider T_{k+1} . Otherwise if the process backs up to the path which was associated with the closing substitution σ_i and for which the untried closing substitution σ_i^* exists, we delete $\sigma_i, \sigma_{i+1}, \dots, \sigma_n$ from $\{\sigma_1, \dots, \sigma_n\}$ add σ_i^* and continue as before. If the procedure processes all paths of T_k , then we are assured that the substitution $\sigma = \sigma_1 \sigma_2 \dots \sigma_m$ has the property that $T_k \sigma$ is closed - thus if we apply multiple closed branch removal to $T_k \sigma$ we obtain ϕ .

The DRI thus takes Θ as input, produces a sequence of trees (each deducible within the type theory logic from the previous one) and checks each to see if an immediate derivation of the empty tree is possible using only tree substitution and multiple closed branch removal. If the input formula is unsatisfiable, then some tree T_k will be generated which closes.

2.2 Simulated binary resolution

In this section, we shall specify a DRI which simulates a simplified version of the Resolution logic formulated by Robinson [34]. This logic is equivalent¹ to the first order predicate

¹Equivalent in the sense that given any formula of the first order predicate calculus, there corresponds a set of elements of Resolution logic which is unsatisfiable if and only if the given formula is valid.

calculus, has a single rule of inference and is comprised of sentences called clauses which possess a very uniform structure.

A complete presentation of Resolution logic is beyond the scope of the present section. We must therefore assume that the reader is generally familiar with Resolution and confine our attention to the following key definitions which derive from a version of Resolution described by Luckham [51].

A resolvent of two clauses² C_1 and C_2 is a third clause D obtained as follows:

- (i) If v_1, \dots, v_m are the variables of C_2 , and the highest variable of C_1 in the lexical order is u_k , let $\zeta = \{ \langle u_{k+1}, v_1 \rangle, \dots, \langle u_{k+m}, v_m \rangle \}$. (None of the variables in $C_2\zeta$ occurs in C_1 .)
- (ii) If there is a pair of sets of literals³, $L = \{L_1, \dots, L_k\}$ and $M = \{M_1, \dots, M_n\}$ such that $L \subseteq C_1$, and $M \subseteq C_2$ and the set $\{L_1, \dots, L_k, M_1'\zeta, \dots, M_n'\zeta\}$ is unifiable let σ_0 be the chosen simplest unifying substitution so that $L\sigma_0$ and $M\zeta\sigma_0$ are complementary literals; then D is the clause

$$(C_1 - L)\sigma_0 \cup (C_2 - M)\zeta\sigma_0$$

²A clause is a finite set (possibly empty) of literals (see footnote 3).

³A literal is either an atom or a negated atom.

(We note that M_i^1 is $\neg M$ if M_i is not a negated literal and M if M_i is $\neg M$.)

Resolution principle

From any two clauses C_1 and C_2 , infer a resolvent of C_1 and C_2 .

A refutation of a set of clauses T is a sequence C_1, C_2, \dots, C_n of clauses such that each clause either belongs to T or is inferred from two earlier clauses by a resolution, and $C_n = \phi$ (the empty clause). If we denote a set of clauses by T and all resolvents of pairs of clauses in T by $R^1(T)$; and define $R^{n+1}(T)$ to be $R^1(R^n(T))$, then it can be shown that $\phi \in R^n(T)$ for some n if and only if T is unsatisfiable. The following procedure is thus a proof procedure for the first order predicate calculus.

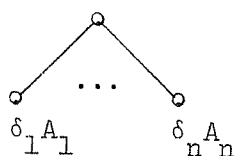
Binary resolution

1. Given an arbitrary formula e of the predicate calculus, calculate the prenex normal form for $\neg e$ and transform its quantifier free matrix into conjunctive normal form.
2. Let $C_1 \& \dots \& C_n$ be the result of introducing Skolem functions into the conjunctive normal form quantifier free matrix obtained in step 1.
3. Let $T = \{C_1, \dots, C_n\}$
4. Generate $R^k(T)$, for $k = 1, 2, \dots$. For each k , test to see if $\phi \in R^k(T)$.

Simulation of binary resolution

The logic of Binary Resolution may be embedded in E by establishing a correspondence between the set of clauses and a subset of ungrounded trees which preserves the operation of forming resolvents. System trees which correspond to clauses will be called E -clauses and the DRI which simulates the operation of forming resolvents will be called rslv. In order to establish the required correspondence, we now introduce the following definitions:

If $\{L_1, \dots, L_n\}$ is a clause, then the corresponding E -clause is the ungrounded tree



where A_i is the atom occurring in the literal L_i and

$$\delta_i = \begin{cases} \textcircled{T} & \text{if } L_i = A_i \\ & \text{and} \\ \textcircled{F} & \text{if } L_i = \neg A_i \end{cases}$$

(If C is a clause, and L a literal, then we denote the corresponding E -clause by D^* and the corresponding E -literal by L^* .) If A is an atom, then the spe δA is called an E -literal. (Note $\delta = \textcircled{T}$ or \textcircled{F} .) Two E -literals are complementary if their suffixes are identical and their prefixes are different. In particular, the complement of the E -literal L (denoted \bar{L}) is obtained by changing its prefix. The E -literals $L = \{L_1, \dots, L_n\}$ are contained

in the E -clause C (denoted $L \subseteq C$) if each L_i is the content of some leaf of C . Finally, we agree that the empty clause corresponds to the empty tree.

The operation of forming resolvants in Binary Resolution is simulated in E by the DRI rslv. This DRI is defined as follows:

1. Applicability test Two E -clauses C_1^* and C_2^* , whose patriarchs are on the same branch, are acceptable inputs for rslv if ζ is as given in (i) above and the following modification of (ii) is satisfied.

(ii) There is a pair of sets of E -literals, $L^* =$

$\{L_1^*, \dots, L_k^*\}$ and $M^* = \{M_1^*, \dots, M_m^*\}$ such that

$L^* \subseteq C_1^*$ and $M^* \subseteq C_2^*$ and the set $\{L_1^*, \dots, L_k^*,$

$\overline{M_1^* \zeta}, \dots, \overline{M_n^* \zeta}\}$ is unifiable.

2. Specification of rslv Let σ_0 be the chosen simplest substitution so that $L\sigma_0$ and $M\zeta\sigma_0$ are complementary E -literals.

(a) Calculate $O_3(C_2, \zeta)$

(b) Append $(O_3) C_1$ to each leaf of the result obtained in (a) which contains one of the M_i .

(c) Calculate $O_3(t, \sigma_0)$ where t is the tree resulting from (b).

(d) Apply multiple closed branch removal to the tree obtained in (c).

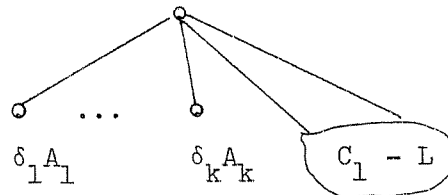
- (e) Apply node removal to each node of the tree obtained in (d) which contains $M_i \zeta \sigma_0$ for some i $1 \leq i \leq k$.
- (f) Apply duplicate branch removal repeatedly until no duplicate branches remain.

Remark If C_1^* and C_2^* satisfy the applicability test given above, then rslv has the following properties:

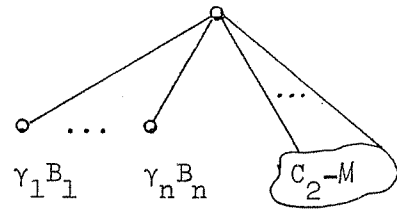
1. It always terminates
2. Its output $\text{rslv}(C_1^*, C_2^*)$ is an \bar{E} -clause whose leaves consist entirely of the \bar{E} -literals corresponding to the literals $(C_1 - L)\sigma_0 \cup (C_2 - M)\zeta\sigma_0$.

Each of the above steps ((a) - (f)) constitutes a DRI. Since each terminates and is executed only once, rslv must terminate. Furthermore, all steps will be executed since the applicability test for each is satisfied. Part 2 of the above remark is justified by the following considerations:

Without loss of generality assume that C_1^* has the form

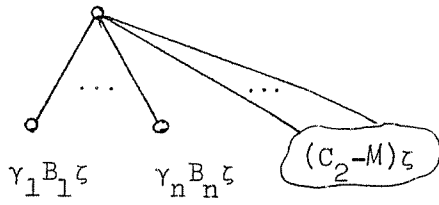


and C_2^* the form

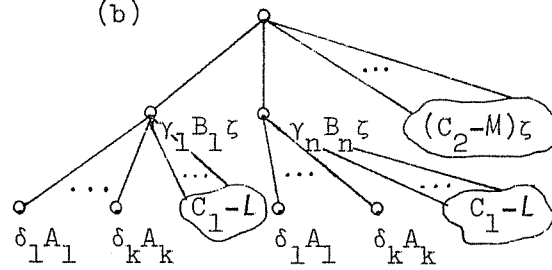


Execution of steps (a) - (f) produces the following sequence of system trees.

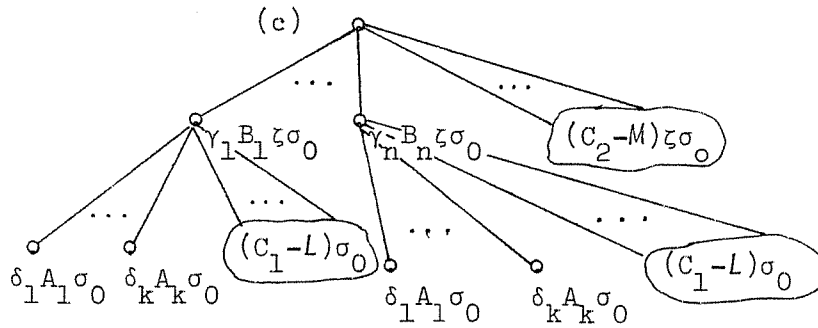
(a)



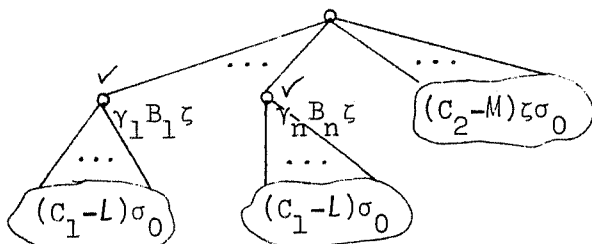
(b)



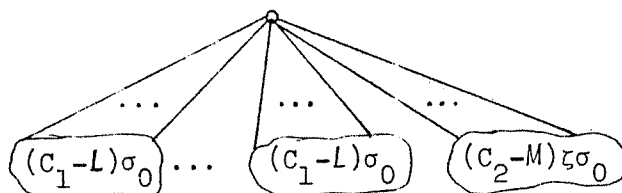
(c)



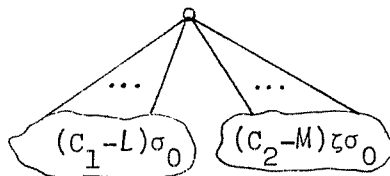
(d) By hypothesis, the indicated branches are closed. Thus, execution of (d) produces



(e) Removal of the indicated nodes is allowable and produces



(f) Finally, multiple applications of duplicate branch removal yields



This is an E -clause and corresponds to the clause $(C_1 - L)\sigma_0 \cup (C_2 - M)\zeta\sigma_0$ which is the resolvent of C_1 and C_2 .

We thus note that if C_1, C_2, D are clauses corresponding to the E -clauses C_1^*, C_2^* and D^* and if D is the resolvent of C_1 and C_2 then $D^* = \text{rslv}(C_1^*, C_2^*)$. Thus, whenever it is possible to infer the resolvent of two clauses within Resolution logic, it is possible to infer the corresponding E -clause within the framework of E . We are now in a position to specify a DRI which simulates Binary Resolution.

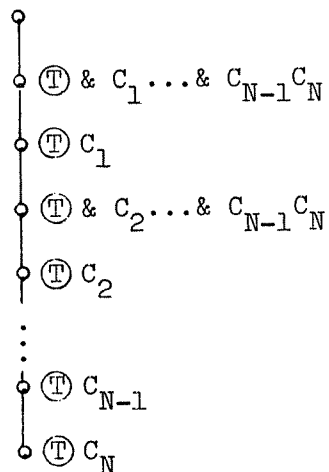
E -Binary Resolution


We assume an input θ^* where θ^* is obtained from the initial tree θ for the Pd equivalent of the expression obtained as the result of step 1 of Binary Resolution. (θ^* is described on page

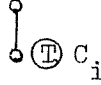
.) The input θ^* has the form $\textcircled{\text{T}} \& C_1 \& \dots \& C_{N-1} C_N$. E -Binary Resolution is divided into two phases. We note that each is a DRI.

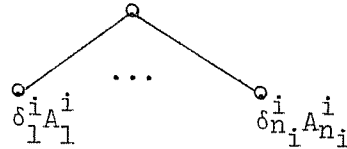
Phase 1 Obtain the constituent E -clauses T^* of θ^* .

(a) Apply O_2 (relative to $\textcircled{\text{T}} \&$) $N - 1$ times
this produces



Each node having content of the form $\textcircled{T} C_i$ determines a proper forest of this tree. Thus, for each $1 \leq i \leq N$ apply O_7 to produce the ungrounded tree 

(b) Apply f_{pn} to each ungrounded tree  formed in (a). This produces for $1 \leq i \leq N$



where A_j^i is the atom contained in the j^{th}

literal L_j^i of C_i and $\delta_j^i = \begin{cases} \textcircled{T} & \text{if } L_j^i = A_j^i \\ \textcircled{F} & \text{if } L_j^i = \neg A_j^i \end{cases}$.

The DRI specified by steps (a) and (b) thus takes the single tree Θ^* as input and produces a set of E -clauses corresponding to the constituent clauses of the content of Θ' .

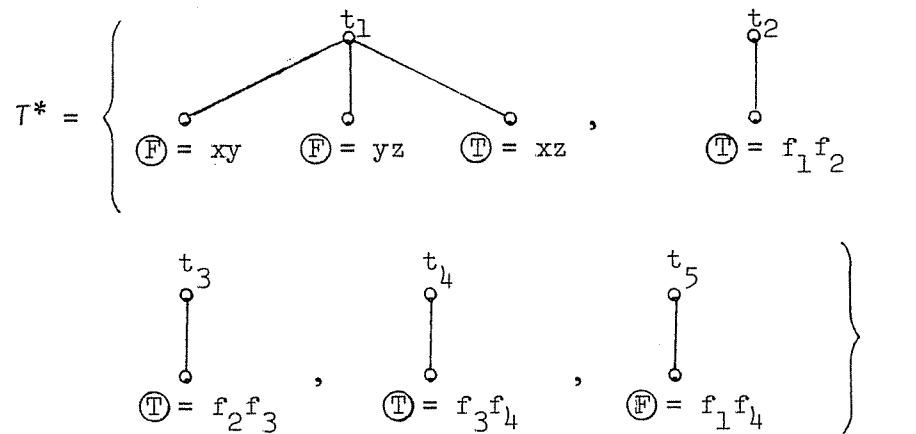
Phase 2 Form the sets corresponding to $R^k(T)$ for $k = 1, 2, \dots$

- (a) Let $K = T^*$ (the E -clauses produced by phase 1).
- (b) Form $R(K)$ by applying $rslv$ to each pair of elements of K satisfying the applicability test for $rslv$.
- (c) If $\phi \in R(K)$ then terminate. Otherwise,
- (d) Set $K = R(K)$ and go to (b).

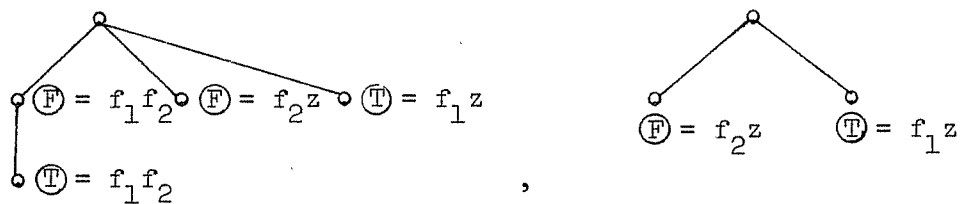
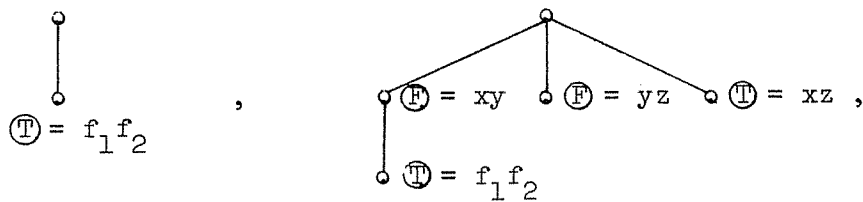
The following is an example of E -Binary Resolution simulating Binary Resolution applied to the formula $\forall x \forall y \forall z (x=y \ \& \ y=z \supset x=z) \supset \forall u \forall v \forall w (u=v \ \& \ v=w \ \& \ w=s \supset u=s)$.

example

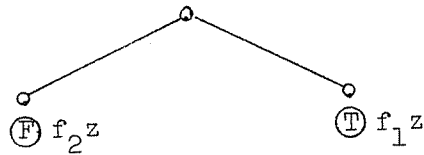
θ^* is $\& C_1 \& C_2 \& C_3 \& C_4 C_5$ where C_1 is $\forall \neg =xy \forall \neg =yz =xz$; C_2 is $= f_1 f_2$; C_3 is $= f_2 f_3$; C_4 is $= f_3 f_4$; C_5 is $\neg = f_1 f_4$. f_1, f_2, f_3, f_4 and f_5 are Skolem functions of zero arguments introduced for u, v, w and s respectively. Phase 1 produces the \bar{E} -clauses



t_1 and t_2 satisfy the applicability test for rslv. We thus obtain the following sequence. (Note that $\zeta = \phi, \sigma_0 = \{ \langle f_1, x \rangle, \langle f_2, y \rangle \}$.)



Thus $\text{rslv}(t_1, t_2)$ is



In a similar manner, we obtain the remaining elements of $R^1(T^*)$.

$$R^1(T^*) - T^* = \left\{ \begin{array}{l} \begin{array}{c} t_6 \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = f_2z \quad \textcircled{T} = f_1z \end{array}, \begin{array}{c} t_7 \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = xf_1 \quad \textcircled{T} = xf_2 \end{array} \end{array} \right\}$$

$$\begin{array}{c} t_8 \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = f_3z \quad \textcircled{T} = f_2z \end{array}, \begin{array}{c} t_9 \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = xf_2 \quad \textcircled{T} = xf_3 \end{array}$$

$$\left. \begin{array}{c} t_{10} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = f_4z \quad \textcircled{T} = f_3z \end{array}, \begin{array}{c} t_{11} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = xf_3 \quad \textcircled{T} = xf_4 \end{array} \right\}$$

$$R^2(T^*) - R^1(T^*) = \left\{ \begin{array}{l} \begin{array}{c} t_{12} \\ \circ \\ \mid \\ \circ \\ \textcircled{T} = f_1f_3 \end{array}, \begin{array}{c} t_{13} \\ \circ \\ \mid \\ \circ \\ \textcircled{T} = f_2f_4 \end{array} \end{array} \right\}$$

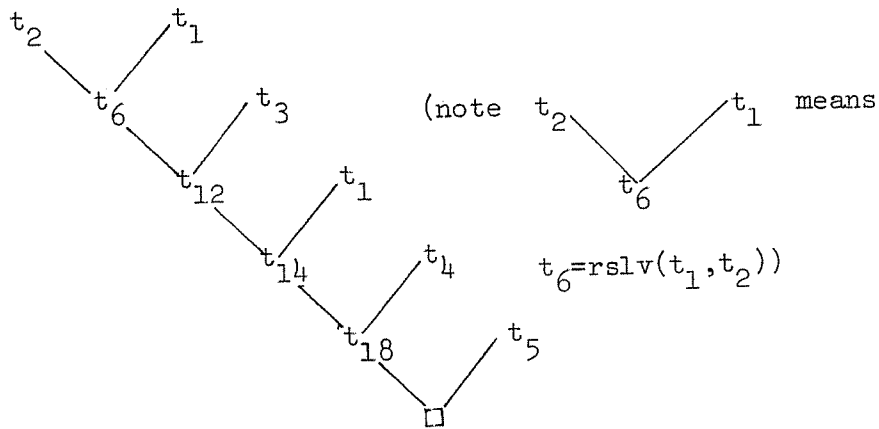
$$R^3(T^*) - R^2(T^*) = \left\{ \begin{array}{l} \begin{array}{c} t_{14} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = f_3 z \quad \textcircled{T} = f_1 z \end{array}, \quad \begin{array}{c} t_{15} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = x f_1 \quad \textcircled{T} = x f_3 \end{array} \end{array} \right.$$

$$\left. \begin{array}{l} \begin{array}{c} t_{16} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = f_4 z \quad \textcircled{T} = f_2 z \end{array}, \quad \begin{array}{c} t_{17} \\ \circ \\ \swarrow \quad \searrow \\ \circ \quad \circ \\ \textcircled{F} = x f_2 \quad \textcircled{T} = x f_4 \end{array} \end{array} \right\}$$

$$R^4(T^*) - R^3(T^*) = \left\{ \begin{array}{c} t_{18} \\ \circ \\ \circ \\ \textcircled{T} = f_1 f_4 \end{array} \right\}$$

$$R^5(T^*) - R^4(T^*) = \{\phi\} .$$

We may thus obtain a refutation of $\{t_1, t_2, t_3, t_4, t_5\}$ as indicated below:



We note in conclusion that given any Binary Resolution refutation of a set of clauses, there corresponds an \bar{E} -Binary Resolution refutation which can be obtained by applying the phase 2 DRI to the \bar{E} -clauses produced by the phase 1 DRI. There is also reason to believe that given any strategy for producing resolvents, there exists a DRI which simulates it. In particular, P_1 -Binary Resolution [35] can be simulated by strengthening the applicability test for rslv. This can be accomplished by requiring that one of the input clauses be positive and the other negative (i.e. all literals of one be of the form $\textcircled{T}A$ and at least one literal of the other be of the form $\textcircled{F}B$). If we wish to insure that the resolvent possess a certain property, we can define a DRI which applies rslv to the inputs and tests the result for the property. If the property is satisfied, then $\text{rslv}(C_1, C_2)$ is the output, if the property is not satisfied then the DRI is not applicable. Semantic Resolution [40] would be an example of a type of Resolution requiring this kind of DRI.

2.3 Simulated Model Elimination

The Model Elimination proof procedure proposed by Loveland [20] is similar to the Binary Resolution procedure presented in the last section. Both procedures operate upon sets of clause-like¹ elements representing the negation of the formula being processed and attempt to produce a terminal element which is manifestly un-

¹If clauses are considered to be sets of literals rather than disjunctions of literals then Resolution deals with clauses and Model Elimination with ordered clauses.

satisfiable. In addition, both procedures produce as a result of applying their respective inference operators sequences of intermediate elements, which possess the same uniform structure as the initial input elements.

Model Elimination, however, differs from Binary Resolution in several respects. Unlike Binary Resolution, Model Elimination deals with sets of ordered literals. Further, Model Elimination requires three inference operators where Binary Resolution requires but one. Detailed specification of a Model Elimination proof procedure is thus somewhat more complicated than the specification of a Binary Resolution proof procedure since the relative order of inference operator application must be taken into account.

Several Model Elimination papers have appeared in recent years. The procedure outlined below is one of three presented in [20]. Because a complete presentation of this material is beyond the scope of this thesis, we will assume that the reader is generally familiar with Model Elimination.

In the following discussion, the notions of clause and literal will have the meanings given in the last section. Additionally, we let T stand for the set of clauses which represent the negation of the formula to be processed.

Model Elimination deals with clause-like elements called chains which consist of ordered finite sets of literals. Elementary chains are those finite sets of ordered literals which are obtained from clauses. The initial stock of chains is called the auxiliary set

(denoted $M_0(T)$) and consists of certain elementary chains. In particular, there is one elementary chain in $M_0(T)$ for a given clause of T with a given first literal. Literals comprising chains are partitioned into A-literals and B-literals. Elementary chains by definition contain only B-literals. Arbitrary chains may contain both. A chain is preadmissible if three conditions are satisfied:

- a) Complementary B-literals are separated by A-literals.
- b) If an A-literal and B-literal are identical, then the B-literal must precede the A-literal.
- c) No two A-literals have identical atoms.

A chain is admissible if it is preadmissible and its last element is a B-literal.

The Model Elimination procedure to be simulated, possesses three operators which produce new chains from existing chains. Each operator accepts one (or two) input chains satisfying stated conditions and produces a single output chain called the derived chain. A deduction consists of a finite sequence K_0, \dots, K_n of chains such that K_{i+1} is derived from K_i $0 \leq i \leq n - 1$.

The operations used to produce derived chains are defined below. In each case we shall assume that unless otherwise stated the classification of a literal in the derived chain is hereditary (i.e. if a literal L_1 in the derived chain is derived from the literal L_0 in an input chain, then L_1 is an A-literal (B-literal) if L_0 is an A-literal (B-literal)).

Basic Extension The input chains are an admissible chain K and a chain $C \in M_0(T)$. Let L_1 be the last literal of $K\xi_K^2$ and L_2 the first literal of $C\eta_c^3$. If a match⁴ is possible between L_1 and L_2 , then let σ be the most general unifier associated with the match. (If no match is possible, then Basic Extension is not applicable to the input chains.) The derived chain is obtained by deleting $L_2\sigma$ from $C\eta_c\sigma$ and appending the result to $K\xi_K\sigma$. The literal $L_1\sigma$ becomes an A-literal.

Basic Reduction The input chain is an admissible chain K . A match is sought for some A-literal L_1 and a B-literal L_2 following L_1 in K . (If no match is possible, then Basic Reduction is not applicable to K .) Let σ be the most general unifier for L_1 and L_2 . The derived chain is obtained by deleting $L_2\sigma$ from $K\sigma$.

Basic Contraction The input chain is a preadmissible chain K . The derived chain is obtained by deleting all A-literals from K which follow the last B-literal of K .

²If v_1, \dots, v_k are variables occurring in E , then by $E\xi_E$ we mean the chain obtained by applying the substitution $\xi_E = \{\dots, \langle x_i, v_i \rangle, \dots\}$ to E . (Note: for no i and j is it the case that $x_i = v_j$.)

³Same as footnote 2 except the substitution $\eta_E = \{\dots, \langle y_i, v_i \rangle, \dots\}$ is used. The sets $\{v_1, \dots, v_k\}$, $\{x_1, \dots, x_k\}$ and $\{y_1, \dots, y_k\}$ are always assumed to be pairwise disjoint.

⁴ L_1 and L_2 has a possible match if there is a most general unifier σ for their atoms and $L_1\sigma, L_2\sigma$ are complementary.

The Model Elimination Procedure C^1

Commencing with the auxiliary set $M_0(T)$, the three operations given above may be used to generate a sequence of sets of chains $C_0^1 \subseteq C_1^1 \subseteq C_2^1 \subseteq \dots$ having the property that the set T is unsatisfiable if and only if there is some N such that the empty chain (consisting of no literals and assumed to be admissible) is an element of C_N^1 . The sets C_n^1 are defined as follows:

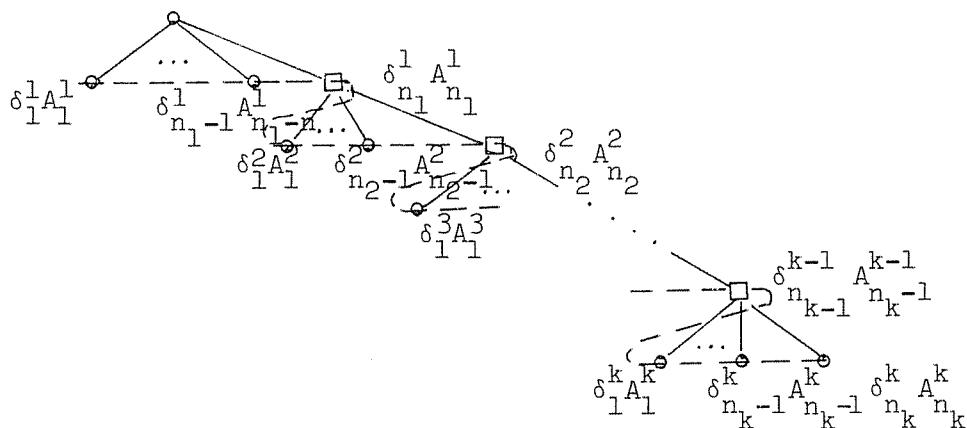
$$C_0^1 = M_0(T)$$

$C_n^1 = \{K : K \text{ is an admissible chain deducible from } M_0(T) \text{ whose deduction contains at most } n \text{ applications of Basic Extension.}\}$

The Model Elimination proof procedure C^1 thus consists of generating the sets C_0^1, C_1^1, \dots and checking each for the presence of the empty chain. If T is unsatisfiable, then C^1 will always terminate. If T is satisfiable, then it will not.

Simulation of C^1

Within the framework of E , and arbitrary chain $L_1^1, \dots, L_{n_1}^1, L_1^2, \dots, L_{n_2}^2, \dots, L_1^k, \dots, L_{n_k}^k$ with the A-literals $L_{n_i}^i$ $1 \leq i \leq k - 1$ has the following representation:



Where the order of the literals is indicated by the dashed line and where A_j^i is an atom and

$$\delta_j^i = \begin{cases} \textcircled{\text{T}} & \text{if } L_j^i = A_j^i \\ & \text{and} \\ \textcircled{\text{F}} & \text{if } L_j^i = \neg A_j^i \end{cases}$$

In particular, we allow the case where a node containing an A-literal has no successor or only a single immediate successor. (In the latter case, if the A-literal is not the last literal of the chain, then its immediate successor is also an A-literal.) An elementary chain has the same representation as a clause, (see the previous section). We shall denote the system tree corresponding to the chain K by K^* .

The proof of procedure C^1 can be simulated in E in the sense that given a deduction K_0, K_1, \dots, K_n of the chain K , it is possible to produce a sequence of system trees $K_0^*, K_1^*, \dots, K_n^*$ ⁵ using the

⁵In this particular case, the correspondence between K_i and K_i^* implied by the notation is slightly inaccurate. The exact corre-

three DRIs ext, red, and cont specified below. These correspond to the Model Elimination operators Basic Extension, Basic Reduction and Basic Contraction respectively. In fact, if we ignore certain minor differences⁶, it is true that if Basic Extension applied to K_A and K_B produces K_C or Basic reduction applied to K_A produces K_B or Basic contraction applied to K_A produces K_B , then $\text{ext}(K_A^*, K_B^*) = K_C^*$ or $\text{red}(K_A^*) = K_B^*$ or $\text{cont}(K_A^*) = K_B^*$.

The DRIs ext, red and cont are defined as follows:

ext The input E -chains⁷ are an admissible E -chain K^* and an elementary E -chain $C^* \in M_0(T^*)$. (Recall that an elementary E -chain is obtained from an elementary chain as indicated above.)

1. $K^* \xi_K$ and $C^* \eta_C$ are formed by applying $O_3(K^*, \xi_K)$ and $O_3(C^*, \eta_C)$ respectively. (If no match is possible between the last literal L_1 of $K^* \xi_K$ and the first literal L_2 of $C^* \eta_C$ then ext is not applicable to the inputs.) Let σ be the most general unifier of the atoms of L_1 and L_2 . Form the restricted append of $K^* \xi_K$ and $C^* \eta_C$ and let the result be t . Apply $O_3(t, \sigma)$.

⁵(continued) spondence may be inferred from the definitions of the DRIs ext, red and cont and is discussed at the end of this section.

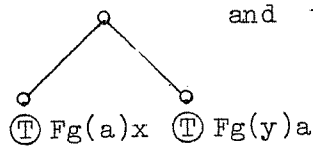
⁶These differences are discussed at the end of this section.

⁷Here we use the same sort of notation as in the last section. An E -chain K^* is the system tree which represents the chain K .

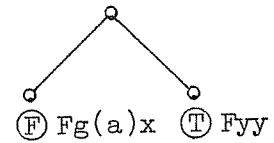
2. a) If $C^* \eta_C$ contains only one branch, then $\text{ext}(K^*, C^*) = O_3(t, \sigma)$
- b) If $C^* \eta_C$ contains more than one branch, then $\text{ext}(K^*, C^*)$ is the result of applying multiple closed branch removal to the branch of $O_3(t, \sigma)$ which contains the literal corresponding to L_2 .
3. The literal of $\text{ext}(K^*, C^*)$ corresponding to L_1 is designated an A-literal.

ext example

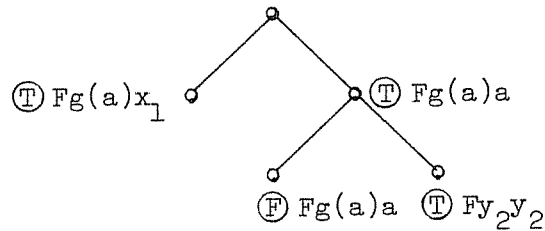
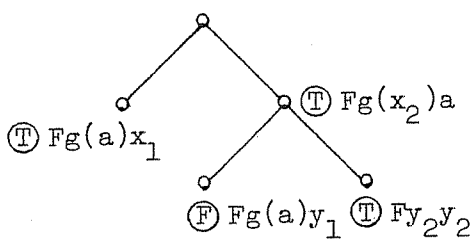
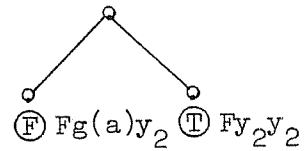
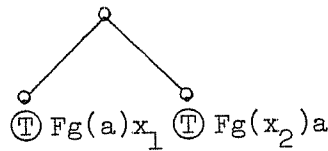
Let K_0 be



and t_1 be

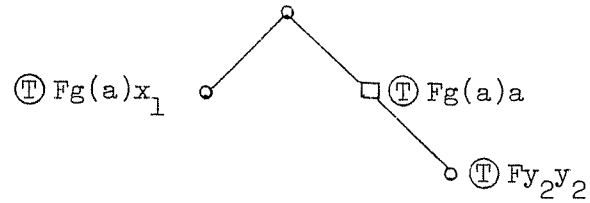


step 1 produces



where $\sigma = \{ \langle a, x_2 \rangle, \langle a, y_1 \rangle \}$

steps 2b and 3 produce

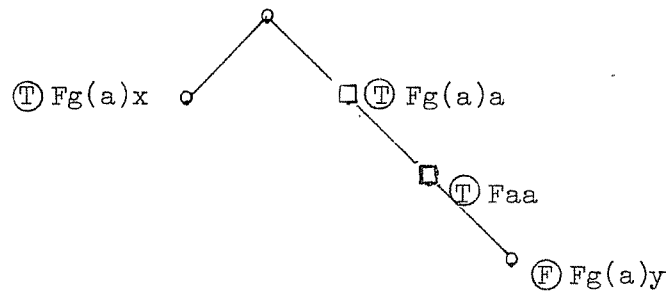


(Note that A-literals are indicated by square nodes.)

red The input E -chain is an admissible E -chain K^* . A match is sought for some A-literal L_1 and a B-literal L_2 following L_1 in K^* . (If the match is impossible, red does not apply to the input.) Let σ be the most general unifier for the atoms of L_1 and L_2 . If L_2 is not the last literal of K^* then $\text{red}(K^*)$ is the result of applying multiple closed branch removal (O_4) to the branch of $O_3(K^*, \sigma)$ corresponding to the literal L_2 . If L_2 is the last literal of K^* then $\text{red}(K^*) = O_3(K^*, \sigma)$.

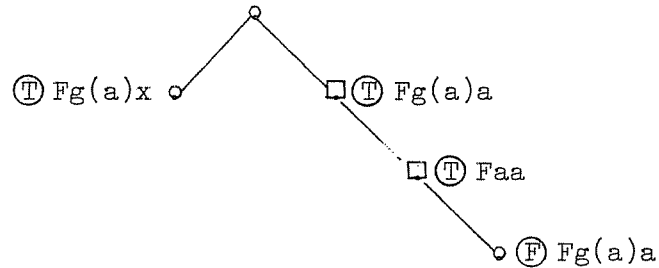
red example

Let K_2 be



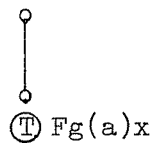
there is a match between $\oplus Fg(a)y$ and $\oplus Fg(a)a$ with $\sigma = \{ \langle a, y \rangle \}$

since $\textcircled{F} Fg(a)y$ is the last literal, $\text{red}(K_2)$ is



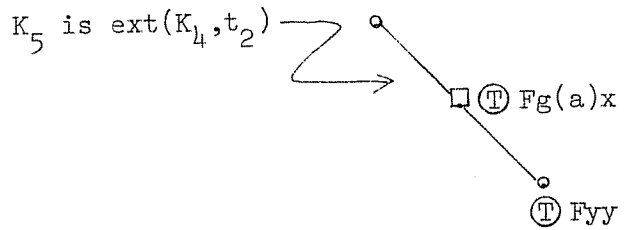
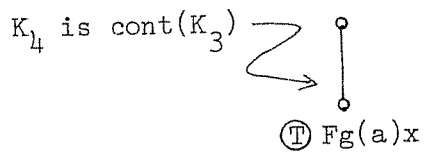
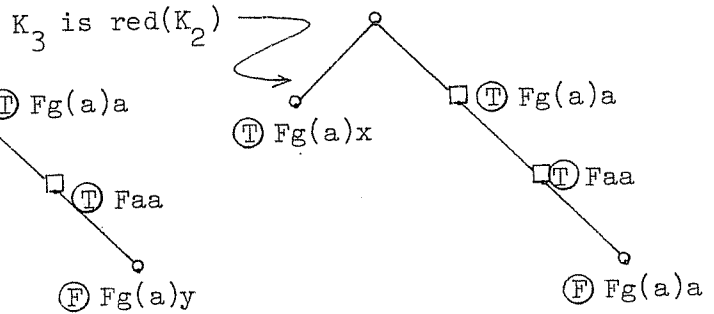
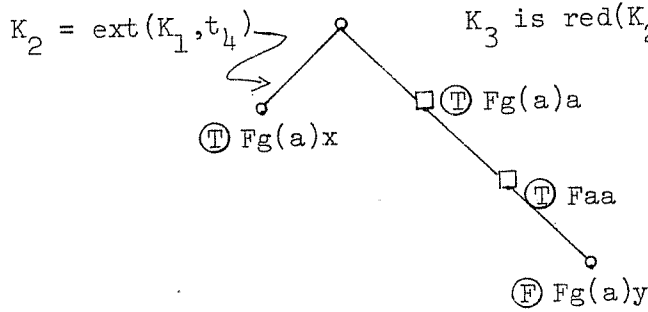
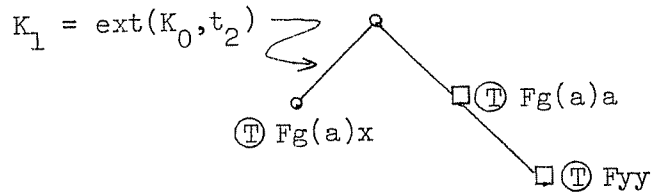
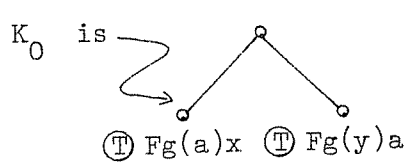
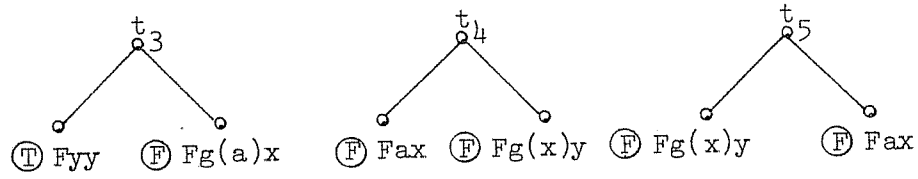
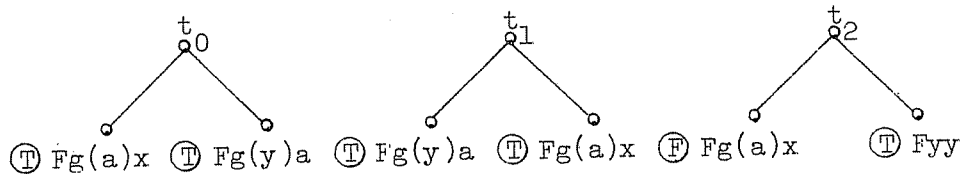
cont The input E -chain is a preadmissible E -chain K^* which contains at least one A-literal and whose last A-literal is followed by exactly one B-literal. The E -chain $\text{cont}(K^*)$ is the result of applying multiple closed branch removal to the last branch of K^* .

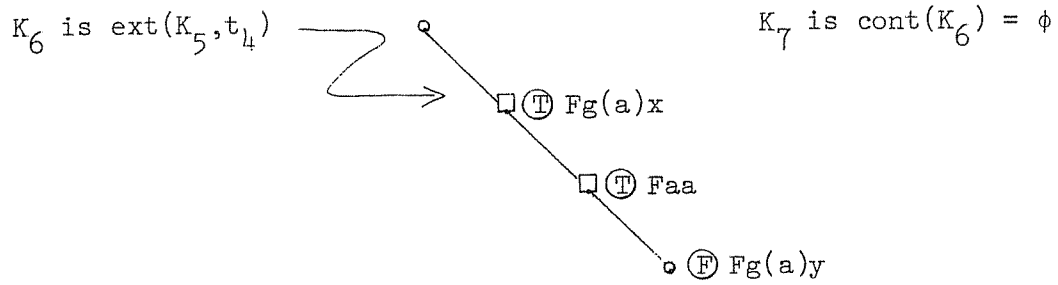
cont example Let K_3 be $\text{red}(K_2)$, then $\text{cont}(K_3)$ is



The following example illustrates the use of the preceding DRIs. It is essentially the E -equivalent of the example presented in [20].

example Refutation of $T = \{Fg(a)x \vee Fg(y)a, \neg Fg(a)x \vee Fyy, \neg Fax \vee \neg Fg(x)y\}$ $M_0(T^*)$ contains the six E -chains





This refutation corresponds to the Model Elimination deduction given below.

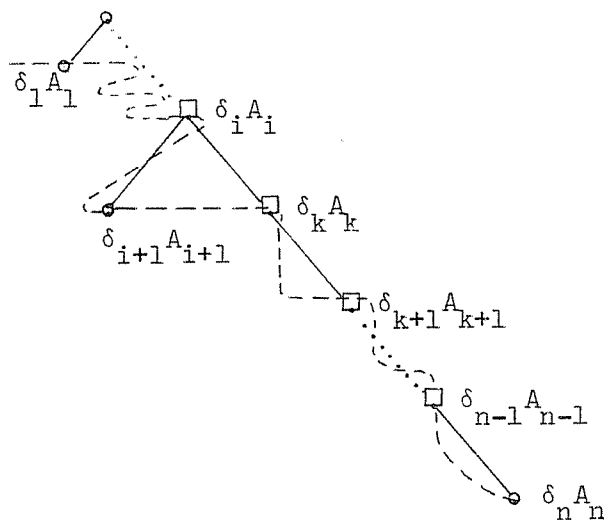
- Given clauses
- I. $Fg(a)x \vee Fg(y)a$
 - II. $\neg Fg(a)x \vee Fyy$
 - III. $\neg Fax \vee \neg Fg(x)y$

<u>Deduction</u>	(A-literals are underlined.)	<u>Reason</u>
1.	$Fg(a)x \quad Fg(y)a$	Initial chain I
2.	$Fg(a)x \quad \underline{Fg(a)a} \quad Fyy$	Extension using II
3.	$Fg(a)x \quad \underline{Fg(a)a} \quad \underline{Faa} \quad \neg Fg(a)y$	Extension using III
4.	$Fg(a)x \quad \underline{Fg(a)a} \quad \underline{Faa}$	Reduction
5.	$Fg(a)x$	Contraction
6.	$\underline{Fg(a)x} \quad Fyy$	Extension using II
7.	$\underline{Fg(a)x} \quad \underline{Faa} \quad \neg Fg(a)y$	Extension using III
8.	$\underline{Fg(a)x} \quad \underline{Faa}$	Reduction
9.	ϕ (Empty chain)	Contraction

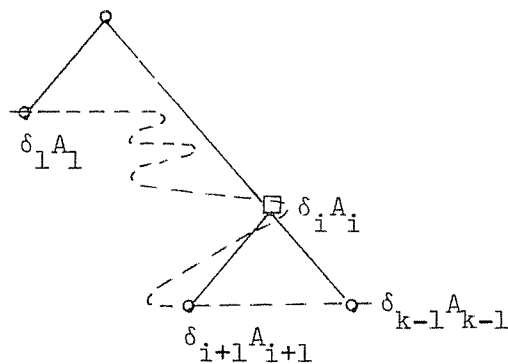
The primary differences between the Model Elimination operators and the DRIs given previously have to do with the contraction operation. We note that immediately preceding any contraction (in the Model Elimination procedure) either a reduction operation or a degenerate⁸ extension operation must have been applied. In either case, the removed B-literal is the conjugate of some A-literal. Since the logic of E allows only closed branches to be removed, this literal is required to justify removal of the A-literals during contraction. Thus, the DRIs ext and red behave exactly as their Model Elimination counterparts with the exception that under the appropriate conditions, the B-literal necessary for contraction is left in the output E -chain. The DRI cont then removes this B-literal as well as all A-literals following the next to last B-literal. That the DRI cont removes the correct A-literals is demonstrated as follows:

Suppose $L_1, \dots, L_k, \dots, L_{n-1}, L_n$ is an admissible chain in which L_{k-1} and L_n are B-literals and $L_k, L_{k+1}, \dots, L_{n-1}$ are A-literals. Further assume that L_n is the complement of some preceding A-literal. Basic Reduction then produces $L_1, \dots, L_k, \dots, L_{n-1}$ with $\sigma = \phi$ (the empty substitution). Basic Contraction applied to the result produces L_1, \dots, L_{k-1} . Now the E -chain for L_1, \dots, L_n is

⁸A degenerate extension is an extension in which the chain chosen from $M_0(T)$ consists of a single literal. Thus, after a degenerate extension, no descendant literals of this chain occur in the derived chain.



where the order of literals is as indicated by the dashed line, the DRI removes the branch segment which extends from the node containing $\delta_i A_i$ to the leaf containing $\delta_n A_n$. This is due to the fact that since there are no B-literals in the sequence L_k, \dots, L_{n-1} , none of the nodes containing $\delta_k A_k, \dots, \delta_{n-1} A_{n-1}$ has more than one immediate successor. Further, by hypothesis L_{k-1} is a B-literal and thus the node containing the A-literal which is its immediate predecessor has more than one immediate successor. Thus the DRI cont produces



which is the E -chain for L_1, \dots, L_{k-1} .

Finally we note that the correspondence between the operations of the Model Elimination procedure and the DRIs ext , red and $cont$ while not exact, is sufficiently close as to allow any sequence of Model Elimination operations to be mimicked by these DRIs and vice versa. The E -equivalent of the classes C_0^1, C_1^1, \dots may thus be formed in exactly the same manner as is done in the Model Elimination procedure C^1 . If the empty tree is produced at any stage, then we have produced a proof of the initial expression within the logic E since O_{10} may then be applied to the tree containing the patriarchs of the initial E -elementary chains.

2.4 Simulation of Friedman's decision procedure for the prefix

$$\underline{(\exists y_1)(\exists y_2)(\forall z_1) \dots (\forall z_n)}$$

The last system selected for simulation is the decision procedure of Joyce Friedman for the class of first order formulas whose Skolem Normal form is $(\exists y_1)(\exists y_2)(\forall z_1) \dots (\forall z_n)M$. This system, while the simpler of the two systems presented in [10], is sufficiently different from those described previously in this thesis as to provide an interesting example of DRI simulation. Because of the relatively complicated nature of this system, we must refer the reader to Friedman's paper for details, examples and motivating discussion. The following outline is included mainly to provide a reference point for the discussion of the simulating DRI.

We assume that the input formula has the form

$$(1) (\exists y_1)(\exists y_2)(\forall z_1)\dots(\forall z_n)M$$

where M is a quantifier free matrix containing the N functional variables¹ F_1, \dots, F_N . Further, it is assumed that all permissible elementary parts are included in M ². Let A_1, \dots, A_m be the elementary parts occurring in M . It is possible (in theory) to construct a table for (1) whose columns have the heading:

y_1	y_2	z_1	\dots	z_n	A_1	A_2	\dots	A_m
-------	-------	-------	---------	-------	-------	-------	---------	-------

Entries for the first $n + 2$ columns are natural numbers whose choice depends on the exact form of the prefix of (1). The remaining columns contain instances of the elementary parts which head the particular column. The particular instance is determined by the values assigned to the variables $y_1, y_2, z_1, \dots, z_n$ for the row in which the given instance is to occur. The reader is referred to [10] for details and examples of the exact construction of this table. For our purposes, it is only necessary to know that a given row of the table contains specific values for each of the variables $y_1, y_2, z_1, \dots, z_n$ that the values of z_1, \dots, z_n depend upon values arbitrarily (but systematically) assigned to y_1, y_2 , and that the instances of the elementary parts within the row are determined by the values taken on by the variables in the row. The table, in

¹Functional variables are here taken to mean predicate letters.

²For example, if M does not contain the elementary part $F_i z_j y_1$, then replace M with the equivalent matrix $M \& (\neg F_i z_j y_1 \vee F_i z_j y_1)$. (See Friedman [10])

essence, amounts to instantiation of (1) over its Herbrand universe.

It can be demonstrated that (1) is a theorem if and only if it is not falsifiable in the decision table described above. In order that (1) be falsifiable in this table, it must be possible to consistently³ assign truth values to the elementary part instances in each row in such a way that the corresponding instance of M is false.

In considering possible truth value assignments for each of the rows of the decision table, it is only necessary to consider those truth value assignments which cause M to be false. These are called falsifying systems for M and can be computed for a given M in a straight forward manner.

Friedman observed that the consistency requirement (mentioned above) limits the choice of possible falsifying systems for a given row of the decision table. Choice of a given falsifying system for a particular row may, for example, make it impossible to falsify some other row or at least make it impossible to falsify some other row with certain falsifying systems.

Friedman's decision procedure utilizes these observations (in the form of three rules) to reject certain falsifying systems or certain combinations of falsifying systems from the set of falsifying systems for M . The first two rules reject falsifying systems whose assignment to one row of the table would make it impossible

³For example, an elementary part instance such as $F_i(1,2)$ might occur in two or more rows of the table. The truth value assignments for each of these rows must assign the same truth value to each occurrence of $F_i(1,2)$ if the assignments are to be consistent.

to falsify some other row. The third rule determines when a pair of falsifying systems conflict and splits the set containing these into two descendant sets, neither of which contains both elements of the pair.

Commencing with the set S_f of falsifying systems for M , rules 1, 2 and 3 are applied to S_f and its descendants to form the sets $\{S_1^i, \dots, S_{n_i}^i\}$. For any formula of the form (1), there exists a K such that rules 1, 2 or 3 do not apply to any of the sets comprising $\{S_1^K, \dots, S_{n_K}^K\}$. If $\bigcup_{j=1}^{n_K} S_j^K = \phi$, then (1) is a theorem. If $\bigcup_{j=1}^{n_K} S_j^K \neq \phi$, then (1) is not a theorem. Rules 1, 2 and 3 are presented for reference below. They are stated exactly as in [10].

Rule 1 Let the elementary parts which contain the functional variable F_i and none of the individual variables z_1, \dots, z_n be $A_{i1}, A_{i2}, \dots, A_{in_i}$. For each possible set of truth-values a_1, a_2, \dots, a_N , if there is no remaining system of S in which

a_1 is assigned to all of the elementary parts $A_{11}, A_{12}, \dots, A_{1n_1}$,
 a_2 is assigned to all of the elementary parts $A_{21}, A_{22}, \dots, A_{2n_2}$,
and a_N is assigned to all of the elementary parts $A_{N1}, A_{N2}, \dots, A_{Nn_N}$,

and in which the assignment to any two elementary parts B_1, B_2 is the same whenever $S_{y_1}^{y_2} B_1 |^4$ is identical with $S_{y_1}^{y_2} B_2 |$, then delete from S all systems in which for some individual variable u , $F_1(u, \dots, u) = a_1$, $F_2(u, \dots, u) = a_2 \dots$, and $F_N(u, \dots, u) = a_N$.

⁴The notation $S_{x_1 \dots x_n}^{y_1 \dots y_n} D |$ means the same thing as $D\sigma$ where $\sigma = \{\dots, \langle x_i, y_i \rangle, \dots\}$.

Rule 2 For each possible set of truth-values $a_{11}, a_{12}, \dots, a_{Nh_N}$, if there is no remaining system of S in which the elementary parts $A_{11}, A_{12}, \dots, A_{Nh_N}$ are assigned the respective truth-values $a_{11}, a_{12}, \dots, a_{Nh_N}$, delete from S :

- (a) all systems in which the elementary parts $S_{y_2 y_1}^{y_1 y_2} A_{i\lambda}$ are assigned $a_{i\lambda}$ ($i = 1, \dots, N; \lambda = 1, \dots, h_i$),
- (b) and all systems in which for some $j, 1 \leq j \leq n$, the elementary parts $S_{z_j}^{y_2} A_{i\lambda}$ are assigned $a_{i\lambda}$ ($i = 1, \dots, N; \lambda = 1, \dots, h_i$),
- (c) and all systems in which for some $j, 1 \leq j \leq n$, the elementary parts $S_{y_2 z_j}^{y_1 y_2} A_{i\lambda}$ are assigned $a_{i\lambda}$ ($i = 1, \dots, N; \lambda = 1, \dots, h_i$),
- (d) and all systems in which for some $j, k, 1 \leq j < k \leq n$, the elementary parts $S_{z_j z_k}^{y_1 y_2} A_{i\lambda}$ are assigned $a_{i\lambda}$ ($i = 1, \dots, N; \lambda = 1, \dots, h_i$).

Rule 3 If there is no remaining system in which the elementary parts $F_1(y_1, \dots, y_1), F_2(y_1, \dots, y_1), \dots, F_N(y_1, \dots, y_1)$ have the respective truth values a_1, \dots, a_N and in which $F_1(y_2, \dots, y_2), F_2(y_2, \dots, y_2), \dots, F_N(y_2, \dots, y_2)$ have the truth-values b_1, b_2, \dots, b_N , then form, from S , two sets such that: (1) the first contains all systems of S except those in which the elementary parts $F_1(u, \dots, u), F_2(u, \dots, u), \dots, F_N(u, \dots, u)$ are assigned a_1, \dots, a_N for some individual variable u ; and (2) the second contains all

systems of S except those in which the elementary parts $F_1(t, \dots, t)$, $F_2(t, \dots, t), \dots, F_N(t, \dots, t)$ are assigned b_1, \dots, b_N for some individual variable t .

Simulation of the decision procedure

In discussing simulation of the decision procedure given above, we shall employ the same approach as was used in previous sections of this chapter by establishing a correspondence between sets of falsifying systems and certain system trees of E , and a correspondence between the three rules given above and the DRIs presented below.

We first note that demonstrating (1) is not falsifiable in the decision table is equivalent to demonstrating that the formula obtained by negating (1) and introducing Skolem functions is unsatisfiable. This follows immediately from the fact that the instantiation effected by the decision table treats the variables y_1 and y_2 as if they were universally quantified (which indeed is the case if we are dealing with the negation of (1)) and the variables z_1, \dots, z_n as functionally dependent on y_1 and y_2 . Thus the first $n + 2$ columns of the table, in effect, introduce Skolem functions into the negation of (1). Finally, we note that demonstrating M is falsifiable in this context is equivalent to demonstrating $\neg M$ is satisfiable.

Now suppose that Θ is the initial tree corresponding to the

Pd ⁵ equivalent of (1). Application of the DRI f_{pn} ⁶ to Θ produces an ungrounded tree which contains only literals and is satisfiable if and only if Θ is satisfiable. Furthermore, it is always possible to transform (1) into an equivalent formula (also in Skolem normal form) which has the property that application of f_{pn} to its initial tree produces a saturated tree⁷. We shall thus assume without loss of generality that (1) has this form. The tree resulting from the application of f_{pn} to Θ will be denoted by S_f^* . We note that any branch of S_f^* has the property that each elementary part of M occurs as content of some node of the branch and that the truth-value assignment implied by the branch⁸ causes M to be false. In fact, the nature of E is such that any truth-value assignment, which causes M to be false, will be represented in S_f^* by some branch. Thus S_f^* constitutes the set of falsifying systems for M ⁹. Viewed in this context the rules discussed above (rules 1, 2 and 3), in effect, cause branches to be removed from S_f^* under certain specified conditions. The three DRIs presented below, which correspond to these rules, do exactly this.

⁵It will be recalled that Pd is the subsystem of E that results from embedding the first order predicate calculus in E .

⁶Fully process node. (This DRI was discussed in the introduction to this chapter.)

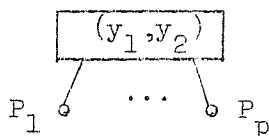
⁷Each branch contains each elementary part in M .

⁸This is determined by assigning the atom A_j^i the truth value T if $\delta_j^i = \textcircled{T}$ ($L_j^i = \delta_j^i A_j^i$) and F if $\delta_j^i = \textcircled{F}$.

⁹We note that given an arbitrary formula of the form (1), a saturated system tree having the property that it is satisfiable

The logic of \bar{E} is such that only branches which are closed may be removed from a system tree. Thus, we can not arbitrarily remove a branch without violating the consistency of \bar{E} . Friedman's rules, however, not only specify under what conditions a branch is to be removed, but also supply exactly enough information as to determine a construction within the framework of \bar{E} which justifies removing the branch within the logic \bar{E} . As an example, consider rule 1.

If rule 1 applies, no remaining falsifying system of S has the properties stated and the rule then specifies that if s_0 is a system such that for some variable u it is true that s_0 prescribes the truth-value assignments $F_1(u, \dots, u) = a_1, \dots, F_N(u, \dots, u) = a_N$, then s_0 may be deleted. Friedman justifies this rule as follows: If we consider a general row of the decision table, it assigns the variables $y_1, y_2, z_1, \dots, z_n$ natural numbers $k_1, k_2, m_1, \dots, m_n$ respectively. Elsewhere in the table there must exist "uniform" rows which assign y_1 and y_2 the numbers $k_1, k_1; k_2, k_2; m_j, m_j$ ($j = 1, \dots, n$). If any system eliminated by this rule were used in the general row, it would be impossible to falsify at least one of these uniform rows. This proof outline may be utilized to motivate removal of the branch of S^* corresponding to S . Suppose that we represent S_f^* schematically as



⁹(continued) if and only if S_f^* is satisfiable may be constructed without first transforming (1). This tree generally will represent the falsifying systems for M in a more compact form than S_f^* .

where P_1, \dots, P_p are the paths of S_f^* which correspond to the falsifying systems of M . (Note that z_1, \dots, z_n do not appear in this notation since the z_i s have been replaced by Skolem functions of the variables y_1, y_2 .) If rule 1 is applicable to the set of falsifying systems of M and if its application causes the falsifying system corresponding to P_1 to be deleted, then the following conditions hold in S_f^* :

There exist truth-values a_1, \dots, a_N and a u such that

1. For $1 \leq i \leq N$ P_1 assigns $F_i(u, \dots, u)$ the truth-value a_i

and

2. For each path P of S_f^*
 - a) there exist an i and j such that a_i is not assigned to A_{ij} ($1 \leq j \leq h_i$)
 - or
 - b) there exists P_1 and B_2 such that

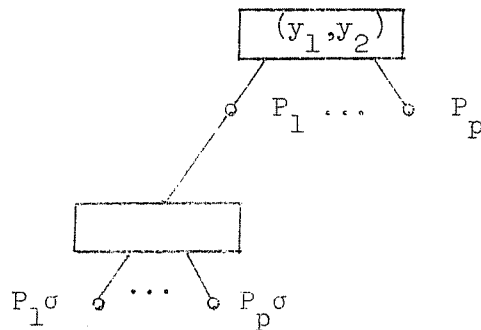
$$B_1\{\langle y_1, y_2 \rangle\} = P_2\{\langle y_1, y_2 \rangle\}$$
 and B_1, B_2 are assigned conjugate truth-values.

Now u is one of the following, $y_1, y_2, \hat{z}_1, \dots, \hat{z}_n$ ¹⁰. We thus define σ as

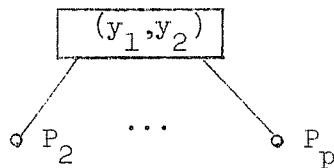
$$\sigma = \begin{cases} \{\langle y_1, y_2 \rangle\} & \text{if } u = y_1 \\ \{\langle y_2, y_1 \rangle\} & \text{if } u = y_2 \\ \{\langle \hat{z}_i, y_1 \rangle, \langle \hat{z}_i, y_2 \rangle\} & \text{if } u = \hat{z}_i \quad (1 \leq i \leq n) \end{cases}$$

¹⁰Here we use \hat{z}_i to denote $f_i(y_1, y_2)$ where f_i is the Skolem function introduced for z_i .

Let t be the result of appending S_f^* to the leaf of S_f^* which determines the branch P_1 . The tree t may be represented schematically as



We shall now demonstrate that all paths of t , which contain P_1 as an initial segment, close. It will then follow that application of multiple closed branch removal to each of these paths produces the system tree



i.e. S_f^* with P_1 removed. Thus, if the falsifying system s_1 corresponding to the path P_1 of S_f^* can be removed by application of rule 1, then P_1 can be removed from S_f^* within the framework of E . Demonstration that P_1 may be removed follows:

Let us partition the paths of S_f^* into two classes C_1 and C_2 . C_1 consists of those paths for which 2a holds and C_2 those for which 2b holds. (Since rule 1 is assumed to apply to S_f^* each path of S_f^* is in one of these classes.) On the path P_1 ,

it is true by hypothesis that $F_i(u, \dots, u)$ is assigned a_i ($1 \leq i \leq N$). Further, for each path there exists i_0, j_0 such that $A_{i_0 j_0}$ is not assigned a_{i_0} . Suppose $u = y_1$. Then, P_1 assigns $F_i(y_1, \dots, y_1)$ the truth-value a_i . In $S_f^* \sigma$, $A_{i_0 j_0} \sigma = F_{i_0}(y_1, \dots, y_1)$ since, by hypothesis, A_{i_j} contains only the individual variables y_1 and y_2 . Now $A_{i_0 j_0}$ and $A_{i_0 j_0} \sigma$ must be assigned the same truth-value (since $P_k \sigma$ is a substitution instance of P_k) and this is not a_{i_0} . Thus, each path containing P_1 as an initial segment and a substitution instance of one of the elements of C_1 as a final segment, assigns $F_{i_0}(y_1, \dots, y_1)$ the truth-values a_{i_0} and \bar{a}_{i_0} . This means that the prefixes of the two occurrences of this elementary part are conjugate and thus the path closes. An identical argument holds for $u = y_2$ or $u = \hat{z}_i$.

Finally, if a path P belongs to C_2 , then there exist B_1, B_2 such that $B_1 \{ \langle y_1, y_2 \rangle \} = B_2 \{ \langle y_1, y_2 \rangle \}$ and B_1, B_2 are assigned conjugate truth-values. But, on the path $P\sigma$ of $S_f^* \sigma$, $B_1 \sigma = B_2 \sigma$ thus $P\sigma$ has conjugate nodes and thus closes.

The above observations lead us to define the DRI rl as follows:

rl Let S^* be a system tree representing some of the remaining falsifying systems of M then

1. Define the applicability test for rl exactly as in Rule 1 (reading "path" for "system").
2. Append $S^* \sigma$ to the path of S^* designated by Rule 1 (σ was defined above).

3. Apply multiple closed branch removal to the paths of the result of step 2 whose final segments occur in $S^* \sigma$.

From the above arguments, we see that if S^* corresponds to some set of Friedman systems S , and application of Rule 1 to S produces S_1 then application of r_1 to S^* produces S_1^* . The DRI r_2 which corresponds to Rule 2 also has this property.

r_2 Let S^* be a system tree representing some of the remaining systems of M then

1. Define the applicability test for r_2 exactly as in Rule 2 (reading "path" for "system").
2. Append $S_f^* \xi \sigma$ to the path of S^* designated by Rule 2 (σ and ξ are defined below).
3. Apply multiple closed branch removal to the paths of the result of step 2 whose final segments occur in $S^* \sigma$.

If Rule 2 subpart a applies then $\sigma = \{ \langle y_2, x_1 \rangle, \langle y_1, x_2 \rangle \}$

and $\xi = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \}$

If Rule 2 subpart b applies then $\sigma = \{ \langle \hat{z}_j, y_2 \rangle \}$ and $\xi = \phi$

If Rule 2 subpart c applies then $\sigma = \{ \langle y_2, x_1 \rangle, \langle \hat{z}_j, x_2 \rangle \}$

and $\xi = \{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \}$

If Rule 2 subpart d applies then $\sigma = \{ \langle \hat{z}_j, y_1 \rangle, \langle \hat{z}_k, y_2 \rangle \}$

and $\xi = \phi$

It can be demonstrated by analysis of the cases a, b, c and d, using arguments similar to those used above, that each path having

a final segment in S^* closes. Thus, as in the case of r_1, r_2 removes any path of S^* whose corresponding falsifying system would be removed by Rule 2.

The DRI r_3 , which simulates Rule 3, produces two output trees S_1^* and S_2^* whose mutual unsatisfiability implies the unsatisfiability of r_3 's single input tree. As in the case of r_1 and r_2 , we begin by specifying those conditions which hold if the DRI is applicable to the input tree.

Suppose that Rule 3 is applicable to a set of falsifying systems S and that S^* is a system tree corresponding to S . Further, suppose that $a_1, \dots, a_N, b_1, \dots, b_N$ are truth-values, $u, t \in \{y_1, y_2, \hat{z}_1, \dots, \hat{z}_n\}$ are specified in Rule 3, and

$$P = \{p: p \text{ is a branch of } S^* \text{ such that for some } u, \\ p \text{ assigns } a_i \text{ to } F_i(u, \dots, u) (1 \leq i \leq N)\}$$

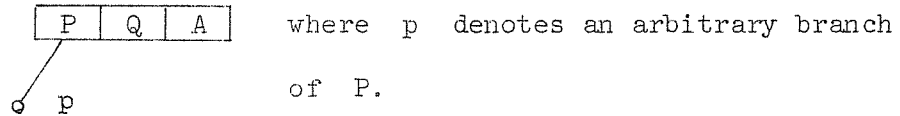
$$Q = \{q: q \text{ is a branch of } S^* \text{ such that for some } t, \\ q \text{ assigns } b_j \text{ to } F_j(t, \dots, t) (1 \leq j \leq N)\}$$

The following condition holds as a result of Rule 3 being applicable to S : For every path r occurring in S^* , either there exists i_0 such that r does not assign a_{i_0} to $F_{i_0}(y_1, \dots, y_1)$ or there exists j_0 such that r does not assign b_{j_0} to $F_{j_0}(y_2, \dots, y_2)$.

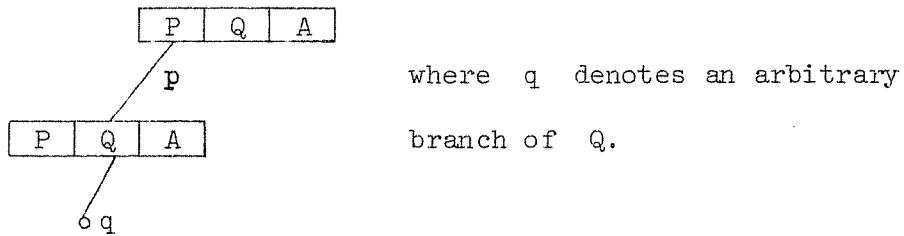
Friedman's Rule 3 produces the two sets $S_1 = S - \bar{Q}$ and

$S_2 = S - \bar{P}$. (\bar{P}, \bar{Q} are sets of falsifying systems corresponding to the branch sets P and Q respectively.) We shall now show how S_1^* and S_2^* may be produced within the framework of E . The following discussion constitutes an implicit specification of r_3 .

Let S^* , P and Q be as above and let A consist of those branches of S^* which are neither in P nor in Q . We shall denote S^* in the following manner:

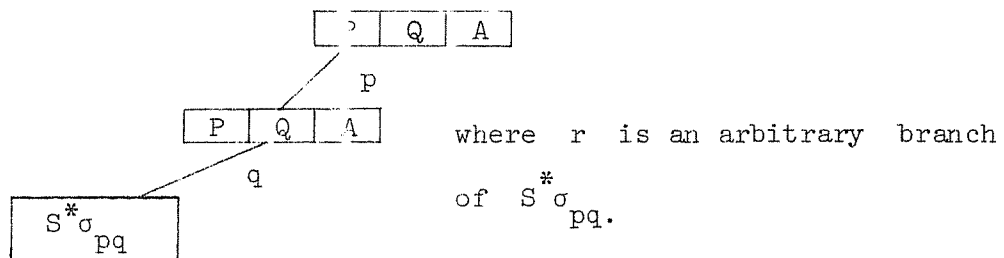


We may append a copy of S^* to p thus obtaining



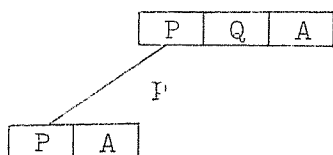
Now define $\sigma_{pq} = \{ \langle u, y_1 \rangle, \langle t, y_2 \rangle \}$ where u and t are determined by p, q and the applicability test for Rule 3. Further, let

S_{pq}^* denote the tree which results from applying σ_{pq} to S^* . For a given p and q (as indicated above) we may then obtain the tree

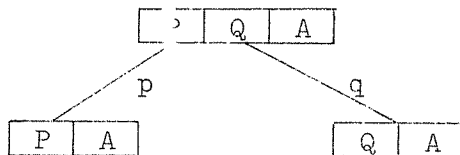


We shall now demonstrate that for given p, q and arbitrary r , the branch $p-q-r$ is closed.

Since r is a substitution instance of some branch d of S^* , and since by hypothesis either there exists i_0 such that d does not assign a_{i_0} to $F_{i_0}(y_1, \dots, y_1)$ or there exists some j_0 such that d does not assign b_{j_0} to $F_{j_0}(y_2, \dots, y_2)$ we have, in particular, for r , either r does not assign a_{i_0} to $F_{i_0}(u, \dots, u)$ or r does not assign b_{j_0} to $F_{j_0}(t, \dots, t)$. Without loss of generality, let us assume the former. We then have by hypothesis (since $p \in P$) that p assigns a_{i_0} to $F_{i_0}(u, \dots, u)$. Thus the path $p-q-r$ assigns $F_{i_0}(u, \dots, u)$ contradictory values. Thus, $p-q-r$ is closed and the branch segments $q - r$ may be removed for the given q and all r . The above procedure may be repeated for each $q \in Q$ finally yielding the tree



In a similar manner, we may obtain the tree

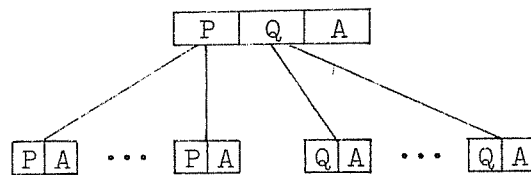


It can be shown that [P | A] and [Q | A] are proper forests for the above tree. We may thus apply O_7 (copy) to these

to produce the output trees $\begin{array}{|c|c|} \hline P & A \\ \hline \end{array}$ and $\begin{array}{|c|c|} \hline Q & A \\ \hline \end{array}$. These correspond exactly to S_1 and S_2 .

We have shown that given an acceptable input tree S^* corresponding to the falsifying systems S , r3 produces output trees S_1^* and S_2^* which correspond exactly to the two falsifying system sets produced by Rule 3. It remains to be shown that the unsatisfiability of S_1^* and S_2^* implies the unsatisfiability of S^* . To this end, it suffices to show that if S_1^* and S_2^* close, then S^* also closes.

We first note that if $\begin{array}{|c|c|} \hline P & A \\ \hline \end{array}$ and $\begin{array}{|c|c|} \hline Q & A \\ \hline \end{array}$ close (i.e. we may derive the empty tree from each) we can delete P and Q from S^* ($= \begin{array}{|c|c|c|} \hline P & Q & A \\ \hline \end{array}$) as follows: To each $p \in P$ and $q \in Q$ append the proper forests $\begin{array}{|c|c|} \hline P & A \\ \hline \end{array}$, $\begin{array}{|c|c|} \hline Q & A \\ \hline \end{array}$ respectively. This produces the tree



Applications of O_8 then produce $\begin{array}{|c|} \hline A \\ \hline \end{array}$. Now $\begin{array}{|c|c|} \hline P & A \\ \hline \end{array}$ unsatisfiable and $\begin{array}{|c|c|} \hline Q & A \\ \hline \end{array}$ unsatisfiable implies $\begin{array}{|c|} \hline A \\ \hline \end{array}$ is closable. Thus, S^* is closable and thus unsatisfiable.

We thus see that if S_f^* is the tree corresponding to the set of falsifying systems for $(\exists y_1)(\exists y_2)(\forall z_1)\dots(\forall z_n)M$ then the rules r1, r2, and r3 may be applied to S_f^* and its descendants in such

a way as to maintain an exact correspondence between the falsifying system sets produced by the Friedman procedure and trees of \bar{E} . If the Friedman procedure produces a proof, then its simulation will derive the empty tree (within the framework of \bar{E}). If the Friedman procedure terminates without producing a proof, its simulation will terminate without producing the empty tree.

CHAPTER 4 CONCLUDING REMARKS

The extendible type-logic E , presented in chapter 2, possesses four features which make it suitable as a basis of a general purpose mathematically oriented question answering system. These features are:

1. It is a higher order logic.
2. It provides for the introduction of defined constants.
3. Specified defined constants, which have been introduced by the user, may be assimilated into the inferential portion of the logic.
4. Provision is made for selective incremental unabbreviation.

We have already observed that higher order logics possess greater expressive power than first order logics and that the ability to deal with defined constants occurring in the input language makes possible concise representation of hierarchical user concepts. The ability to assimilate selected defined constants is a powerful feature which deserves further attention.

The logic E can handle defined constants directly by means of the inference operator O_2 and the tfat's computed for selected defined constants. Thus, a defined constant such as " \equiv " may be processed via the unabbreviation and λ -reduction operators prior to the time a tfat for it has been calculated. However, once a tfat has been computed, unabbreviation of further occurrences of " \equiv " is unnecessary since " \equiv " is then part of the logic in essen-

tially the same way as is the primitive operator " \oplus ". Dependency forests make possible the extension of this sort of assimilation to defined constants involving quantification. We note that whether a given defined constant is assimilated or processed by the unabbreviation and λ -reduction operators, its occurrences within expressions are handled much more directly than if the defined constants were introduced via a series of axioms. Thus, \bar{E} provides a relatively direct means of handling those user-concepts embodied in user-introduced defined constants.

The provision in \bar{E} for selective incremental unabbreviation makes it possible for strategies utilizing \bar{E} to take advantage of strategic information implicit in the abbreviational structure of the input expression. In this paper, we have not considered what form such strategies might take, a problem which is clearly critical if the full power of \bar{E} is to be brought to bear. Determination of such strategies represents one area in which research on \bar{E} might continue.

It would seem that the next logical step in the development of \bar{E} would be to implement a system which embodied the logic of \bar{E} and provided a test-bed for experimentation with various theorem proving strategies, probably a system somewhat along the lines suggested in appendix A. It is felt that such a system would provide the most fruitful approach to the problem of developing powerful adaptive strategies which utilize \bar{E} .

The ability to introduce arbitrary defined rules of inference is also an extremely important feature of such a system, especially since the notion of defined rule of inference is hierarchical and thus makes it possible to build on previous work. A test-bed of the sort mentioned above would provide a framework for investigation in this direction also. As a matter of practical concern, we note the following: The defined rules of inference, considered in chapter 3, mimicked certain inferential operations of the simulated system by means of sequences of \bar{E} -inference operators. In a practical system, such constructions could be considered to constitute meta-theoretical justification for the particular end result and implemented in a more direct manner. For example, the constructions used to simulate Friedman's rules, which primarily justify the removal of certain branches, could be implemented simply by removing the specified branch once it was known that the construction was possible.

From a theoretical point of view, the ability to simulate various proof procedures in \bar{E} places such procedures in a common framework thus raising not only the possibility of mixing parts of these strategies within a larger procedure, but also the possibility of comparing simulated strategies. To what extent such comparisons may prove fruitful in suggesting extensions or modification of current strategies depends on one's intuitive feeling for the underlying operations of \bar{E} . Whether such intuition is more easily acquired in a system such as \bar{E} or in one of the systems considered in chapter 3, remains to be determined. None the less, it is interesting that strategies such

as Friedman's and Robinson's, which at least on the surface are quite different, can be expressed in the same system.

We have taken the view, in this paper, that any logical system, which is to serve as the basis of a general purpose mathematically oriented question answering system, must have the capability to adapt to arbitrary user-specified theories. The features included in E provide it with a modest adaptive capability since the set of defined constants introduced by the user, which reflect the state of development of his theory, may be assimilated into the inferential portion of E . Furthermore, selective incremental unabbreviation makes possible the writing of strategies which are sensitive to the abbreviational structure of user-supplied expressions. Since these presumably have structural significance relative to the user's theory, this capability permits the writing of strategies which are sensitive to that structure.

The current modular approach to mechanical theorem prover writing provides no integrated framework in which user-theory-sensitive strategies may be produced. The logic system proposed in this paper does. It, therefore represents an interesting alternative to the current modular approach, while at the same time providing a higher-order refutation logic for mechanical theorem prover strategy experimentation.

APPENDIX A Implementation of defined rules of inference1.0 Introduction

In this appendix, we sketch a system called DELS¹ in which defined rules of inference might be implemented. The presentation is not intended as a complete specification but rather as an outline of the general structure of such a system. Sufficient detail is included as to allow the reader to have some feeling for the size and complexity of some of the defined rules of inference discussed in chapter 3. In particular, only those primitive functions required to specify example defined rules of inference are detailed.

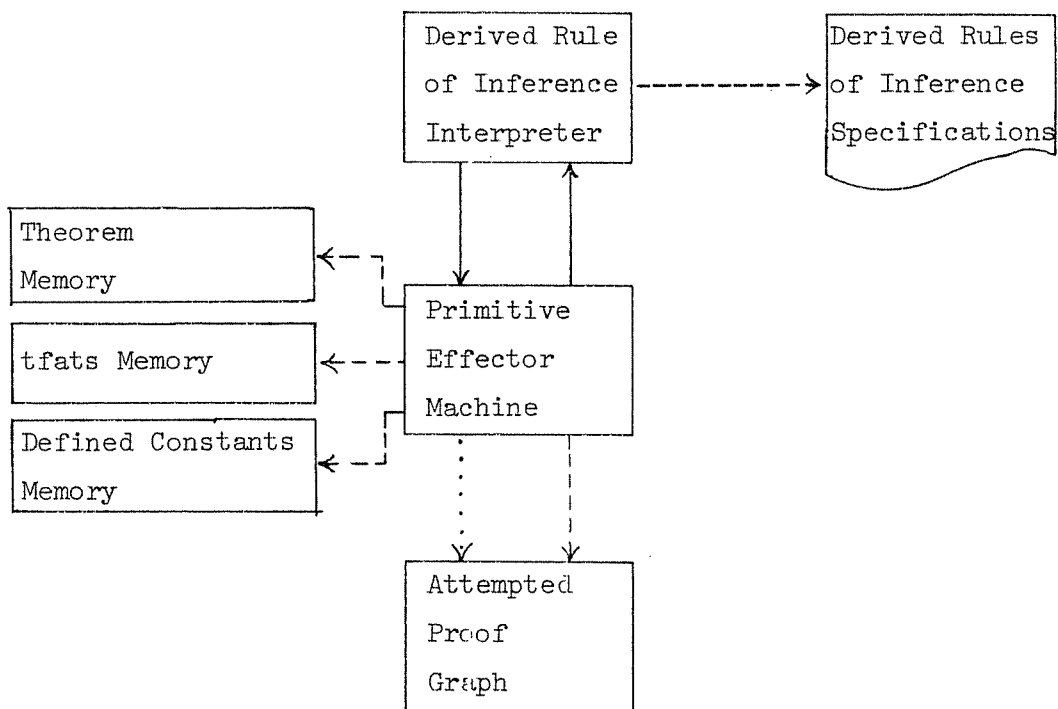
DELS is essentially a problem oriented language embedded in LISP [23] consisting of certain LISP representable operations and data structures which are designated as DELS primitives. Within this context, defined rules of inference are just partial functions² written in the LISP meta language³ which are constructed in terms of DELS primitives and which satisfy the requirements set forth in chapter 3.

The overall operation of DELS is depicted in figure 1.

¹Definitionally Extendible Logic System.

²See McCarthy ... [22].

³In certain cases, infix rather than prefix notation is employed. Furthermore, M expressions are formed over an extended set of symbols.



A deposits results in B $A \cdots \rightarrow B$

A references B $A \text{---} \rightarrow B$

Control passes $A \longrightarrow B$

Included among the primitive operators of DELS are the operators D_1, D_2, \dots, D_{12} which implement the E -inference operators O_1, O_2, \dots, O_{12} . These may be viewed as comprising the command repertoire of a machine called the primitive effector machine which accepts commands from the defined rule of inference interpreter⁴. checks

⁴That portion of the LISP interpreter required for execution of defined rule of inference specifications.

these commands for validity and, if valid, executes them.

A command issued by the defined rule of inference interpreter is a command to apply some operator to a set of designated parameters and is valid if 1) it involves one of the operators recognized by the machine (D_1, D_2, \dots, D_{12}) and 2) the parameters supplied with the operator meet the requirements of the particular operator. Execution of a valid command causes a tree to be added to the attempted proof graph. Invalid commands are not executed. In either case, control passes back to the defined rule of inference interpreter.

In determining if the parameters supplied as part of a command are acceptable, the primitive effector machine may reference one or more memory structures. All operators reference the attempted proof graph since, all operators apply to parameters which explicitly or implicitly reference system trees contained in the attempted proof graph. In addition, D_9 (restricted append) requires that patriarch information associated with the input trees be available. (This information is assumed to be retrievable from the attempted proof graph.) Certain operators reference other memory structures in addition to the attempted proof graph. For example, D_{11} (unabbreviation) references the defined constant memory which contains all defined constants and their definitions. D_2 references the memory which contains tfat's (tfat memory) both to determine if a tfat for the major connective referenced by D_2 exists, and for the purpose of constructing the output tree. Finally, D_{12} (theorem addition)

references previously proved theorems which are contained in the theorem memory. Excepting the attempted proof graph, the structure of each of these memories is left unspecified since such information is not required to understand the material presented in this appendix.

2.0 Data objects referenced by defined rule of inference specifications

Defined rules of inference specifications are written in the LISP meta language utilizing primitives of DELS and reference various objects which are peculiar to DELS. In this section we shall indicate the nature of these objects and describe how they may be referenced in a defined rule of inference specification. Such references may either specify an object which already is part of the DELS data base (for example, a tree contained in the attempted proof graph) or an object which will be part of the DELS data base as a result of the primitive effector machine executing valid commands. (For example, the tree which will be in the attempted proof graph after D_2 is applied to some pair of acceptable parameters.)

DELS contains the following categories of data objects:

1. Attempted proof graph
2. Ordered sets
3. Substitutions
4. Trees
 - a) Nodes
 - b) Leaves
 - c) Branch-elements

Attempted proof graph This is essentially the structure discussed in chapter 2 augmented to allow retrieval of patriarch information. (Another slight modification is discussed later in this section.) We assume a set of interrogation primitives which are at least sufficient to retrieve all information stored in the attempted proof graph. These would include a node content function, immediate successors function, and so forth. The attempted proof graph is accessible only through these primitives.

Ordered sets Ordered sets, which are essentially lists in LISP, provide a convenient way of representing small finite sets of objects which must be enumerated in the course of a computation. An ordered set is simply an n-tuple $\langle d_1, \dots, d_n \rangle$ whose elements are data objects of categories 2, 3, and 4. Furthermore, all data objects occurring in a given ordered set are of the same category. (We note, however, that subcategories a, b, and c may be mixed in the sense that data objects of category 4a, b, and c may occur in a given ordered set.)

Initially, the only ordered set in the DELS data base is the empty set ϕ . Ordered sets may be synthesized from ϕ and other data objects by means of the operator \oplus . (In LISP, \oplus has the same properties as cons.) Thus, if d is a data object of category 2, 3, or 4 and x is either ϕ or an ordered set comprised of data objects of the same category as d and of the form $\langle x_1, \dots, x_n \rangle$, then $d \oplus x$ is either the ordered set $\langle d \rangle$ or the ordered set $\langle d, x_1, \dots, x_n \rangle$. We note, in particular, that if $d = \langle d_1, \dots, d_n \rangle$ is an ordered set, then $d = d_1 \oplus (d_2 \oplus (\dots (d_n \oplus \phi) \dots))$.

Ordered sets may be analyzed using the operators α and ω . (In LISP, these have the same properties as `car` and `cdr` respectively.) If $y = d \oplus x$ where d is a data element and y an ordered set, then $\alpha[y] = d$ and $\omega[y] = x$.

Substitutions Substitutions were discussed in chapter 2. Initially, the DELS data base contains only the empty substitution ϕ . New substitutions are created by the functions "unify" (see appendix B) and composition, (denoted by the infix operator *).

Trees Initially the only tree contained in the DELS data base is the initial tree θ . This is part of the initial attempted proof graph as mentioned previously. New trees are created and added to the attempted proof graph as the result of executing primitive effector operations which explicitly or implicitly reference existing trees. New nodes, leaves and branch-elements (i.e. all nodes of a designated branch) are created at the time the parent tree is created. It is assumed that given a node, it is possible to determine the tree in which the node occurs.

The primitive effector operators D_1, D_2, \dots, D_{12} while executed primarily for their side effects (i.e. they create trees and add them to the attempted proof graph), do return a value. This value, which specifies the tree created by the operator, is an ordered set consisting of the names of nodes comprising the tree ordered by some arbitrary but fixed rule⁵. In general, an ordered

⁵One such rule might be:

set consisting of the names of nodes comprising a given tree and ordered by this fixed rule will be called a tree specification.

It is assumed that DELS contains a set of primitive tree-interrogation functions which allow retrieval of all information contained in any ungrounded tree. (This includes the capability of analyzing the content of any node.) These primitives include among others, the following:

Let n name a node of the tree t

1. immediate-successors [n] produces as its value an ordered set of immediate successors of the referenced node ordered according to the ordering of the ordered tree.
2. content [n] produces as its value the content of the referenced node.
3. parent-tree-specification [n] produces as its value the ordered set which specifies t .
4. atomic [n] produces as its value T or F depending upon whether or not the suffix of the content of n is atomic.

⁵(continued)

1. The origin is the first node
2. If $n < k$ then n precedes k
3. If n_1 and n_2 are immediate successors of n and if the tree ordering function specifies n_1 precedes n_2 , then all successors of n_1 precede all successors of n_2 .

In particular, whatever rule is used for ordering the nodes of a tree, it is important that a given tree always be specified by the same ordered set.

DELS also contains three functions which operate on tree specifications (i.e. ordered sets which specify trees) and produce ordered sets representing node subsets of the input tree. These functions are leaves, branch-elements and new-nodes.

Suppose t is a tree specification and k a leaf of the tree specified by t , then

leaves $[t]$ produces an ordered set consisting of the leaves of t .

(The ordering is arbitrary but fixed and might, for example, be obtained from the ordering of the ordered set t .)

branch-elements $[t,k]$ produces an ordered set consisting of all

nodes of t on the branch determined by the leaf k .

(The ordering is arbitrary but fixed and might, for example, be obtained from the immediate successors function.)

new-nodes $[t]$ Suppose t is produced as the result of executing some DRI which involves n applications of the operator D_2 , then new-nodes $[t]$ produces as its value the ordered set consisting of the images⁶ in t of all nodes (having content) which occur in t 's associated with the n applications of D_2 . In the simple case where the DRI

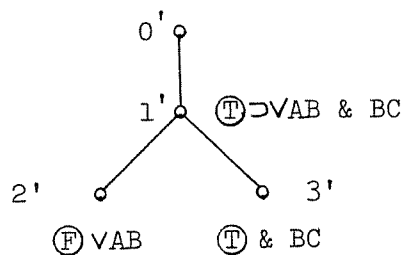
⁶This information can be computed at the time the DRI in question is executed. At the end of the computation, it is assumed to be stored (in some form) with t in the attempted proof graph.

is just D_2 , $\text{new-nodes}[t]$ produces an ordered set of those nodes of t which are images of nodes (having content) which occur in the t fat associated with D_2 .

example 1 Let θ be

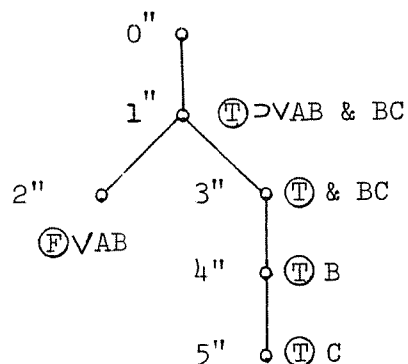


θ is then specified by $\langle 0,1 \rangle$. If D_2 is applied to θ we obtain the tree t_1



which may be specified by $\langle 0',1',2',3' \rangle$. We note in particular that $\langle 0',1',2',3' \rangle = D_2[1,1]$.

If D_2 is applied to $3'$ we obtain the tree t_2



which may be specified by $\langle 0'',1'',2'',3'',4'',5'' \rangle = D_2[3',3']$

The leaves of t_1 and t_2 may be obtained by applying "leaves" to the specifications of t_1 and t_2 , thus

leaves [$\langle 0', 1', 2', 3' \rangle$] = $\langle 3', 2' \rangle$ (note here we have assumed orderings opposite that given in the diagrams for t_1 and t_2)
 leaves [$\langle 0'', 1'', 2'', 3'', 4'', 5'' \rangle$] = $\langle 5'', 2'' \rangle$

The calculation of t_2 from θ may thus be specified as follows:

(1) $D_2[p, p]$ where $p = \alpha[\text{leaves}[D_2[1, 1]]]$

i.e. $p = \alpha[\text{leaves}[\langle 0', 1', 2', 3' \rangle]] = \alpha[\langle 3', 2' \rangle] = 3'$

thus $D_2[p, p] = D_2[3', 3'] = \langle 0'', 1'', 2'', 3'', 4'', 5'' \rangle$.

If (1) is designated as DRI^7 , then:

new-nodes [$D_2[p, p]$] = $\langle 2'', 3'', 4'', 5'' \rangle$ (note that this ordered set does not constitute a tree specification)

Finally, branch-elements [$D_2[p, p], \alpha[\text{leaves}[D_2[p, p]]]$]
 = branch-elements [$D_2[p, p], \alpha[\langle 5'', 2'' \rangle]$]
 = branch-elements [$D_2[p, p], 5''$] = $\langle 5'', 4'', 3'', 1'', 0'' \rangle$.

We note that in the above examples, references to data objects may be explicit (i.e. the node $3'$) or implicit (i.e. $\alpha[\text{leaves}[D_2[1, 1]]]$). However, prior to the computation of $\alpha[\text{leaves}[D_2[1, 1]]]$, the node named by $3'$ is not in the data base.

⁷This causes newly introduced nodes to be accumulated as the primitive effector operations comprising the DRI are executed.

Image sets The image of a node n in a descendant tree of the tree containing n is defined in appendix C. DELS contains a primitive function im which produces as its value the name of a node in the descendant tree which is the image of the given node. As an illustration, consider the previous example. We have,

$$\text{im}[1; \langle 0", 1", 2", 3", 4", 5" \rangle] = 1"$$

It is convenient to have a function which produces all images of the nodes comprising some subset of all the nodes of a given tree. For example, the images of the leaves of t_1 in t_2 . This function may be defined in terms of "im" as follows:

$$\text{Im}[x;s] = \underbrace{[\text{null}[x] \rightarrow \phi; T \rightarrow \text{im}[\alpha[x];s] \oplus \text{Im}[\omega[x];s]]}_{\text{body}}$$

The recursively defined function "Im" takes as parameters an ordered set of nodes (x) occurring in some tree t and a specification s of a tree which is a descendant of t . The body of the definition of "Im" is a conditional which specifies that the value of $\text{Im}[x;s]$ is ϕ if x is empty and $\text{im}[\alpha[x];s] \oplus \text{Im}[\omega[x];s]$ otherwise. $\text{Im}[x;s]$ is computed by enumerating the nodes of x , calculating their images in s ($\text{im}[\alpha[x];s]$) and forming an ordered set of the images ($\text{im}[\alpha[x];s] \oplus \text{Im}[\omega[x];s]$). The termination condition $\text{null}[x]$ ensures that all nodes of the original set x have been processed. We note that if $\text{Im}[x;s]$ and s are regarded as sets, then $\text{Im}[x;s] \subseteq s$.

example 2 Referring to example 1, we obtain

$$\text{Im}[\text{leaves}[D_2[1,1]]; D_2[p,p]] = \langle 3, 2 \rangle$$

3.0 Specification of selected defined rules of inference

We are now in a position to specify some of the defined rules of inference presented in chapter 3. The syntax used is essentially that of the LISP meta language augmented by symbols such as im , Im , α , ω , \oplus etc. In particular we take $\{n\}$ to mean $n \oplus \phi$ and use infix rather than prefix notation where such notation enhances readability.

The defined rule of inference fpn (fully process node) utilizes the function lsuc (leaf successors) which can be defined in terms of DELS primitives. The function "lsuc" has the following characteristics:

If n names a node of the tree t , then

$$\text{lsuc}[n] = \begin{cases} \{n\} & \text{if } n \text{ is a leaf.} \\ \langle k_1, \dots, k_m \rangle & \text{where } k_i \text{ are all leaves dominated by } n \text{ in } t. \end{cases}$$

fpn Let n be a node then

$\text{fpn}[n] = [\text{atomic}[n] \rightarrow \text{parent-tree-specification}[n];$

$T \rightarrow \text{extl}[n; \text{lsuc}[\{n\}]]]$

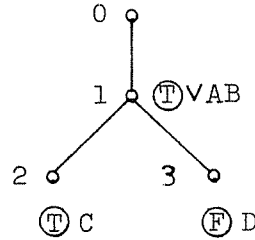
If the suffix of the spe contained at the node n is an atom, fpn terminates with value an ordered set specifying the input tree. Otherwise, $\text{fpn}[n]$ is calculated using the auxiliary functions extl and ext .

$$\text{extl}[n;x]=D_4[\text{im}[n;\text{ext}[n;x]]]^8$$

$$\text{ext}[n;x]=[\text{null}[\omega[x]] \rightarrow D_2[n;\alpha[x]]; T \rightarrow D_2[\text{im}[n;p];\text{im}[\alpha[x];p]]]$$

where $p = \text{ext}[n;\omega[x]]$

example Let the input tree be



$$\text{fpn}[1] = \text{extl}[1, \text{lsuc}[\langle 1 \rangle]] \text{ since } \text{atomic}[1] = F$$

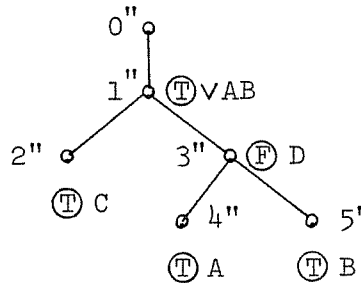
$$\text{lsuc}[\langle 1 \rangle] = \langle 2, 3 \rangle \text{ thus } \text{fpn}[1] = \text{extl}[1, \langle 2, 3 \rangle]$$

$$\text{extl}[1, \langle 2, 3 \rangle] = D_4[\text{im}[1; \text{ext}[1; \langle 2, 3 \rangle]]]$$

$$\text{ext}[1; \langle 2, 3 \rangle] = D_2[\text{im}[1; \text{ext}[1, \langle 3 \rangle]], \text{im}[2; \text{ext}[1, \langle 3 \rangle]]]$$

$$\text{ext}[1, \langle 3 \rangle] = D_2[1; 3]$$

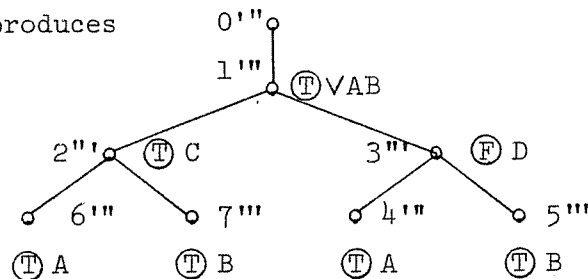
D_2 applied to (1,3) produces



$$\therefore \text{ext}[1, \langle 3 \rangle] = D_2[1; 3] = \langle 0'', 1'', 2'', 3'', 4'', 5'' \rangle \text{ and } \text{im}[1; \text{ext}[1, \langle 3 \rangle]] = 1'' \text{ sim. for } 2$$

$$\therefore \text{ext}[1; \langle 2, 3 \rangle] = D_2[1''; 2'']$$

D_2 applied to (1'',2'') produces

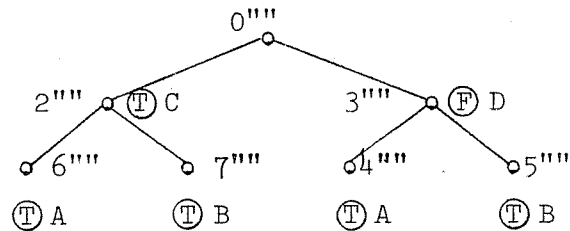


⁸ D_4 is the primitive effector operation - multiple closed branch removal.

thus, $\text{ext}[1; \langle 2, 3 \rangle] = D_2[1'', 2''] = \langle 0''', 1''', 2''', 3''', 4''', 5''', 6''', 7''' \rangle$

and $\text{im}[1; \text{ext}[1; \langle 2, 3 \rangle]] = 1''''$

$\text{extl}[1; \langle 2, 3 \rangle] = D_4[1'''']$ which produces the tree



thus, $\text{fpn}[1] = \text{extl}[1; \langle 2, 3 \rangle] = D_4[1''''] = \langle 0''', 2''', \dots, 7'''' \rangle$

The defined rule of inference fep (fully expand propositional), discussed in chapter 3, may be specified in terms of fpn as follows:

$$\text{fep}[x] = [\text{atomic}[\alpha[x]] \& \text{null}[\omega[x]] \rightarrow p;$$

$$T \rightarrow \text{fep}[\text{union}[\text{new-nodes}[p]; \text{Im}[\omega[x]; p]]]$$

where $p = \text{fpn}[\alpha[x]]$ and $\text{union}[\langle x_1, \dots, x_n \rangle; y] = x_1 \oplus (\dots \oplus (x_n \oplus y) \dots)$

The defined rule of inference "fep" repeatedly applies "fpn" to nonatomic nodes and their descendants until only atomic nodes remain. The value of "fep" is a tree specification (ordered set) of the tree described under fep in chapter 3.

As a final example of a defined rule of inference specification in DELS, we have "prawitz." If we allow θ to be a specification of an initial tree, we have

prawitz [0] = prawitzl[s,s] where s = expand[determine-clauses[$\alpha\omega[0]$]]

prawitzl [c;s] = [null[closeall[c]] \rightarrow T;

T \rightarrow prawitzl[sa[D₃[s; changevar[varp[s]; $\lambda[[x];[v \in V\text{-varp}[c]]],c];s]]$

closeall [c] = [null[a] \rightarrow c; T \rightarrow D₄[leaves[D₃[c;a]]]]

where a = close-sub[c;leaves[c]]

close-sub [s;z] = [null[z] \rightarrow ϕ ; T \rightarrow ex[s; $\omega[z]$; unifysp[$\alpha[z]$]]

ex [s;l;x] = [null[x] \rightarrow ϕ ; null [p] \rightarrow ex[s;l;cdr[x]; T \rightarrow σ *p]

where σ = subst-part[x]

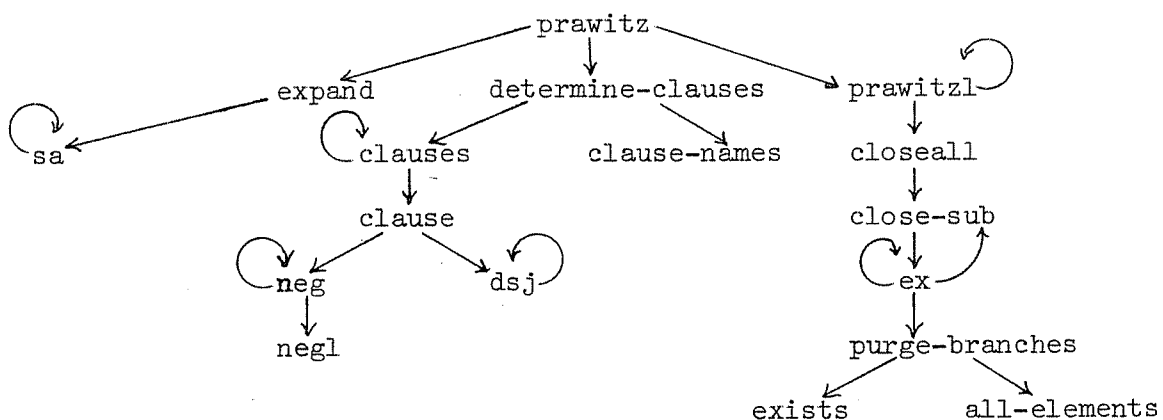
t = D₃[s; σ]

p = close-sub[t;purge-branches[l; node-set-part[x];t;s]]

purge-branches[e;r;t;s]+Im[all-elements[e; $\lambda[[l];[exists[r;\lambda[[a];[$

a \in branch-elements[s;l]]]]]]];t]


The function "prawitz" is the top level function. The remaining functions are auxiliary. Closely related auxiliary functions are given above. Others are given in appendix G. Figure 1 depicts the calling structure of prawitz. (a \rightarrow b means that function b occurs in the definition of function a.)



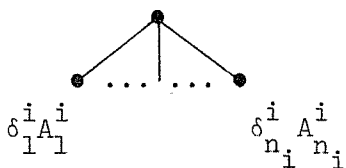
Note: D₃ is the tree substitution operation and D₄ is multiple closed branch removal.

Description of "prawitz[0]

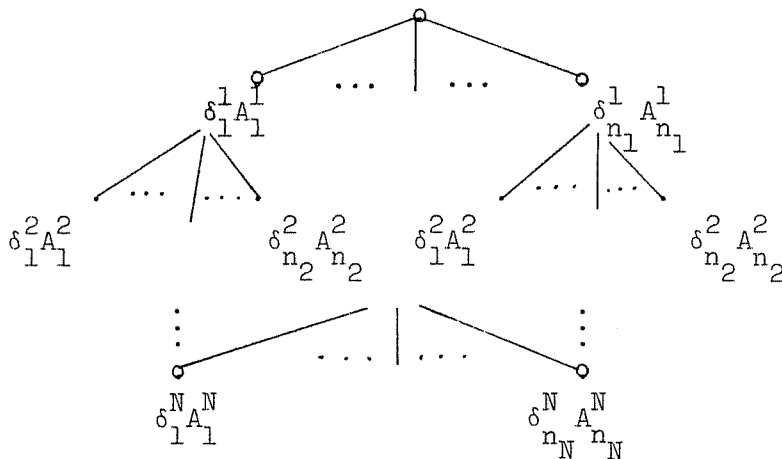
The function prawitz takes an initial tree "0" of the form



as argument and produces the value "T" if $\bigoplus S_k$ is unsatisfiable for some k , "F" otherwise. The top level function prawitz serves mainly to initialize the strategy by effecting the computation of T_0 . In fact, determine-clauses[$\alpha\omega[0]$] just produces the representation



for the clause $L_1^i \vee \dots \vee L_{n_i}^i$, ($1 \leq i \leq N$) where $\delta_k^i = \bigoplus$ if $L_k^i = A_k^i$ and $\delta_k^i = \bigopl�$ if $L_k^i = \neg A_k^i$ so that the value of s computed for $\text{expand}[\text{determine-clauses}[\alpha\omega[0]]]$ is just the DNF representation of $M(x_1, \dots, x_n)$



which is just T_0

Prawitzl determines if the current tree ($=T_k$) can be closed ($\text{null}[\text{closeall}[c]]$) and if it can, returns "T" otherwise T_{k+1} is computed and prawitzl called recursively. The expression

$$D_3[s;\text{changevar}[\text{varp}[s]; \lambda[[x]; [v \in V-\text{varp}[c]]]]]$$

produces a variant of the original tree ($s = T_0$) by computing a substitution $\sigma = \{\dots, \langle e, v \rangle, \dots\}$ such that v is a universally quantified variable in $M(x_1, \dots, x_n)$ (+variable) and e is a variable in the set of system variables V but which does not occur in the current tree T_k . If the original formula which produced the initial tree Θ is valid, then "prawitzl" will eventually terminate.

Closeall[c] returns the tree $c (= T_k)$ if T_k can not be closed by substitution, otherwise closeall returns the empty tree which results from applying D_4 to the closed tree $T_k \sigma$. (Here σ is the closing substitution.)

Close-sub[c;leaves[c]] in conjunction with its auxiliary function "ex" produces ϕ if T_k is not closeable and the closing substitution if it is closeable. As part of the computation of a closing substitution, "purge-branches" removes from consideration those remaining branches which contain neither current contradictory pair.

APPENDIX B Unification related functions

Given an arbitrary pair of type 1 expressions containing Skolem functions, it is possible to determine if the pair is unifiable and if unifiable, what the most general unifier is. In particular, it is possible to specify a function called "unify" which given as arguments the suffixes e_1 and e_2 of two spes, produces either the most general unifier or the value ϕ .

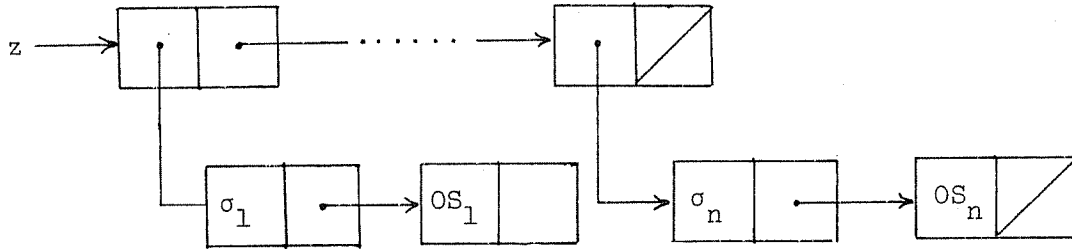
$$\text{unify } [e_1, e_2] = \begin{cases} \sigma \text{ (= the most general unifier) If } e_1 \text{ and } e_2 \\ \text{are unifiable} \\ \phi \text{ If } e_1 \text{ and } e_2 \text{ are not unifiable} \end{cases}$$

The function "unify" applies to a pair of type 1 expressions. It is useful to be able to determine, given two sets of type 1 expressions, all unifiable pairs¹ and associated unifying substitutions. For example, we may partition the set of atomic nodes occurring on a given branch into those with prefix $\textcircled{\text{T}}(Q_{\text{T}})$ and those with prefix $\textcircled{\text{F}}(Q_{\text{F}})$. Any substitution which unifies a pair of suffixes (one chosen from a node occurring in Q_{T} and the other from Q_{F}) will be a closing substitution for the branch (i.e. applying the tree substitution (D_3) with the unifying substitution will produce a tree in which the image of the branch in question is closed).

We may define a function "unify sp" (unify set pairwise) in terms of the primitive "unify" which has the following characteristics:

¹One component chosen from each of the input sets.

Let k be a leaf, then $\text{unifysp}[k]$ is the list structure



where OS_i is an ordered set of nodes occurring on the branch determined by the leaf k . Further, if $OS_i = \langle n_1, m_1, \dots, n_p, m_p \rangle$ then the contents of n_j, m_j have conjugate prefixes and unifiable suffices. σ_i is the most general unifier.

Associated with "unify sp" we have the following functions:

$$\text{substpart}[z] = \text{caar}[z] = \sigma_1$$

$$\text{nodesetpart}[z] = \text{cadar}[z] = OS_1$$

These are introduced mainly as a mnemonic device.

APPENDIX C Image of a node in a descendant tree

Any tree, produced by primitive effector operations, may be thought of as resulting from operations applied directly to an isomorphic image of the input tree (s) (i.e. to a copy of the input tree(s)). For example, the tree produced by one of the node removing operations (say D_4) applied to the tree t can be obtained by copying t and then removing prescribed branches. Each of the operators D_1, \dots, D_{12} may be categorized according to whether or not it increases, decreases or leaves unchanged the number of nodes in the isomorphic image(s) of the input tree(s). (This categorization is given below.) If a given operator either leaves unchanged or increases the number of nodes in the isomorphic image of the input tree, then each node of the input tree has an isomorphic image in the output tree. If the operator decreases the number of nodes, then a prescribed subset of nodes in the input tree (prescribed subsets of nodes in the input trees) has (have) an isomorphic image in the output tree.

Let Ψ be an isomorphism between a given input tree and its copy. (In the case where there are two input trees then Ψ is an isomorphism between the first and its copy and the second and its copy.) The image of a node of an input tree in the output tree is defined as follows:

category 1 increase or leave unchanged number of nodes

If the output tree is obtained by applying one of the operators $\{D_1, D_2, D_3, D_{11}\}$ then for any node n occurring in the input tree, $\text{image}[n] = \Psi(n)$

category 2 decrease number of nodes

If the output tree is obtained by applying one of the operators $\{D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{12}\}$ then the image of any node n occurring in the subset(s) prescribed below is $\Psi(n)$.

<u>operator</u>	<u>prescribed subset</u>
1. D_7	all nodes of the proper subforest
2. $D_4, D_5, D_6, D_8, D_{10}$	all nodes n such that $\Psi(n)$ is not removed
3. D_9	all nodes except the origin of the appended tree
4. D_{12}	all nodes except the origin of the appended tree and nodes n such that $\Psi(n)$ is removed.

The function "image" is thus a partial function from the set of nodes of an input tree (input trees) into the set of nodes of the output tree.

An output tree is a trivial example of a descendant of the given input trees. It is possible to extend the notion of image to cover arbitrary descendants of given sets of input trees. This is accomplished as follows:

We call a tree t a linear descendant of the tree s if there exist trees t_0, \dots, t_n such that $t_0 = s$, $t_n = t$ and t_i is obtained from t_{i-1} by application of one of the operators $D_1, D_2, D_7, D_3, D_4, D_5, D_6, D_8, D_{10}, D_{11}, D_{12}$ ($1 \leq i \leq n$). (We allow the special case where $n = 0$.)

The tree t is a descendant of trees t_1, \dots, t_n if either it is a linear descendant of one of these trees or there exist trees s_1, s_2 such that s_1 and s_2 are descendants of t_1, \dots, t_n and t is

obtained from s_1 and s_2 by application of D_9 .

If t is a descendant of a set of trees T , then there always exists a minimal subset $T_1 \subseteq T$ such that t is a descendant of the trees comprising T_1 . Any time we say that t is a descendant of the trees of T , we shall assume that T is minimal.

Let t be a descendant of $\{t_1, \dots, t_k\}$. We define the image of a node n of any t_i in t $im[n;t]$ in terms of the partial function "image" given above as follows:

Suppose that we encounter the trees $s_0 = t_i, s_1, \dots, s_{m-1}, s_m = t$ as we traverse the attempted proof graph from t_i to t . Let $image[n] = n_1$ and $image[n_j] = n_{j+1}$ (We note that if n_j is defined for $1 \leq j \leq m$ then n_j occurs in s_j .) If n_j is defined for $1 \leq j \leq m$ then $im[n;t] = n_m$. If there exists a j $1 \leq j \leq m$ for which n_j is not defined, then $im[n;t]$ is not defined.

APPENDIX D Definition of the function ζ

For each ungrounded tree t occurring as part of an attempted proof, we define the function ζ_t constructively in terms of the function(s) ζ_{t_1} (and ζ_{t_2}) for the operand tree(s) t_1 (and t_2) and operator used to generate t . Thus, ζ_t ultimately depends on the function ζ_θ which is defined as follows:

Let δe be the initial spe for θ with associated dpf F . The construction of F and δe from the initial polarized expression produces a correspondence between the pseudo atoms of e and the leaves of F . Suppose n is any node of θ other than the origin and that $C_\theta(n) = \delta_n e_n$ (where δ_n is either \textcircled{T} or \textcircled{F}). If e_n is a pseudo atom of e , then $\zeta_\theta(n) = m$: where m is the leaf of F which corresponds to this pseudo atom of e . The function ζ_θ is undefined for all other nodes of θ .

Suppose the ungrounded tree s is produced by applying an operator to the tree t . The function ζ_s is constructed from ζ_t as follows:

1. operators which remove nodes but leave the dpf unchanged

$$(0_4, 0_5, 0_6, 0_8, 0_{10}).$$

If k is any node of s corresponding to a node n of t such that $\zeta_t(n) = m$, then $\zeta_s(k) = m$. The function ζ_s is undefined for all other nodes of s .

2. tree substitution O_3

Since $s = t\sigma$ and $\text{dpf}(s) = \text{dpf}(t)\sigma$, there exist natural correspondences between s , t and between $\text{dpf}(s)$, $\text{dpf}(t)$. In particular, if m is a leaf of $\text{dpf}(t)$ corresponding to the leaf ℓ of $\text{dpf}(s)$, then we shall denote ℓ by m^* .

Let k be any node of s corresponding to the node n of t and suppose $\zeta_t(n) = m$, then $\zeta_s(k) = m^*$. The function ζ_s is undefined for all other nodes of s .

3. proper subforest copy O_7

In this case each leaf of $\text{dpf}(s)$ corresponds to a unique leaf of $\text{dpf}(t)$. In addition, each node k of s (other than the origin) corresponds to a unique node n of t . Furthermore, if $\zeta_t(n) = m$, then m will correspond to a leaf of $\text{dpf}(s)$. Thus, if k corresponds to n and $\zeta_t(n) = m$, we define $\zeta_s(k)$ to be m^* . The function ζ_s is undefined for all other nodes of s .

4. λ -reduction O_1

If case 2 holds or if d_1 as calculated in step 1.1.1 is a pseudo atom (see definition of O_1 page), then $\text{dpf}(s)$ differs from $\text{dpf}(t)$ only in that a subexpression of the context of some leaf of $\text{dpf}(t)$ is replaced with another subexpression. In this case the nodes of t and s correspond and the leaves of $\text{dpf}(s)$ and $\text{dpf}(t)$ correspond. Thus, if the node k of s corresponds to the node n of t and $\zeta_t(n) = m$, then $\zeta_s(k) = m^*$.

If d_1 is not a pseudo atom, then $\zeta_s(k)$ is defined (as above) for all k which do not correspond to a node n such that $\zeta_t(n)$ is a leaf effected by the application of O_1 to t .

5. unit unabbreviation O_{11}

The function ζ_s is defined in the same manner as for O_1 .

6. the operator O_2

The hereditary notion of linear descendant, which relates two nodes one having content whose suffix is a subexpression of the content of the other, may be generalized to nodes occurring in trees which are derivable from others under the full set of E -inference operators. If n is a node of the ungrounded tree t and m a node of the tree s such that $t \in \Gamma$, $\Gamma \Rightarrow s$ and such that n and m are related by this generalized notion of linear descendancy, we shall say that m is a logical descendant of n and n a logical ancestor of m .

The function ζ will be described for an unextended application of O_2 only (i.e. application of O_2 to a node whose content has the form $\delta \oplus e_1 e_2$). This is sufficient to define ζ for extended application of O_2 since the algorithm for extended O_2 application (i.e. to nodes whose content are maximal abbreviations) may be regarded as specifying a sequence of E -inference operator applications not involving extended O_2 .

Let the node n contained in the ungrounded tree t be an ancestor of the node m contained in the tree s . We define the

distance between n and m to be the number of operators involved in the derivation of s from t .

Suppose s is obtained by applying O_2 to a node m of the tree t and suppose $C_t(m) = \delta \textcircled{1} e_1 e_2$ where δ is either \textcircled{T} or \textcircled{F} . Let n be the nearest logical ancestor of m having the property that O_1 or O_{11} is applied to n^1 to produce the next ungrounded tree in the deduction of t from the tree r containing n .

In this case, $\zeta_r(n)$ is defined and $C_r(n)$ is a pseudo atom. The application of O_1 or O_{11} to n causes the leaf $\zeta_r(n)$ of the dpf for r to be expanded. Expansion of a leaf amounts to calculating the structural representation for the content of the leaf. Thus, there exists a unique correspondence between the pseudo atom subexpressions of $C_r(n)$ and the subforest of the newly generated dpf created by expanding the leaf $\zeta_r(n)$. The function ζ_s is defined as follows:

1. If the node a_1 of s is the image of a node \bar{a}_1 of t and if $\zeta_t(\bar{a}_1) = a_2$ then $\zeta_s(a_1) = a_2^*$.
2. If the node a_1 of s has no image in t (i.e. it is a node of the appended t \bar{t}) then $\zeta_s(a_1)$ is defined just in the case that $C_s(a_1)$ is a pseudo atom. In this case, since the suffix of $C_s(a_1)$ is a pseudo atom

¹We assume that O_1 is applied to a type 1 expression whose kernel is not a pseudo atom.

of $C_r(n)$, we define $\zeta_s(a_1)$ to be the image of the leaf of the dpf caused by leaf expansion of $\zeta_r(n)$ which corresponds to the appropriate pseudo atom of $C_r(n)$.

If m has no ancestor to which O_1 or O_{11} has been applied, take n to be that ancestor of m which occurs in the initial tree. This node corresponds to a subforest of the dpf associated with the initial tree. The function ζ_s may now be defined as in 1. and 2. above.

7. restricted append O_9

Suppose s results from applying O_9 to the ungrounded trees t_1 and t_2 . Since $\text{dpf}(s) = \text{dpf}(t_1)\sigma_1 \cup \text{dpf}(t_2)\sigma_2$ (where σ_1 and σ_2 are substitutions specified in the definition of O_9) we may define ζ_s to be the union of the functions $\zeta_{t_1\sigma_1}$ and $\zeta_{t_2\sigma_2}$ where $\zeta_{t_1\sigma_1}$ and $\zeta_{t_2\sigma_2}$ are calculated from ζ_{t_1} and ζ_{t_2} as outlined in step 2 above.

8. theorem utilization O_{12}

The operator O_{12} is essentially a combination of substitution, append and multiple closed branch removal. Thus, if s results from applying O_{12} to the ungrounded trees t_1 and t_2 (where t_2 results from applying a substitution to some ungrounded tree t representing a theorem), we may construct ζ_s from ζ_{t_0} and ζ_{t_1} using the rules given in steps 1, 2 and 7.

APPENDIX E Abbreviation and symbol glossary1.0 Abbreviations


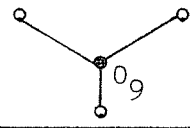
<u>Abbreviation</u>	<u>Page Introduced</u>	<u>Meaning</u>
CNF	156	Conjunctive normal form
DL	91	Definition language
DNF	146	Disjunctive normal form
dpf	31	Dependency forest
DRI	139	Defined rule of inference
EOL	91	Extended object language
fpn	140	Fully process node
fep	142	Fully expand propositional
mabr	64	Multiple closed branch removal
OL	91	Object language
pe	28	Polarized expression
POL	28	Syntax rule for construction of pe's
sc	41	Substitution component
spe	31	Skolemized polarized ex- pression
tfat	113	Truth function analysis tree
TSS	24	Type system symbol

2.0 Symbols

Many of the symbols occurring in this paper are used in several unrelated senses. In these cases, the meaning will be clear from the context. As a general policy, the auxiliary symbols $-$, $\hat{}$, $'$, as well as subscripts and superscripts, will be used to denote different elements of the same class. Thus, t may be used in certain contexts as a generic symbol for tree. In this case, \bar{t} , \hat{t} , t' , t_1 , t_2, \dots might be used to denote other trees.

Unless otherwise stated in the text, the symbols and notation given below have the meanings established on the indicated pages.

C	96	C	13	0_6	65	α	213	θ	53
D_L	101	C_t	13	0_7	72	ε membership		Ψ	15
E	89	D	45	0_8	73	λ	23	Ξ	
E_L	95	D^*	45	0_9	77	θ	13	T	84
		Appendix H D_1, \dots, D_{12}		0_{10}	79	θ_t	13	T_N	84
P	145			0_{11}	108	ν	48		
Pd	150			0_{12}	135	$\nu_D^P(e)$	105		
S	24	N	12	P	12	σ	42		
$T(e)$	32	N_t	11	P_t	11	τ	23		
$T^*(e)$	33	0_1	59			ϕ	41		
u	21	0_2	61			ω	213		
u_L	23	0_3	63						
u_L^*	38	0_4	64						
v	24	0_5	65						

$\langle \cdot \rangle_t$	12	$\textcircled{1}$	23	\bullet	23
$\langle \cdot \rangle$	12	$(\tau_1, \dots, \tau_n; \tau)$	23	\xrightarrow{L}	67
\neg	103	\textcircled{T}	23	$\xrightarrow{L_1}$	74
\vee	103	\textcircled{F}	23	$\xrightarrow{L_2}$	74
$\&$	103	$\lambda x_1 \dots x_n \bullet e$	25	\longleftrightarrow	101
\supset	103	$v+$	33	\longleftrightarrow	105
\equiv	103	$v-$	33	$\Gamma \implies t$	77
\forall	23	ζ	230		83
$\bar{\forall}$	103	ζ^{-1}	230		84
\oplus	212	$*$	213		

$\text{org}(t)$	11	$\{x_1, \dots, x_n\}$	-	set consisting of the elements x_1, \dots, x_n
		$\langle x_1, \dots, x_n \rangle$	-	ordered set consisting of the elements x_1, \dots, x_n with order given

APPENDIX F Subject Index

- dependency forests - algebraically replacing a dpf with another
 dpf, 39; leaf expansion in, 39
- forest - determined by a tree, 19
- inference operators - (for U), 52 ; utilizing truth-function
 analysis trees, 122; defined rules of inference, 139
- semantics - derived domain, value assignment, 45; valu-
 ation of U_L -expressions, 46; satisfiability,
 50 ; validity, unsatisfiability, 50; seman-
 tic considerations in E_L , 103
- substitution - substitution components, instantiation,
 unifiable, composition of, in trees with
 contents, 41, 42, 43
- syntax - symbols of U , 23; formation rules of U ,
 expressions of U , type 1 expression of U ,
 24; subexpression, 26; atomic expressions 27;
 polarized (pe), conjugate of, 28; skolemized
 polarized (spe), structural representation
 of a type 1 expression, 32; dependency forest
 associated with a type 1 expression, 33;
 skolemization of type 1 expressions via their
 dependency forests, 36; unification of spe's,
 42; linear extension 67; linear descendant, 69;
 proper subforest, 70;

patriarchs, 75; derivable in U 77; attempted
 proof graph, 83; abbreviation, 96; extended
 object language E_L , 96; pseudo atoms, 96;
 definition language, pseudo type-1 expressions,
 99 ; definition in \mathcal{D}_L , 101; primitively defin-
 able, 106; total vs partial unabbreviation, 110
 depth control in truth-function analysis, 119
 tree
 - ordered, leaf of, path of, 12; with contents
 from a set, subtree, major subtrees, 14, 15;
 isomorphism, image of tree in another, 15;
 appending, 16; node removal from, branch re-
 moval from, 17; ungrounded, 52; initial, 53

APPENDIX G Alphabetic listing and description of additional DELS
functions

1. all-elements[s;p] s is an ordered set and p is a predicate function whose domain includes the elements of s . An ordered set is returned which contains all elements of s that satisfy p . The relative order of elements is the same as in s .

$$= [\text{null}[s] \rightarrow \phi; p[\alpha[s]] \rightarrow \alpha[s] \oplus \text{all-elements}[\omega[s]; p]; \\ T \rightarrow \text{all-elements}[\omega[s]; p]]$$

2. clash[x;y] x and y are leaves of the E -clauses A and B respectively (see Binary Resolution chapter 3). Clash is true if and only if it is not the case there exists a ϵ leaves $[A]-\{x\}$ and $b \in$ leaves $[B]-\{y\}$ and such that $\text{conj}[a;b]$ is true.

$$= \text{clash1}[\text{leaves}[\text{parent-tree-specification}[x]]-\{x\}; \\ \text{leaves}[\text{parent-tree-specification}[y]]-\{y\}]$$

auxiliary function

$$\text{clash1}[u;v] = [\text{null}[u] \rightarrow T; \text{exists}[v; \lambda[[b]; [\text{conj}[\alpha[u]; b]]] \rightarrow F; \\ T \rightarrow \text{clash1}[v; \omega[u]]]$$

3. changevar[x;p] produces the substitution σ whose i^{th} component is $\langle y_i, z_i \rangle$ where y_i, z_i have the same type, y_i is a variable, z_i an expression, $y_i \in x$ and z_i satisfies the predicate p .

changevar is a DELS primitive.

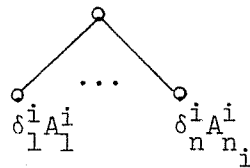
4. conj[n,m] If n and m name atomic nodes whose contents have conjugate prefixes, then conj[n,m] is true. Otherwise, conj[n,m] is false

$$=[\text{atomic}[n]\&\text{atomic}[m] \rightarrow [\text{not}[\text{prefix}[n]=\text{prefix}[m]] \rightarrow \text{T}; \text{T} \rightarrow \text{F}]; \\ \text{T} \rightarrow \text{F}]$$

5. det clauses[n] If $\text{content}[n] = \textcircled{\text{T}} \& C_1 \& \dots \& C_{N-1} C_N$ and if $C_i = \vee D_1^i \vee \dots \vee D_{n_i-1}^i D_{n_i}^i$ and $L_j^i = A_j^i$ or $A_j^i = \neg L_j^i$ where A_j^i is an atom, then detclauses returns

$$\langle t_1, \dots, t_N \rangle$$

where t_i is a tree specification of the tree



$$\text{where } \delta_j^i = \begin{cases} \textcircled{\text{T}} & \text{if } L_j^i = A_j^i \\ \textcircled{\text{F}} & \text{if } L_j^i = \neg A_j^i \end{cases}$$

$$=\text{clauses}[\text{clausenames}[n]]$$

auxiliary functions clause, clauses

$$\text{clausenames}[n] = [\text{operator}[n] = \textcircled{\text{T}} \& \rightarrow \alpha[p] \oplus \text{clausenames}[\text{aw}[p]]; \text{T} \rightarrow \{n\}]$$

$$\text{where } p = \text{new-nodes}[D_2[n;n]]$$

If $\text{content}[n] = \textcircled{\text{T}} \& C_1 \& \dots \& C_{N-1} C_N$ then clausenames[n] returns the names of those nodes (of tree added to attempted proof

graph during execution of clausenames) whose contents are

$$\bigoplus C_i \quad (1 \leq i \leq N).$$

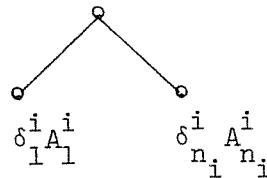
$$\underline{\text{clauses}}[x] = [\text{null}[x] \rightarrow \phi; T \rightarrow \text{clause}[\alpha[x]] \oplus \text{clauses}[\omega[x]]]$$

$$\underline{\text{clause}}[n] = \text{fep}[\alpha\omega[D_7[\{n\}]]]$$

If the content of n is $\bigoplus C_i$ then $D_7[\{n\}]$ is $\langle 1, 2 \rangle$

where $\langle 1, 2 \rangle$ is a tree specification for $2 \begin{matrix} 1 \\ \circ \end{matrix} \bigoplus C_i$

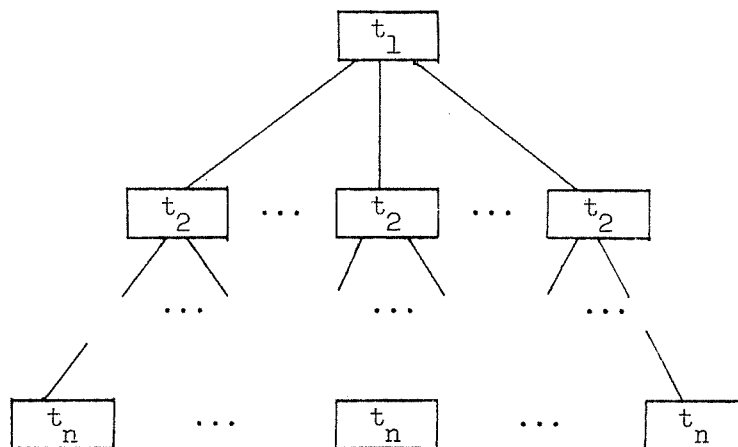
Thus, $\alpha\omega[D_7[\{n\}]] = 2$. Clause $[n]$ produces a tree specification for the \bar{E} -clause



6. exists $[x;p]$ If x is an ordered set and p is a predicate function whose domain includes the elements of x , then $\text{exists}[x;p]$ is true if there exists an element $a \in x$ such that $p[a]$ is true. Otherwise, $\text{exists}[x;p]$ is false.

$$=[\text{null}[x] \rightarrow F; p[\alpha[x]] \rightarrow T; T \rightarrow \text{exists}[\omega[x];p]]$$

7. expand $[s]$ If s is an ordered set whose elements are the tree specifications t_1, t_2, \dots, t_n , then $\text{expand}[s]$ is a tree specification for the tree



where $\boxed{t_i}$ denotes the tree specified by t_i . We note that t_1, \dots, t_n must specify ungrounded trees whose patriarchs are all on the same branch of some system tree.

$=[\text{null}[s] \rightarrow \phi; T \rightarrow \text{sa}[\text{expand}[\omega[s]]]; \text{leaves}[\alpha[s]]]$

8. member[x;y] If y is an ordered set and x an element, then $\text{member}[x;y]$ is true. Otherwise, it is false. In writing functions which utilize "member," we use the notation $x \in y$ in place of $\text{member}[x;y]$.
9. node-set-part[z] (see appendix B)
10. sa[z;x] (series append) If z is an ungrounded tree specification and x is a set of leaves of some tree, then $\text{sa}[z;x]$ appends the tree specified by z to each leaf in x . The

value of $sa[z;x]$ is a tree specification of the resulting tree.

$=[\text{null}[z] \rightarrow \text{parent-tree-specification}[\alpha[x]]];$

$\text{null}[\omega[x]] \rightarrow D_9[z;\alpha[x]];$

$T \rightarrow D_9[z;\text{im}[\alpha[x];sa[z;\omega[x]]]]]$

11. set-difference[x;y] If x and y are ordered sets, then $\text{set-difference}[x;y]$ is an ordered set consisting of all elements of x which do not occur in y . In writing functions which utilize "set-difference" we use $x-y$ in place of set-difference .

$=\text{all}[x;\lambda[[z];[\text{not}[z \in y]]]]]$

12. subspart[z] (see appendix B)

13. varp[x] If x is a tree specification, then $\text{varp}[x]$ is an ordered set consisting of all positive variables occurring in the tree specified by x .

varp is a DELS primitive.

APPENDIX H Primitive effector operations of DELS

The operators D_1, \dots, D_{12} , listed below, implement the E -inference operators O_1, \dots, O_{12} respectively. In each case the tree described in Chapter 2 for the corresponding E -inference operator is added to the attempted proof graph and its specification returned as the value of the operator.

operator	parameters	
D_1	node specification	natural number (specified first character of designated subexpression)
D_2	node specification	leaf specification
D_3	tree specification	substitution
D_4	ordered set (consisting of leaves which specify branches to be removed)	
D_5	node specification	
D_6	leaf specification (designates branch)	
D_7	ordered set (consisting of nodes of proper subforest)	
D_8	same as for D_7	
D_9	leaf specification	tree specification
D_{10}	same as for D_4	
D_{11}	node specification	natural number (see parameters for D_1)
D_{12}	leaf specification	substitution tree specification ¹

¹Specifies tree in theorem memory.

APPENDIX I Selected proofs

In this appendix, we sketch proofs of the truth preservation properties of the inference operators of U . These results support the consistency results presented at the end of section 2 chapter 2. The operators O_{11} and O_{12} also preserve truth. This may be demonstrated using techniques similar to those used in this appendix.

lemma 1 (λ -reduction) If t is satisfiable and s results from applying O_1 to t , then s is satisfiable.

proof Suppose O_1 is applied to node n of t and that $C_t(n) = \delta e$. Two cases are possible: the expression $e_1 = \lambda x_1 \dots x_n \bullet a a_1 \dots a_n$ to which λ -reduction is applied is either 1) e , or 2) not type 1.

case 2 In this case e_1 is replaced with $a\sigma$ producing $C_s(n) = \bar{e}$ where $\sigma = \{\dots, \langle a_i, x_i \rangle, \dots\}$. Suppose t is true under the valuation v which assigns values in some domain D to each Skolem symbol occurring in t . If $C_t(n)$ is true under v and some assignment to the positive variables occurring in t , then \bar{e} is also true under this same assignment.

case 1 In this case e is replaced with \bar{e} where \bar{e} results from Skolemizing $a\sigma$ in the context of δe . Suppose the new Skolem symbols h_1, \dots, h_m are introduced. Further suppose that t is true under the valuation v which assigns values in some domain D to the Skolem symbols occurring in t . It suffices to prove that

v may be extended to v^* (i.e. to include value-assignments in D for the new symbols h_1, \dots, h_n) in such a way that s is true under v^* .

Now s is true under an arbitrary extension v^* for each value assignment to the positive variables occurring in t which causes some branch of t not containing n to be true. Let the class of value assignments to the positive variables occurring in t not having this property be χ . Thus, $\delta \lambda x_1 \dots x_n \bullet a a_1 \dots a_n$ is true for every value assignment of the form $v \cup x$ where $x \in \chi^1$. This means that $\delta a \sigma$ is true for the assignment $v \cup x$.

Let Σ be the class of value assignments to the positive variables² occurring in s which do not occur in t and let M be the result of deleting from $\delta a \sigma$ all quantifiers deleted in the process of forming \bar{e} . Since $\delta a \sigma$ is true for every value assignment $v \cup x$, there exists a value assignment μ to the negative variables occurring in M , (these are determined from $\delta a \sigma$) such that M is true for any value assignment of the form $v \cup \mu \cup x \cup y$ ($x \in \chi$, $y \in \Sigma$). Suppose that the negative variable x_i occurring in M is

¹i.e. a value assignment to the Skolem symbols occurring in $t(v)$ and some assignment to the positive variables occurring in t which makes the branch containing n (but no other branch) true ($x \in \chi$).

²Let us call the symbols \textcircled{D} and \textcircled{F} negation symbols. A quantified variable x is called a positive (negative) variable of the expression e if e contains a subexpression of the form $\forall x e_1$ and $\forall x e_1$ occurs within the scope of an even (odd) number of negation symbols. It can be shown (by an inductive argument) that this notion of positive (negative) variable coincides with that given in chapter 2. (i.e. If a variable is positive (negative) in this sense, then Skolemization produces a dpf in which the variable in question has a + (-) suffix.)

replaced with the completed Skolem symbol $h_i z_1^i \dots z_{n_i}^i$ in the process of forming \bar{e} and that μ assigns x_i the value x_i^* . Let h be a value assignment which assigns the value $\{\langle z_1^{i*}, \dots, z_{n_i}^{i*} \rangle, x_i^*\}$: z_k^{i*} is the value assigned z_k^i under $x \cup y$ and x_i^* is value assigned x_i under μ to h_i . Under evaluation, $h_i z_1^i \dots z_{n_i}^i$ receives the value x_i^* if the z_k^i are assigned values either directly by value assignments of the form $x \cup y$ or indirectly by evaluation. Thus, \bar{e} is true under any assignment of the form $v \cup h \cup x \cup y$ which implies s is true under $v \cup h$.

In each of the following lemmas, v will be assumed to be a value-assignment for the Skolem functions occurring in the operand tree (s).

lemma 2 (unextended O_2) If t is true under v , then $O_2(n,k)$ is also true under v .

proof By hypothesis, $C_t(n)$ is either $\textcircled{T} \textcircled{\perp} e_1 e_2$ or $\textcircled{F} \textcircled{\perp} e_1 e_2$. Since t is true under v , there exists a true branch for each possible value-assignment to the positive variables occurring in t . If this branch is not the branch determined by k , then $O_2(n,k)$ contains this true branch. Otherwise, $C_t(n)$ is true.

case 1 $C_t(n) = \textcircled{T} \textcircled{\perp} e_1 e_2$.

In this case $\textcircled{\perp} e_1 e_2$ is true which implies either e_1 is false or e_2 is false (i.e. $\textcircled{F} e_1$ or $\textcircled{F} e_2$ is true).

Let m be that leaf of $O_2(n,k)$ such that

$$C_{O_2}^{(m)}(n,k) = \begin{cases} \textcircled{F} e_1 & \text{if } \textcircled{F} e_1 \text{ is true} \\ \textcircled{F} e_2 & \text{otherwise} \end{cases}$$

The branch of $O_2(n,k)$ determined by m is then true.

case 2 $C_t(n) = \textcircled{F} \textcircled{\textcircled{1}} e_1 e_2$.

In this case $\textcircled{\textcircled{1}} e_1 e_2$ is false which implies that e_1 and e_2 are true (i.e. $\textcircled{\textcircled{T}} e_1$ and $\textcircled{\textcircled{T}} e_2$ are true). Let m be that leaf of $O_2(n,k)$ containing $\textcircled{\textcircled{T}} e_2$. The branch of $O_2(n,k)$ determined by m is then true.

We thus see that for each value-assignment to its positive variables, $O_2(n,k)$ is true. Thus $O_2(n,k)$ is true under v .

lemma 3 (tree substitution) If t is true under v and σ is the substitution $\{\dots, \langle e_i, x_i \rangle, \dots\}$ where the $\{x_i\}$ are some of the positive variables occurring in t , then $O_3(t, \sigma)$ is also true under v .

proof Let $s = O_3(t, \sigma)$. The tree s is true under v if it is true for all possible value assignments for the positive variables occurring in s . It suffices to consider the values assigned to the expressions e_i under the semantic evaluation scheme of section 2.1.3. Since e_i has the same type as x_i there exists a value assignment to the x_i 's which agrees with that calculated for the

e_i . By hypothesis, t is true for this value assignment and thus, so is s .

lemma 4 (multiple closed branch removal) If t is true under v and O_4 removes the branches b_1, \dots, b_n from t to produce the tree s , then s is also true under v .

proof Under the evaluation scheme of section 2.1.3, a closed branch is never satisfiable (i.e. there exists no valuation under which it is true). Thus, for each assignment to the positive variables occurring in t , t must contain a true branch which is not included among those removed. Thus, for each assignment to its positive variables, s contains a true branch.

lemma 5 (node removal) If t is true under v , $n \in N_t$ and $O_5(n)$ is defined, then $O_5(n)$ is also true under v .

proof For each assignment to its positive variables, t contains a true branch. If n does not lie on this branch, then $O_5(n)$ contains the branch and is also true. If n does lie on the branch, then let the branch consist of the nodes $\{n, n_1, \dots, n_k\}$. By hypothesis, the content of each of these nodes is true. Furthermore, since O_5 does not destroy branches, $\{n_1, \dots, n_k\}$ must be a true branch of $O_5(n)$. Thus $O_5(n)$ is true under v .

lemma 6 (duplicate branch removal) If t is true under v and O_6 removes the branches b_1, \dots, b_n which are duplicates of the

branch b , then the resulting tree s , which contains b , is true under v .

proof By hypothesis, for each assignment to its positive variables, t contains a true branch h . If $h \notin \{b, b_1, \dots, b_k\}$ then h is a true branch of s . If $h \in \{b, b_1, \dots, b_k\}$ then b must be a true branch of s .

lemma 7 (proper subforest copy) Let F be a proper subforest of t , $s = O_T(F)$ and r be the tree obtained by removing all branches from t which contain nodes in common with F . If t is true under v , then either s or r is true under v .

proof Since F is a proper subforest of t , there exists a tree which is truth functionally equivalent to t and has the property that if z is a positive variable occurring in F , then z does not occur in the content of any of the elements of $N_t - N_F$. We may thus assume, without loss of generality, that t has this property.

Let $x = \{x_1, \dots, x_k\}$ be the positive variables occurring in F and $y = \{y_1, \dots, y_n\}$ those occurring in contents of elements of $N_t - N_F$. By hypothesis t is true for all value assignments to $x \cup y$. If for each such assignment, some branch of r is true, then r is true under v . Otherwise, suppose for some assignment $x_1^*, \dots, x_k^*, y_1^*, \dots, y_n^*$ no branch of r is true. Thus, some branch of F (and consequently s) must be true for this assignment. But clearly, some branch of s must be true for each assignment

$x_1^{**}, \dots, x_k^{**}, y_1^*, \dots, y_n^*$ since changing the value assignment of x can not effect the truth of the branches of r . Thus, if r is not true under v , s is true for each assignment to its positive variables. Thus, s is true under v .

lemma 8 If t is true under v and $s = O_8(t)$, then s is also true under v .

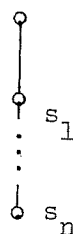
proof It can be shown by an inductive argument that $r \xrightarrow{L} \phi$ implies t is unsatisfiable. By hypothesis, there exists a proper subforest F of t such that $r = O_7(F)$ and $r \xrightarrow{L} \phi$. It follows from lemma 7 that s is true under v .

lemma 9 If t_1 and t_2 are true under v and s is the restricted append of t_2 to the leaf n of t_1 , then s is true under v .

proof It follows from lemma 3 that $t_1\sigma_1$ and $t_2\sigma_2$ are true under v (where σ_1 and σ_2 are as given in the definition of O_9). The trees $t_1\sigma_1$ and $t_2\sigma_2$ contain no common positive variables. Furthermore, for each assignment to the variables occurring in $t_1\sigma_1$, $t_2\sigma_2$ these trees contain true branches b_1 and b_2 respectively. If b_1 is not the branch determined by n , then s contains a true branch. If b_1 is the branch determined by n , then the branch of s consisting of the images of b_1 and b_2 is true. Thus, s contains a true branch for each assignment to its positive variables. Thus, s is true under v .

lemma 10 If t is true under v and $s = O_{10}(t)$, then s is true under v .

proof Utilizing lemma 8 and the fact that $r \xrightarrow{L} \phi$ implies r is unsatisfiable, it can be shown that $r \xrightarrow{L_1} \phi$ implies r is unsatisfiable. Now s is obtained from t by removing all branches which contain all patriarchs s_1, \dots, s_n associated with some occurrence of ϕ in the attempted proof graph. This means that there is a tree r of the form



such that $r \xrightarrow{L_1} \phi$. Thus, r is unsatisfiable (i.e. for every valuation v and all value-assignments to the positive variables occurring in t_1 some s_i ($1 \leq i \leq n$) is false). Since t is assumed to be true under v , some branch is true for each value assignment to the positive variables occurring in t . By the above argument, each branch removed from t by O_{10} contains at least one false node for each such assignment. The tree s must therefore contain a true branch for each value-assignment to this positive variable. Thus, s is true under v .

BIBLIOGRAPHY

- [1] Andrews, P. B. Resolution with merging. J. of ACM, vol. 15 (1968), pp. 367-381.
- [2] Beth, W. W. The Foundations of Mathematics. North Holland (1959).
- [3] Carnap, R. Einführung in die Symbolische Logik, mit besonderer Berücksichtigung ihrer Anwendungen. (Also published in English-Introduction to symbolic logic and its applications. Dover Publications, Inc. (1958)). Wien (1954).
- [4] Church, A. The calculi of lambda-conversion. Princeton Univ. Press (1941).
- [5] Cooper, D. C. Theorem proving in computers. Advances in Programming and Non-numerical Computation (ed. by Fox, L.), pp. 155-182.
- [6] Darlington, J. L. Some theorem-proving strategies based on the resolution principle. Machine Intell. 2, Edinburgh Univ. Press (1968).
- [7] _____. Automatic theorem proving with equality substitutions and mathematical induction. Machine Intell. 3, Edinburgh Univ. Press (1968), pp. 113-127.
- [8] Davis, M. and Putnam, H. A computing procedure for quantification theory. J. of ACM, vol. 7 (1960), pp. 201-215.
- [9] Dunham, B., Fridshal, R., and Sward G. L. A nonheuristic program for proving elementary logical theorems. Proceedings of IFIP Congress, (1959), pp. 282-285.
- [10] Friedman, J. A semi-decision procedure for the functional calculus. J. of ACM, vol. 10 (1963), pp. 1-24.
- [11] _____. A computer program for a solvable case of the decision problems. J. of ACM, vol. 10 (1963), pp. 348-357.
- [12] Gentzen, G. Untersuchungen über das logische schließen. Mathematische Zeitschrift, (1935), pp. 39, 176-210, 405-431.
- [13] Gilmore, P. C. A proof method for quantification theory. IBM J. Res. Dev. (1960), pp. 4, 28-35.
- [14] Herbrand, J. Investigations in proof theory. From Frege to Gödel: A Source Book in Mathematical Logic (ed. by Van Heijenoort, J.), Harvard (1967).

- [15] Hilbert, D. and Ackermann, W. Principles of Mathematical Logic. Chelsea, New York (1950).
- [16] Hintikka, K. J. J. Form and content in quantification theory. Acta Philosophica Fennica, 8, (1955), pp. 7-55.
- [17] Kallick, B. A decision procedure based on the resolution method. Proceedings of IFIP Congress, (1968).
- [18] Kleene, S. C. Introduction to Metamathematics. Princeton: Van-Nostrand, (1952).
- [19] Loveland, D. W. Theorem-provers combining model elimination and resolution. Machine Intell. 4, Edinburgh Univ. Press, (1969), pp. 73-86.
- [20] _____ A simplified format for the model elimination theorem-proving procedure. Carnegie-Mellon Univ., Dept. of Mathematics, Report 67-39 (mimeographed). Also in Machine Intell. 4, Edinburgh Univ. Press, (1968).
- [21] _____. Mechanical theorem proving by model elimination, J. of ACM, vol. 15, (1968), pp. 236-251.
- [22] McCarthy, J. A basis for a mathematical theory of computation. Proceedings of W. Jt. Comp. Conf., (1961), pp.
- [23] _____. et al. Lisp 1.5 programmers manual. The MIT Press, (1962).
- [24] Meltzer, B. A new look at mathematics and its mechanization. Machine Intell. 3, Edinburgh Univ. Press, (1968), pp. 63-70.
- [25] _____. Some notes on resolution strategies. Machine Intell. 3, Edinburgh Univ. Press, (1968), pp. 71-76.
- [26] _____. Theorem-proving for computers: some results on resolution and renaming. Comp. J., vol. 8 (1966), 341-343.
- [27] Mendelson, E. Introduction to Mathematical Logic. Princeton: Van-Nostrand, (1964).
- [28] Popplestone, R. J. Beth tree methods in theorem-proving. Machine Intell. 1, Oliver and Boyd, Edinburgh (1967), pp. 47-61.
- [29] Prawitz, D. Advances and problems in mechanical proof procedures. Machine Intell. 4, (1968), pp. 59-71.

- [30] _____. An improved proof procedure. Theoria, vol. 26 (1960), pp. 102-139.
- [31] _____, Prawitz, H., and Voghera, N. A mechanical proof procedure and its realization in an electronic computer. J. of ACM, vol. 7 (1960), pp. 102-128.
- [32] Quine. Methods of Logic. Henry Holt and Co., New York, (1959).
- [33] _____. A proof procedure for quantification theory. J. of Sym. Logic, vol. 20 (1955), pp. 141-149.
- [34] Robinson, J. A. A machine-oriented logic based on the resolution principle. J. of ACM, vol. 12 (1965), pp. 23-41.
- [35] _____. Automatic deduction with hyper-resolution. Int. J. Comp. Math., vol. 1 (1965), pp. 227-234.
- [36] _____. A review of automatic theorem-proving. Proc. Symp. Appl. Math. 19, Amer. Math. Soc., Providence, R. I. (1967), pp. 686-697.
- [37] Schonfinkel, M. On the building blocks of mathematical logic. From Frege to Gödel: A Source Book in Mathematical Logic (ed. by Van Heijenoort, J.), Harvard (1967), pp. 355-366.
- [38] Sibert, E. E. A machine-oriented logic incorporating the equality relation. (Ph.D. thesis) Rice Univ. (1967).
- [39] _____. The equality relation in mechanical theorem proving. Machine Intell. 4, Edinburgh Univ. Press (1968).
- [40] Slagle, J. Automatic theorem proving with renamable and semantic resolution. J. of ACM, vol. 14 (1967), pp. 687-697.
- [41] Skolem, T. Über einige Grundlagenfragen der Mathematik. Skrifter utgitt av Det Norske Videnskaps-Academi i Oslo I, Mat.-Natur, Klasse, no. 4, (1929).
- [42] _____. Über die mathematische logik. Norsk Mat. Tidsskr. 10, (1928), pp. 125-142.
- [43] Smullyan, R. M. First Order Logic. Springer-Verlag, Inc., New York (1968).
- [44] Wang, H. Towards mechanical mathematics. IBM J. Res. Dev. vol. 4, (1960), pp. 2-22.
- [45] _____. Proving theorems by pattern recognition-I. Communs. ACM, vol. 3, (1960), pp. 220-234.

- [46] _____. Proving theorems by pattern recognition-II. Bell Syst. Tech. J., vol. 40, (1960), pp. 1-41.
- [47] Whitehead, A. N. and Russell, B. Principia Mathematica to *56. Cambridge at the Univ. Press (1962).
- [48] Wos, L., Carson, D., and Robinson, G. The unit preference strategy in theorem proving. Proc. AFIPS 1964 Fall Jt. Comp. Conf. 26 Part II, pp. 615-621.
- [49] _____, Robinson, G., and Carson, D. Efficiency and completeness of the set of support strategy in theorem proving. J. of ACM, vol. 12 (1965), pp. 536-541.
- [50] _____, _____, _____, and Shalla, L. Demodulation in theorem proving. J. of ACM, vol. 14 (1967), pp. 698-709.
- [51] Luckham, D. The resolution principle in theorem-proving. Machine Intell. 1, Oliver and Boyd, Edinburgh, (1967), pp. 47-61.