ON CONNECTIONS AND HIGHER-ORDER LOGIC

by

Peter B. Andrews
Department of Mathematics
Carnegie Mellon University
Pittsburgh, PA 15213

Research Report No. 88-30 2

September 1988

On Connections and Higher-Order Logic

Peter B. Andrews

Mathematics Department,
Carnegie Mellon University,
Pittsburgh, Pa. 15213, U.S.A.

Abstract

This is an expository introduction to an approach to theorem proving in higher-order logic based on establishing appropriate *connections* between subformulas of an expanded form of the theorem to be proved. *Expansion trees* and *expansion proofs* play key roles.

KEY WORDS: higher-order logic, type theory, mating, connection, expansion proof

1. Introduction

Theorem proving is difficult and deals with complex phenomena. The difficulties seem to be compounded when one works with higher-order logic, but the rich expressive power of Church's formulation [18] of this language makes research on theorem proving in this realm very worthwhile. In order to make significant progress on this problem, we need to try many approaches and ideas, and explore many questions. The purpose of this paper is to provide an informal and expository introduction to an approach based on establishing appropriate *connections* between subformulas of an expanded form of the theorem to be proved. We hope to create an awareness of the many problems and questions which await investigation in this research area.

A question which is highly relevant to theorem proving is "What *makes* a logical formula valid?". Of course, there can be many ways of answering this question, and each can be the basis for a method of proof.

One approach to the above question is semantic. Theorems express essential truths and thus are true in all models of the language in which they are expressed. Truth can be perceived from many perspectives, so there

This is an extended version of a lecture presented to the 8th International Conference on Automated Deduction in Oxford, England on July 27, 1986. This material is based upon work supported by the National Science Foundation under grants DCR-8402532 and CCR-8702699.

may be many essentially different proofs of theorems. This point of view is very appealing, but it does not shed much light on the basic question of *what makes* certain sentences true in all models, while others are not. It's certainly a good idea to take advantage of semantical insights wherever these can be used to guide the theorem-proving process, but thus far significant progress in this direction has been made only in a few special contexts, such as geometry theorem proving.

Of course, theorems are formulas which have proofs, and every proof in any logical system may provide some insight. This suggests seeing what one can learn by studying the forms proofs can take. While this may be helpful, many of the most prominent features of proofs seem to be influenced as much by the logical system in which the proof is given as by the theorem that is being proved.

Let's focus on trying to understand what there is about the syntactic structures of theorems that makes them valid.

2. The structure of tautologies of propositional calculus

In the case of formulas of propositional calculus, we can certainly make good use of semantical insights, since in classical two-valued propositional calculus (to which we restrict attention here) the theorems are precisely the tautologies, and we can test a formula for being a tautology in an explicit syntactic way. However, simply checking each line of a truth table is not really very enlightening, and we may still find ourselves asking "What is there about the structure of this formula which makes it a tautology?".

Clearly the pattern of occurrences of positive and negative literals in a tautology is very important. One can gain much insight into the structure of a wff by putting it into negation normal form (nnf). A wff is in negation normal form, and is called a negation normal formula (nnf), iff it contains no propositional connectives other than $\vee$, $\wedge$, and $\sim$, and the scope of each occurrence of $\sim$ is atomic. In order to examine a nnf, it's helpful to arrange it in a two-dimensional diagram which we call a vpform (vertical path form). We display disjunctions horizontally, and conjunctions vertically.

For example, consider the wff below, which we call THM52:

```
[[P ≡ Q] ≡ R] ⊃ [P ≡.Q ≡ R]
```

(A dot in a wff stands for a left bracket whose mate is as far to the right as possible without changing the pairing of brackets already present.)

THM52 with ≡ eliminated is:

```
    [[[[P ⊃ Q] ∧ .Q ⊃ P] ⊃ R]
    ∧ .R ⊃ .[P ⊃ Q] ∧.Q ⊃ P]
⊃     .[P ⊃ .[Q ⊃ R] ∧.R ⊃ Q]
    ∧.[[Q ⊃ R] ∧ .R ⊃ Q] ⊃ P
```

A negation normal form of THM52 is:

```
[[~ P ∨ Q] ∧ [~ Q ∨ P] ∧ ~ R]
∨ [R ∧ .[P ∧ ~ Q] ∨ .Q ∧ ~ P]
∨   .[~ P ∨ .[~ Q ∨ R] ∧ .~ R ∨ Q]
  ∧ .[Q ∧ ~ R] ∨ [R ∧ ~ Q] ∨ P
```

The vpform of this is:

$$\begin{bmatrix} \sim P \lor Q \\ \sim Q \lor P \\ \sim R \end{bmatrix} \lor \begin{bmatrix} R \\ \begin{bmatrix} P \\ \sim Q \end{bmatrix} \lor \begin{bmatrix} Q \\ \sim P \end{bmatrix} \end{bmatrix} \lor \begin{bmatrix} \sim P \lor \begin{bmatrix} \sim Q \lor R \\ \sim R \lor Q \end{bmatrix} \\ \begin{bmatrix} Q \\ \sim R \end{bmatrix} \lor \begin{bmatrix} R \\ \sim Q \end{bmatrix} \lor P \end{bmatrix}$$

In this form the structure of the wff is much easier to comprehend. Let us also look at the negation of this wff. A negation normal form of ~THM52 is:

```
[[P ∧ ~ Q] ∨ [Q ∧ ~ P] ∨ R]
∧ [~ R ∨ .[~ P ∨ Q] ∧.~ Q ∨ P]
∧ .[P ∧ .[Q ∧ ~ R] ∨ .R ∧ ~ Q]
   ∨ .[~ Q ∨ R] ∧ [~ R ∨ Q] ∧ ~ P
```

The vpform of this is:

$$\begin{bmatrix} \begin{bmatrix} P \\ \sim Q \end{bmatrix} \lor \begin{bmatrix} Q \\ \sim P \end{bmatrix} \lor R \\ \sim R \lor \begin{bmatrix} \sim P \lor Q \\ \sim Q \lor P \end{bmatrix} \\ \begin{bmatrix} P \\ \begin{bmatrix} Q \\ \sim R \end{bmatrix} \lor \begin{bmatrix} R \\ \sim Q \end{bmatrix} \end{bmatrix} \lor \begin{bmatrix} \sim Q \lor R \\ \sim R \lor Q \\ \sim P \end{bmatrix} \end{bmatrix}$$

There is an obvious relationship between the two vpforms above.

A *disjunctive (horizontal) path* through a nnf W is a set of literal-occurrences in W corresponding to a disjunction of literals obtained from W by repeatedly replacing subformulas of W of the form $[A^1 \land \dots \land A^n]$ by one of the A^i until no conjunctions remain. Similarly, a *conjunctive (vertical) path* is a set of literal-occurrences corresponding to a conjunction of literals obtained by repeatedly replacing subformulas of the form $[A^1 \lor \dots \lor A^n]$ by one of the A^i until no disjunctions remain. (For example, one vertical path consists of the left-most literals in the vpform above.) A conjunctive normal form of a wff is the conjunction of the disjunctions of literals in its horizontal paths, and a disjunctive normal form of a wff is the disjunction of the conjunctions of literals in its vertical paths. (See [7].)

A *connection* is a pair of literal-occurrences, and a *mating* is a set of connections, i.e., a relation between occurrences of literals. The terminology *mating* was used by Martin Davis [19]. In propositional calculus we require that connected (or mated) literals be complementary, and when these ideas are lifted to first- or higher-order logic we require that there be some substitution which simultaneously makes the pairs in each connection complementary. When speaking about wffs of propositional calculus we usually tacitly assume that literals are connected iff they are complementary.

We say that a mating *spans* a path iff there is a connection in the mating between two literals on the path. This means that two literals on the path are complementary. A wff is a tautology if and only if all the disjunctions in its conjunctive normal form are tautologies, which is equivalent to saying that all its horizontal paths are spanned. Similarly, a wff is a contradiction if and only if all the conjunctions in its disjunctive normal form are contradictions, which is equivalent to saying that all its vertical paths are spanned. Depending on whether we are looking for a tautology or a contradiction, we say that a mating *spans* a nnf iff it spans all its horizontal paths or all its vertical paths. Of course, when checking whether a mating spans a nnf, it's not always necessary to examine each path completely and individually. (By examining the paths in the vpforms above it is easy to see that THM52 is a tautology and that ∼THM52 is a contradiction.)

For many purposes it doesn't matter whether one proves that a wff is a tautology, or proves that its negation is a contradiction. In general, one should be able to display the work in whichever format one prefers, and translate easily between them.

When lifted to first-order logic, these ideas are the foundation of the method of theorem proving which has been called the *connection* method [12] and the *mating* method [5]. Bibel's book [12] contains a wealth of information and ideas, many of which appeared earlier in his many papers on this subject. Of course, one can see the antecedents of this approach to theorem proving in certain early papers in this field, most notably [31].

A mating which spans a nnf is also called *p-acceptable* (*path-acceptable*). We may also use the word *acceptable* in a more general sense and say that a mating is acceptable if it satisfies some criterion which guarantees that the formula is a tautology (or a contradiction). For wffs in conjunctive normal form, another example of a criterion which guarantees that a wff is contradictory is that every cycle must contain a merge (see [3] and [32]). It would be valuable to develop additional criteria for acceptability. Of course, such criteria are most useful if they lift to higher levels of logic, as with resolution.

Note that if we have a number of conditions which must be satisfied by an acceptable mating, we may be able to apply them simultaneously to limit our search, or we may be able to use different criteria in different situations to motivate mating certain literal-occurrences. Perhaps much more can be said on this subject.

3. The structure of valid formulas of higher-order logic

We turn our attention to higher-order logic, which is synonymous with type theory, and was developed by Bertrand Russell. We shall use a very elegant and useful formulation of type theory due to Alonzo Church [18] which is known as the typed λ-calculus. A description of this can be found in [7], but we give a brief introduction to some basic features of the notation before proceeding.

All entities have types, and variables and constants are letters with subscripts which indicate their types. However, we often omit the subscript from a symbol after its first occurrence in a wff. If α and β are types, then $(\alpha\beta)$ is the type of functions from elements of type β to elements of type α. The type of truth values is o, and we may regard entities of type $(o\alpha)$ as sets whose elements are of type α or as properties of objects of type α. In higher-order logic, all types of variables can be quantified. Of course, propositional calculus and first-order logic are parts of type theory.

One view is that in higher-order logic as well as in first-order logic, formulas are valid because they can be expanded into tautologies. (Of course, this is the fundamental idea underlying various forms of Herbrand's Theorem.) The simple tautological structure of a valid wff can be hidden in various ways. Valid formulas may be regarded as tautologies which have been abbreviated by existential generalization, disjunctive simplification, λ-expansion, and the introduction of definitions. Their basically tautological structure has been disguised, and to prove them we must unmask the disguises.

Let us consider an example of how one can start with a tautology and disguise it by using these four operations to obtain a meaningful theorem of type theory. Start with a tautology:

$$
\begin{aligned}
(1)\qquad & [Q_o \wedge \sim S_o] \\
& \vee\ [N_o \wedge \sim R_o] \\
& \vee\ [M_o \wedge \sim Q] \\
& \vee\ .[\sim M \vee S] \wedge .\sim N \vee R
\end{aligned}
$$

Substitute to get another tautology:

$$
\begin{aligned}
(2)\qquad & [P_{o\iota}\ [H_{\iota\iota}\ X_\iota] \wedge \sim P\ .G_{\iota\iota} .H\ X] \\
& \vee\ [P\ Y_\iota \wedge \sim P\ .G\ Y] \\
& \vee\ [P\ X \wedge \sim P\ .H\ X] \\
& \vee\ .[\sim P\ X \vee P.G.H\ X] \wedge .\sim P\ Y \vee P.G\ Y
\end{aligned}
$$

Existentially generalize on $[H_{\iota\iota}\ X_\iota]$:

$$
\begin{aligned}
(3)\qquad & \exists X_\iota\ [P_{o\iota}\ X \wedge \sim P.G_{\iota\iota}\ X] \\
& \vee\ [P\ Y_\iota \wedge \sim P\ .G\ Y] \\
& \vee\ [P\ X \wedge \sim P\ .H_{\iota\iota}\ X] \\
& \vee\ .[\sim P\ X \vee P.G.H\ X] \wedge .\sim P\ Y \vee P.G\ Y
\end{aligned}
$$

Existentially generalize again:

(4) $\exists X_\iota\, [P_{o\iota}\, X \wedge \sim P.G_{\iota\iota}\, X]$
$\vee\ \exists X\, [P\, X \wedge \sim P.G\, X]$
$\vee\ [P\, X \wedge \sim P\, .H_{\iota\iota}\, X]$
$\vee\ .[\sim P\, X \vee P.G.H\, X] \wedge .\sim P\, Y_\iota \vee P.G\, Y$

Apply $\vee$ simplification:

(5) $\exists X_\iota\, [P_{o\iota}\, X \wedge \sim P.G_{\iota\iota}\, X]$
$\vee\ [P\, X \wedge \sim P\, .H_{\iota\iota}\, X]$
$\vee\ .[\sim P\, X \vee P.G.H\, X] \wedge .\sim P\, Y_\iota \vee P.G\, Y$

Existentially generalize:

(6) $\exists X_\iota\, [P_{o\iota}\, X \wedge \sim P.G_{\iota\iota}\, X]$
$\vee\ \exists X\, [P\, X \wedge \sim P.H_{\iota\iota}\, X]$
$\vee\ .[\sim P\, X \vee P.G.H\, X] \wedge .\sim P\, Y_\iota \vee P.G\, Y$

Next we perform two trivial changes of the wff which we don't really regard as disguising its structure in any significant way. Introduce $\forall$:

(7) $\forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \exists X_\iota\, [P_{o\iota}\, X \wedge \sim P.G\, X]$
$\vee\ \exists X\, [P\, X \wedge \sim P.H\, X]$
$\vee\ .\forall X\, [\sim P\, X \vee P.G.H\, X] \wedge \forall Y_\iota .\sim P\, Y \vee P.G\, Y$

Rewrite $\exists$ as $\sim\forall\sim$ and apply De Morgan's Laws:

(8) $\forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim \forall X_\iota\, [\sim P_{o\iota}\, X \vee P.G\, X]$
$\vee \sim \forall X\, [\sim P\, X \vee P.H\, X]$
$\vee\ .\forall X\, [\sim P\, X \vee P.G.H\, X] \wedge \forall Y_\iota .\sim P\, Y \vee P.G\, Y$

λ-expand:

(9) $\forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim [\lambda G\, .\forall X_\iota .\sim P_{o\iota}\, X \vee P.G\, X]\, G$
$\vee \sim [\lambda G\, .\forall X .\sim P\, X \vee P.G\, X]\, H$
$\vee\ .[\lambda G\, .\forall X .\sim P\, X \vee P.G\, X][\lambda Z_\iota\, .G.H\, Z] \wedge \forall Y_\iota .\sim P\, Y \vee P.G\, Y$

Existentially generalize on $[\lambda G_{\iota\iota}\, .\forall X_\iota .\sim P\, X \vee P.G\, X]$:

(10) $\exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim M\, G \vee \sim M\, H$
$\vee\ .M\, [\lambda Z_\iota\, .G.H\, Z] \wedge \forall Y_\iota .\sim P_{o\iota}\, Y \vee P.G\, Y$

λ-expand:

(11) $\exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim M\, G \vee \sim M\, H$
$\vee\ .M\, [[\lambda G\, \lambda H\, \lambda Z_\iota\, .G.H\, Z]\, G\, H] \wedge \forall Y_\iota .\sim P_{o\iota}\, Y \vee P.G\, Y$

Introduce the definition $\circ$ for $[\lambda G_{\iota\iota}\, \lambda H_{\iota\iota}\, \lambda Z_\iota\, .G.H\, Z]$, writing $[G \circ H]$ (the composition of G and H) as an abbreviation for $[\circ\, G\, H]$:

(12) $\exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim M\, G \vee \sim M\, H$

$$\vee \;.M\,[G \circ H] \wedge \forall Y_{\iota}.\sim P_{o\iota}\, Y \vee P.G\, Y$$

Introduce $\forall$:

(13) $\forall P_{o\iota}\, \exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ \sim M\, G \vee \sim M\, H$
$\vee\ .M\,[G \circ H] \wedge \forall Y_{\iota}.\ \sim P\, Y \vee P.G\, Y$

Again we make a trivial notational change, introducing $\supset$:

(14) $\forall P_{o\iota}\, \exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ [M\, G \wedge M\, H]$
$\supset\ .M\,[G \circ H] \wedge \forall Y_{\iota}.P\, Y \supset P.G\, Y$

We have thus arrived at the
THEOREM: For any set P, there is a set M of functions on P which is closed under composition.

We shall call this theorem THM112. It's a trivial theorem, but it provides a nice simple example of many basic phenomena, so it will be used to illustrate a number of points.

Of course, the process leading to THM112 above provides one form of a proof of the theorem, and this suggests an approach to the problem of searching for proofs of theorems of higher-order logic. Starting with the theorem to be proved, apply the inverses of the operations used above to find the tautology hidden in the theorem.

4. Expansion trees and expansion proofs

The tautology hidden in a theorem reminds us of a Herbrand expansion, and indeed, it essentially is one in the context of higher-order logic, and it sheds a lot of light on the essential syntactic structure of the theorem. However, it's a little hard to regard the tautology as a form of proof of the theorem, because we can't recover the theorem from the tautology; indeed, the tautology could be used to prove many theorems, including the ultimately weak theorem $\exists p_o\, p$, which is an immediate consequence of every theorem.

To understand in this sense why a formula is valid one needs more than the tautology hidden within it; one needs to know how to expand the formula into the tautology. If you haven't thought much about how to extend various methods of automated theorem proving to higher-order logic, you may not have worried much about the fact that it can be rather awkward to deal with a situation in which new quantifiers and propositional connectives are continuously introduced into the formula you're working on. To deal with such a situation efficiently, and to make rigorous arguments about the processes involved, it's important to have a nice clean representation of the relevant logical structures.

A very elegant, concise, and nonredundant way of representing the theorem, the tautology, and the relationship between them is an *expansion tree proof*, otherwise known as an *ET-proof* or an *expansion proof*. This concept, which grew out of ideas in [6], was developed by Dale Miller in his thesis [25], and the details of the definition can also be found in [27].

We need not repeat these details here, but let us give an informal disussion of this concept which should convey the main ideas involved in it. A familiar way of representing a wff of first-order logic is to regard it as a tree (i.e., a connected graph with no cycles) growing downward from a topmost node which is called its root. With each node of the tree is associated a propositional connective, quantifier, or atom, and if the node is not associated with an atom, there are subtrees below the node which represent the scopes of the connective or quantifier. Starting from this idea, let us enrich it to obtain what Miller calls an *expansion tree* for a wff of first- or higher-order logic.

Let us call the wff represented by a tree Q as described above the *shallow formula* Sh(Q) of the tree (and of the node which is its root). With each expansion tree Q is also associated a *deep formula* Dp(Q) which represents the result of instantiating the quantifier-occurrences in Q with terms which are attached as labels to the arcs descending from their nodes.

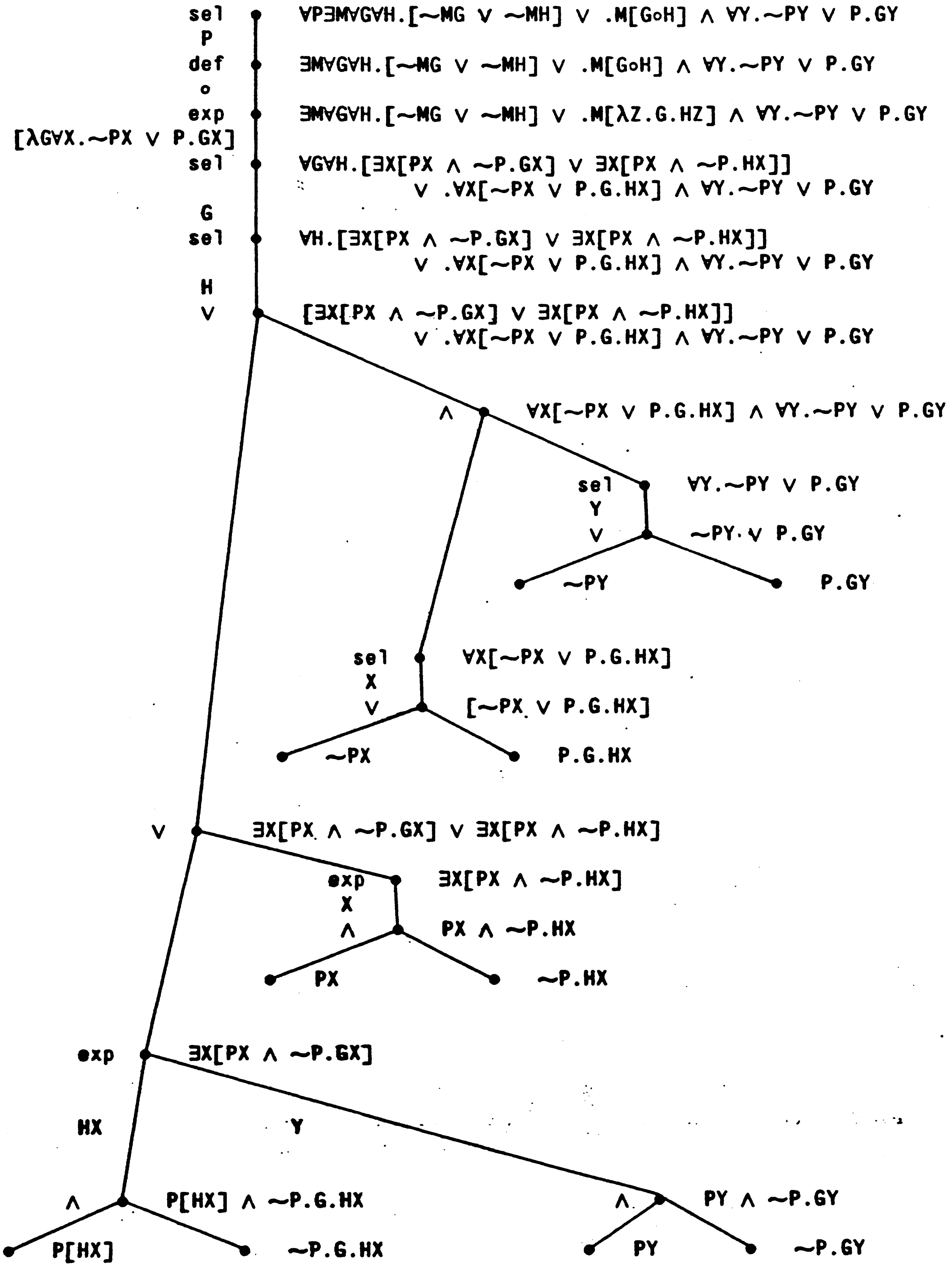
Let us call a node of a tree an *expansion node* if it corresponds to an essentially existential (see p. 123 of [7]) quantifier, and a *selection node* if it corresponds to an essentially universal quantifier. (In *dual expansion trees*, the roles of universal and existential quantifiers are interchanged.) Let finitely many arcs labeled with terms descend from each expansion node, so that if the expansion node has shallow formula $\exists x$ **B**, and if t is the term labeling an arc descending from that node, then the type of t is the same as that of x, and the node at the lower end of that arc has as shallow formula the λ-normal form of $[[\lambda x\ B]\ t]$, i.e., the result of instantiating the quantifier with the term t. The term t is called an *expansion term*. Selection nodes satisfy a similar condition, except that only one arc may descend from a selection node, and the term labeling it must be a suitably chosen parameter called a *selected parameter.*

In an expansion tree, if Q is an expansion node and Q_1, ... Q_n are the nodes immediately below Q, then Dp(Q) is $[Dp(Q_1) \vee ... \vee Dp(Q_n)]$. (In a dual expansion tree, however, Dp(Q) is $[Dp(Q_1) \wedge ... \wedge Dp(Q_n)]$.) The deep and shallow formulas are the same for the leaves of these trees.

In order to deal effectively with definitions, it is useful to add one more condition to the inductive definition of an expansion tree. If Q is a node whose shallow formula contains an abbreviation (such as the wff $[\circ\ G\ H]$ of the example above), then one may call Q a *definition node* and create a node Q_1 below Q whose shallow formula is the result of replacing the abbreviation by the wff for which it stands ($[[\lambda G\ \lambda H\ \lambda Z_\iota .G.H\ Z]\ G\ H]$ in the example above) and putting the result into λ-normal form. Q_1 and Q have the same deep formulas.

An example of an expansion tree which represents the proof of THM112 given above may be seen in Figure 4-1. The shallow formula for each node is written to the right of the node. The deep formulas are not displayed, but it is not hard to see that the deep formula for the whole expansion tree is formula (2) of the previous section.

Figure 4-1: EXPANSION TREE PROOF OF THM112


sel
P
def
o
exp
[λG∀X.∼PX ∨ P.GX]
sel
G
sel
H
∨
∀P∃M∀G∀H.[∼MG ∨ ∼MH] ∨ .M[G∘H] ∧ ∀Y.∼PY ∨ P.GY
∃M∀G∀H.[∼MG ∨ ∼MH] ∨ .M[G∘H] ∧ ∀Y.∼PY ∨ P.GY
∃M∀G∀H.[∼MG ∨ ∼MH] ∨ .M[λZ.G.HZ] ∧ ∀Y.∼PY ∨ P.GY
∀G∀H.[∃X[PX ∧ ∼P.GX] ∨ ∃X[PX ∧ ∼P.HX]] ∨ .∀X[∼PX ∨ P.G.HX] ∧ ∀Y.∼PY ∨ P.GY
∀H.[∃X[PX ∧ ∼P.GX] ∨ ∃X[PX ∧ ∼P.HX]] ∨ .∀X[∼PX ∨ P.G.HX] ∧ ∀Y.∼PY ∨ P.GY
[∃X[PX ∧ ∼P.GX] ∨ ∃X[PX ∧ ∼P.HX]] ∨ .∀X[∼PX ∨ P.G.HX] ∧ ∀Y.∼PY ∨ P.GY
∧
∀X[∼PX ∨ P.G.HX] ∧ ∀Y.∼PY ∨ P.GY
sel
Y
∨
∀Y.∼PY ∨ P.GY
∼PY ∨ P.GY
∼PY
P.GY
sel
X
∨
∀X[∼PX ∨ P.G.HX]
[∼PX ∨ P.G.HX]
∼PX
P.G.HX
∨
∃X[PX ∧ ∼P.GX] ∨ ∃X[PX ∧ ∼P.HX]
exp
X
∧
∃X[PX ∧ ∼P.HX]
PX ∧ ∼P.HX
PX
∼P.HX
exp
∃X[PX ∧ ∼P.GX]
HX
Y
∧
P[HX] ∧ ∼P.G.HX
P[HX]
∼P.G.HX
∧
PY ∧ ∼P.GY
PY
∼P.GY

It may be helpful to visualize expansion trees as being laid out in three dimensions. Go downward along the z-axis to do selections, expansions, and definition instantiations, display conjunctions parallel to the y-axis, and display disjunctions parallel to the x-axis. Note how appropriate the words *deep* and *shallow* are from this point of view.

An expansion tree which satisfies certain conditions is called an *expansion proof.* In such a tree, the shallow formula is the wff being proved, and the deep formula is the underlying tautology. The conditions are:

1. The deep formula must be a tautology.

2. Selections must be done right. (We discuss this more thoroughly below.)

If one uses dual expansion trees instead of expansion trees, an expansion proof (which might also be called an *expansion refutation*) must have for its deep formula a contradiction instead of a tautology. In an expansion tree, which represents a proof which works its way up from the bottom of the tree to the top node, expansion corresponds to inverse existential generalization, whereas in a dual tree, which represents a refutation which starts at the top node and works down to a contradiction, expansion corresponds to universal instantiation. Actually, from now on we'll mainly be concerned with dual trees. A dual expansion tree for a wff equivalent to ∼THM112 is in Figure 4-2, which should be compared with Figure 4-1. Note that its deep formula is the contradiction

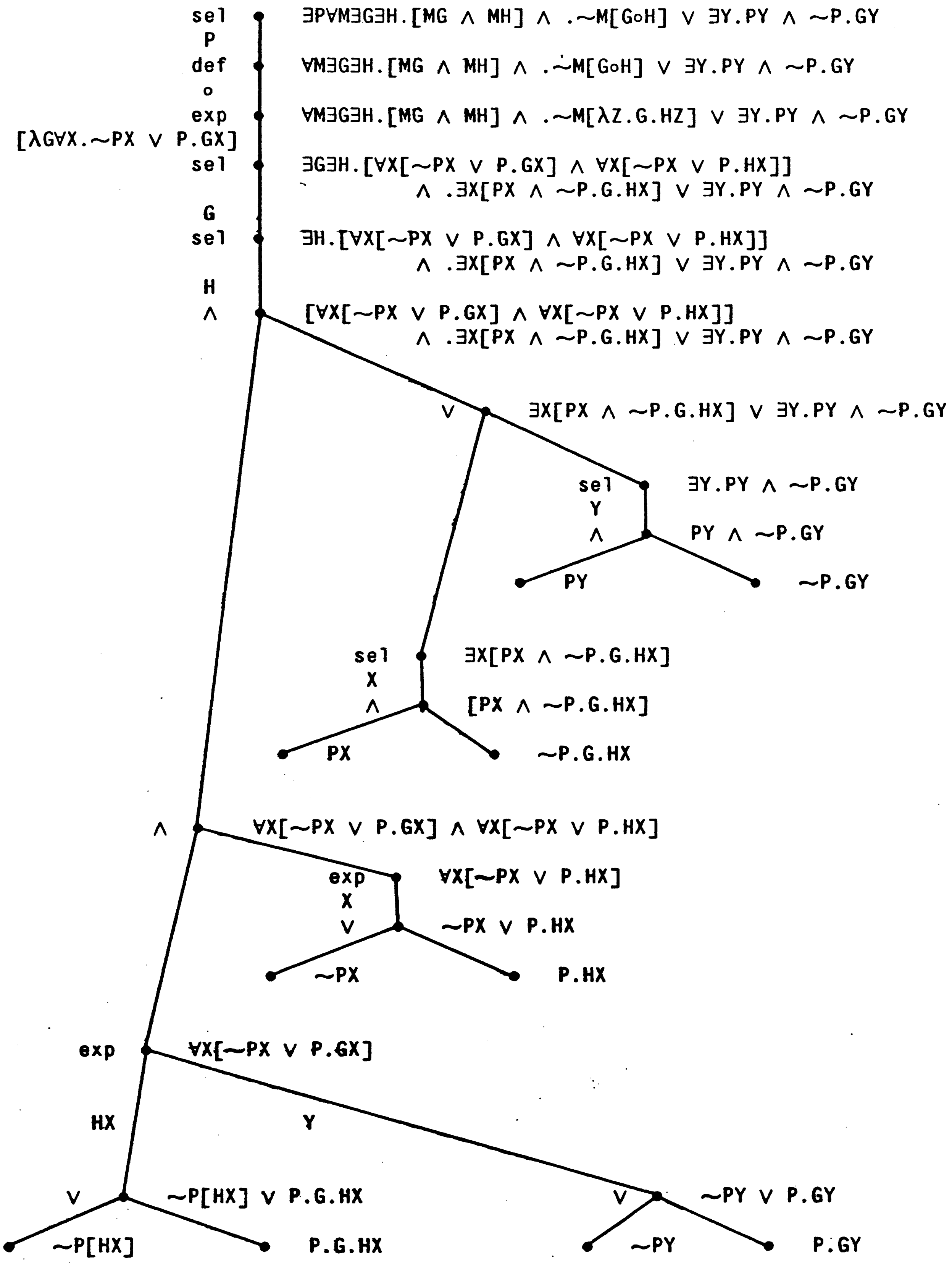
$$[\sim P_{o\iota}\ [H_{\iota\iota}\ X_{\iota}] \lor P\ [G_{\iota\iota}.H\ X]] \land [\sim P\ Y_{\iota} \lor P\ [G\ Y]] \land [\sim P\ X \lor P\ [H\ X]]$$
$$\land\ .[\ P\ X \land \sim P.G.H\ X] \lor .P\ Y \land \sim P.G\ Y.$$

One can make the condition that "selections must be done right" precise in a variety of ways, and the technical details need not concern us much right now. If one generates the expansion tree in a step-by-step process starting from the formula to be proved, then it suffices to choose a new parameter which occurs nowhere in the tree each time a selection is made.

Of course, if one is just presented with an expansion tree, and asked whether it is an expansion proof, one needs a criterion to check. One such criterion, which is in Miller's thesis and is similar to one in Bibel's book, is that the imbedding relation on selected parameters be irreflexive. The imbedding relation is the transitive closure of the relation $<^0$ such that **a** $<^0$ **b** iff there is an expansion term occurrence **t** such that **a** occurs in **t** and **b** is selected below **t** in Q.

When searching for an expansion proof, we generally won't know just what expansion terms we should use, so the expansion terms will contain free variables for which we will make substitutions later. Of course, when we make a substitution, we must do it in a way that does not violate the imbedding condition. Just as in first-order logic, we can do this by introducing Skolem terms in place of the selected parameters. In higher-order logic, we may introduce new quantifiers in the expansion terms, so we can't do all the Skolemization in a preprocessing stage. This is why we need the selection operation even if we use Skolemization.

Figure 4-2: DUAL EXPANSION TREE FOR ~THM112


sel
P
def
o
exp
[λG∀X.~PX ∨ P.GX]
sel
G
sel
H
∧
∃P∀M∃G∃H.[MG ∧ MH] ∧ .~M[G∘H] ∨ ∃Y.PY ∧ ~P.GY
∀M∃G∃H.[MG ∧ MH] ∧ .~M[G∘H] ∨ ∃Y.PY ∧ ~P.GY
∀M∃G∃H.[MG ∧ MH] ∧ .~M[λZ.G.HZ] ∨ ∃Y.PY ∧ ~P.GY
∃G∃H.[∀X[~PX ∨ P.GX] ∧ ∀X[~PX ∨ P.HX]] ∧ .∃X[PX ∧ ~P.G.HX] ∨ ∃Y.PY ∧ ~P.GY
∃H.[∀X[~PX ∨ P.GX] ∧ ∀X[~PX ∨ P.HX]] ∧ .∃X[PX ∧ ~P.G.HX] ∨ ∃Y.PY ∧ ~P.GY
[∀X[~PX ∨ P.GX] ∧ ∀X[~PX ∨ P.HX]] ∧ .∃X[PX ∧ ~P.G.HX] ∨ ∃Y.PY ∧ ~P.GY
∨
∃X[PX ∧ ~P.G.HX] ∨ ∃Y.PY ∧ ~P.GY
sel
Y
∧
∃Y.PY ∧ ~P.GY
PY ∧ ~P.GY
PY
~P.GY
sel
X
∧
∃X[PX ∧ ~P.G.HX]
[PX ∧ ~P.G.HX]
PX
~P.G.HX
∧
∀X[~PX ∨ P.GX] ∧ ∀X[~PX ∨ P.HX]
exp
X
∨
∀X[~PX ∨ P.HX]
~PX ∨ P.HX
~PX
P.HX
exp
∀X[~PX ∨ P.GX]
HX
Y
∨
~P[HX] ∨ P.G.HX
~P[HX]
P.G.HX
∨
~PY ∨ P.GY
~PY
P.GY

Each Skolem term consists of a new function symbol with arguments which are the expansion terms above its selection node in the tree. Note that if **a** and **b** are selected parameters, **a** < **b** means that the Skolem term for **a** is imbedded in the Skolem term for **b**. If one Skolemizes naively in higher-order logic, the Skolem functions can serve as choice functions, and one can derive certain consequences of the Axiom of Choice even if one did not intend to enrich one's logic in this way. The right way to do Skolemization in higher-order logic is discussed in Miller's thesis. Skolem terms often get awkwardly long, but when we use Skolemization, we can simplify the notation by just writing a single label for each Skolem term, and keeping track of what it stands for. We will do this in examples below.

Of course, even if one wants to substitute for variables in the expansion tree, one can use selected parameters instead of Skolem terms if one takes care not to violate the imbedding condition. In particular, one can integrate the imbedding relation into the unification algorithm so that only appropriate substitutions are produced. This is discussed for first-order logic in Bibel's book, and the details for higher-order logic should be fully worked out soon.

Note that an expansion proof can be used to provide an extremely economical representation of the essential information needed to prove a theorem. This information consists of the theorem itself, the general structure of the expansion tree, the expansion terms at each expansion node, the selected parameter at each selection node, and information about which occurrences of definitions are instantiated at definition nodes. The wffs associated with the nodes of the tree other than the root can be calculated. Thus, the expansion tree displayed in Figure 4-1 can also be displayed as in Figure 4-3.

The idea of an expansion tree is really very simple and natural. However, I want to emphasize its importance as a conceptual advance. It provides a fundamental conceptual building block on which we can base the extension of the connection or mating approach to theorem proving from first- to higher-order logic.

5. Translations between formats

As research on automated deduction progresses, we shall probably reach a point where we are less concerned with what format is being used, and more with deeper issues. Nevertheless, certain ideas do occur more naturally in one context than in another, and it's well known in artificial intelligence, mathematics, and other disciplines that the way one represents a problem can be very important, so it's nice to be able to translate easily from one proof format to another. The translation process may in principle be trivial in the sense that no search need be involved, but it's often highly nontrivial to design a good algorithm to do this and to prove that it always works.

A familiar format for presenting proofs is natural deduction. An example of a natural deduction proof for THM112 is in Figure 5-1. (See [4] or §30 of [7] for more details about this formulation of natural deduction.) Note how a natural deduction proof differs from building up the wff from a single tautology. In natural

Figure 4-3: EXPANSION TREE PROOF OF THM112

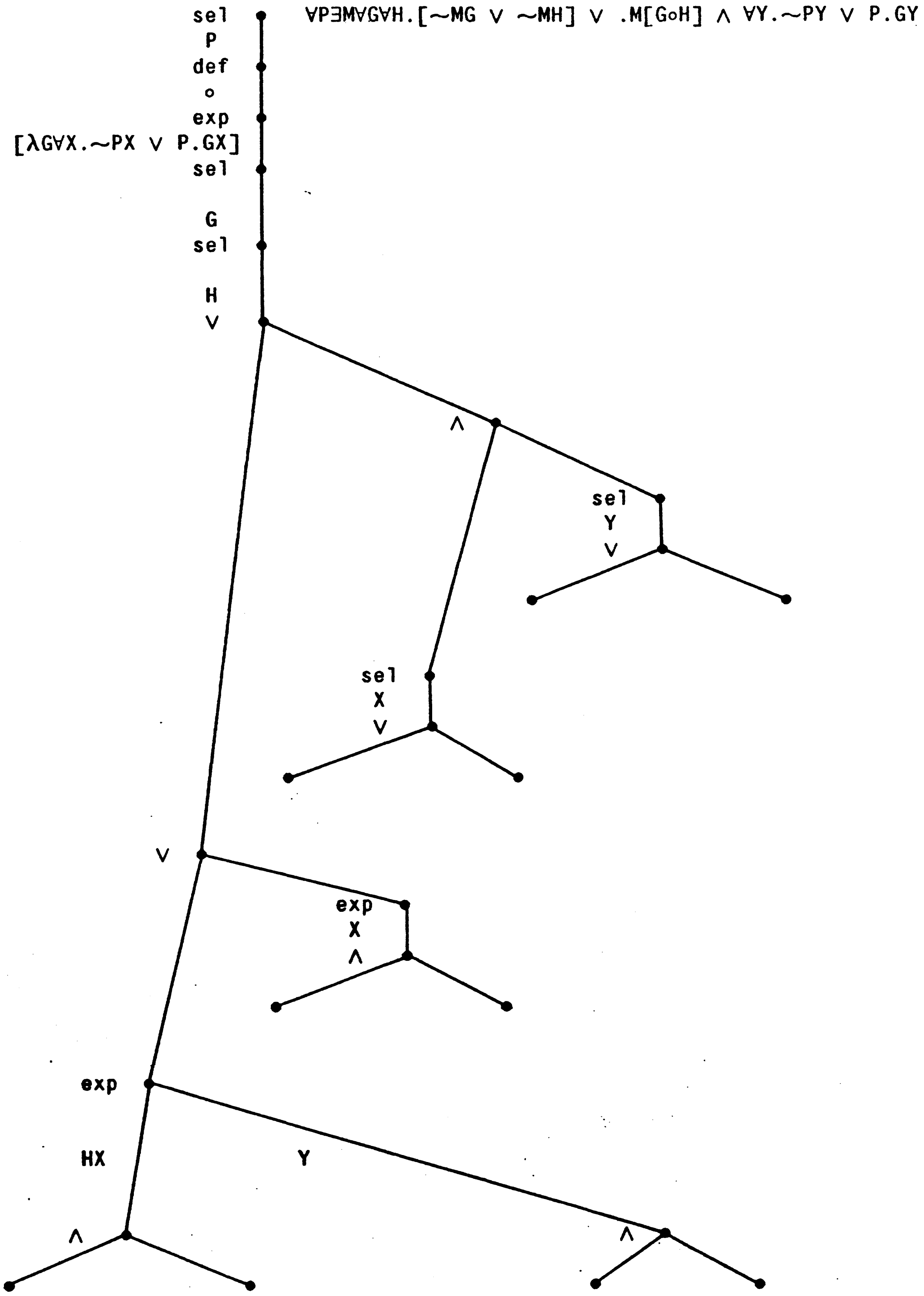


Figure 5-1: NATURAL DEDUCTION PROOF OF THM112

(1)1	$\vdash$	$\forall X_\iota\ [P_{o\iota}\ X \supset P.G_{\iota\iota}\ X] \land \forall X.P\ X \supset P.H_{\iota\iota}\ X$	Hyp
(2)1	$\vdash$	$\forall X_\iota.P_{o\iota}\ X \supset P.G_{\iota\iota}\ X$	Conj: 1
(3)1	$\vdash$	$\forall X_\iota.P_{o\iota}\ X \supset P.H_{\iota\iota}\ X$	Conj: 1
(4)1	$\vdash$	$P_{o\iota}\ [H_{\iota\iota}\ X_\iota] \supset P.G_{\iota\iota}.H\ X$	UI: [H X] 2
(5)1	$\vdash$	$P_{o\iota}\ X_\iota \supset P.H_{\iota\iota}\ X$	UI: X 3
(6)1	$\vdash$	$P_{o\iota}\ X_\iota \supset P.G_{\iota\iota}.H_{\iota\iota}\ X$	RuleP: 4 5
(7)1	$\vdash$	$\forall X_\iota.P_{o\iota}\ X \supset P.G_{\iota\iota}.H_{\iota\iota}\ X$	UGen: X 6
(8)1	$\vdash$	$P_{o\iota}\ Y_\iota \supset P.G_{\iota\iota}\ Y$	UI: Y 2
(9)1	$\vdash$	$\forall Y_\iota.P_{o\iota}\ Y \supset P.G_{\iota\iota}\ Y$	UGen: Y 8
(10)1	$\vdash$	$\forall X_\iota\ [P_{o\iota}\ X \supset P.G_{\iota\iota}.H_{\iota\iota}\ X] \land \forall Y_\iota.P\ Y \supset P.G\ Y$	Conj: 7 9
(11)	$\vdash$	$\forall X_\iota\ [P_{o\iota}\ X \supset P.G_{\iota\iota}\ X] \land \forall X\ [P\ X \supset P.H_{\iota\iota}\ X]$ $\supset \forall X\ [P\ X \supset P.G.H\ X] \land \forall Y_\iota.P\ Y \supset P.G\ Y$	Deduct: 10
(12)	$\vdash$	$\forall H_{\iota\iota}.\quad \forall X_\iota\ [P_{o\iota}\ X \supset P.G_{\iota\iota}\ X] \land \forall X\ [P\ X \supset P.H\ X]$ $\supset \forall X\ [P\ X \supset P.G.H\ X] \land \forall Y_\iota.P\ Y \supset P.G\ Y$	UGen: H 11
(13)	$\vdash$	$\forall G_{\iota\iota}\ \forall H_{\iota\iota}.\quad \forall X_\iota\ [P_{o\iota}\ X \supset P.G\ X] \land \forall X\ [P\ X \supset P.H\ X]$ $\supset \forall X\ [P\ X \supset P.G.H\ X] \land \forall Y_\iota.P\ Y \supset P.G\ Y$	UGen: G 12
(14)	$\vdash$	$\forall G_{\iota\iota}\ \forall H_{\iota\iota}.\quad [\lambda G\ \forall X_\iota.P_{o\iota}\ X \supset P.G\ X]\ G \land [\lambda G\ \forall X.P\ X \supset P.G\ X]\ H$ $\supset [\lambda G\ \forall X.P\ X \supset P.G\ X][\lambda Z_\iota\ G.H\ Z] \land \forall Y_\iota.P\ Y \supset P.G\ Y$	Lambda: 13
(15)	$\vdash$	$\exists M_{o(\iota\iota)}\ \forall G_{\iota\iota}\ \forall H_{\iota\iota}.\ M\ G \land M\ H \supset M\ [\lambda Z_\iota\ G.H\ Z] \land \forall Y_\iota.\ P_{o\iota}\ Y \supset P.G\ Y$	EGen: $[\lambda G\ \forall X_\iota.P\ X \supset P.G\ X]$ 14
(16)	$\vdash$	$\exists M_{o(\iota\iota)}\ \forall G_{\iota\iota}\ \forall H_{\iota\iota}.\ M\ G \land M\ H \supset M\ [G \circ H] \land \forall Y_\iota.\ P_{o\iota}\ Y \supset P.G\ Y$	Def: 15
(17)	$\vdash$	$\forall P_{o\iota}\ \exists M_{o(\iota\iota)}\ \forall G_{\iota\iota}\ \forall H_{\iota\iota}.\ M\ G \land M\ H \supset M\ [G \circ H] \land \forall Y_\iota.\ P\ Y \supset P.G\ Y$	UGen: P 16

deduction proofs we may have hypotheses introduced and eliminated, conjunctions introduced and eliminated, proofs by cases, indirect proofs, the transitive law of implication, etc. In general, a truly "natural" deduction breaks up the proof and the theorem into parts on which we can focus.

As you can see, a natural deduction proof contains a great deal of redundancy, since certain formulas occur as subformulas of many lines of the proof. Redundancy occurs naturally in human discourse, since our short-term memories have limited capacity, and there are limitations on the ways we can restructure information in our heads. It's quite appropriate that proofs in natural deduction style also contain this redundancy. However, in a computer system redundancy is often a nuisance or a problem. It wastes memory, and may create the need for additional processing. When we see redundancy, we should ask whether there is some good reason for it. What does it cost, and what does it achieve? Of course, redundancy in the final proof format should be distinguished from redundancies and inefficiencies in the search for the proof and associated data structures. The latter are much more serious.

When one is *searching* for a proof, it is desirable to work within a context where one can focus on the essential features of the problem as directly and economically as possible. Thus, it makes a lot of sense to search for ways of proving theorems of higher-order logic by searching for expansion proofs.

Miller showed [25] [26] [27] that once an expansion proof has been found, it can be converted without further search into a natural deduction proof. This extended related work for first-order logic in [4] and [12]. Of course, being able to make this translation justifies searching for a proof by looking for an expansion proof.

Some choices have to be made to determine how the natural deduction proof is arranged, since different natural deduction proofs may correspond to the same expansion proof, and heuristics can be used to make these choices in ways that will tend to produce stylistically pleasing proofs. Miller has done some work [25] under the name *focusing* on this problem too, but more needs to be done.

Of course, when we are trying to construct examples of proofs interactively for research purposes, it's usually easiest for us to construct them first in natural deduction style, so we'd like to be able to translate such proofs automatically into expansion proofs. However, there are some pitfalls when one tries to translate in this direction.

When Herbrand was proving the famous theorem which bears his name, he needed to show that every theorem of first-order logic has what we now call a tautologous Herbrand expansion. He presented an apparently straightforward inductive argument that every formula in any proof in a certain system of first-order logic has such an expansion. However, he ran into trouble [20] when he tried to prove that if A and [A ⊃ B] both have tautologous Herbrand expansions, then B has one too. This error was not repaired [21] for thirty years.

In higher-order logic, a purely syntactic proof that every natural deduction proof can be translated into an expansion proof runs up against Gödel's Second Incompleteness Theorem, just as a purely syntactic proof of cut-elimination in type theory does. To explain this, let us introduce an axiom of infinity (a sentence whose models are all infinite) which we shall call Infin:

$$\exists R_{o\iota\iota}\, \forall X_\iota\, \forall Y_\iota\, \forall Z_\iota .\ \exists W_\iota\, R\, X\, W \wedge \sim R\, X\, X \wedge .\!\sim R\, X\, Y \vee \sim R\, Y\, Z \vee R\, X\, Z$$

"Analysis" is the name for the logical system of type theory with added axioms of extensionality, descriptions, and Infin. Most of mathematics can be formalized in this system. Now let us suppose for a moment that analysis is inconsistent. It can be shown by a purely syntactic argument [2] that if analysis is inconsistent, one can prove ~Infin in type theory without axioms of extensionality and descriptions. Now for the sake of simplicity, consider a stronger axiom of infinity which we shall call SkInfin:

$$\exists G_{\iota\iota}\, \exists R_{o\iota\iota}\, \forall X_\iota\, \forall Y_\iota\, \forall Z_\iota .\ R\, X\, [G\, X] \wedge \sim R\, X\, X \wedge .\ \sim R\, X\, Y \vee \sim R\, Y\, Z \vee R\, X\, Z$$

SkInfin implies Infin, so if analysis is inconsistent, ~SkInfin can be proved in type theory. If every proof in type theory can be translated into an expansion proof, there must be a dual expansion tree for SkInfin which has a contradictory deep formula. Here is the vpform for SkInfin:

```
∃GR
┌∀XYZ                                  ┐
│ ┌          R X [G X]                ┐│
│ │                                   ││
│ │           ~ R X X                 ││
│ │                                   ││
└ └~ R X Y v ~ R Y Z v R X Z          ┘┘
```

Let's write λ(A) for the (essentially unique) λ-normal form [1] of a wff A. It is not hard to see that the deep formula must be a conjunction of formulas which will be true if we make the atom [R A B] true whenever the number of occurrences of G in λ(A) is less than the number of occurrences in λ(B), so it cannot be contradictory. Thus, we reach a contradiction.

If the proof that the translation process above always works could be formalized in analysis, we could formalize the argument above to obtain a proof within analysis that analysis is consistent, contradicting Gödel's Second Theorem. Thus, the proof that the translation process always works cannot be formalized in analysis, and the problem of proving that there is such a translation is intrinsically difficult.

Frank Pfenning showed in [28] how to translate a natural deduction proof in first-order logic into an expansion proof, and deals with the problem for higher-order logic in his thesis [29]. Pfenning can prove that his translation for type theory works when applied to cut-free proofs, and it's been known for a long time (by a non-constructive argument) that natural deduction proofs in type theory can be made cut-free. Pfenning has also defined a translation process for proofs which do contain cuts, but he has no proof that the translation process terminates. (Of course, any such proof would have to use methods which could not be formalized in analysis; for examples of such proofs, see references cited in [2].)

It is our thesis that proofs of formulas of first- and higher-order logic in virtually any format induce expansion proofs, and can be constructed from expansion proofs. Frank Pfenning has amassed considerable evidence for this in his thesis, and has shown how expansion trees can be extended to accommodate equality and extensionality. Expansion trees are important tools for comparing proofs and methods of searching for proofs. An expansion proof reveals how a tautology is hidden in its theorem, and provides a nice answer to the question "What is the essential reason that this wff is valid?"

An expansion proof embodies essential ideas which can be expressed in many ways in different proofs; the expansion proof is a key to translating between them. If we have a clumsy natural deduction proof, we can transform it into the expansion proof which it induces and then (if we have sufficiently good translation methods) translate that back into a well structured natural deduction proof.

Obviously, proofs which induce the same expansion proofs are equivalent in a very important and fundamental sense. It's conceivable that by looking even further in this direction of abstracting the essential ideas out of theorems we can eventually devise a significant scheme for syntactically classifying theorems. At present we have no sensible way to organize a universal library of theorems which have been proved, or considered. Mathematicians discover from time to time that they have worked hard to prove theorems which others proved earlier.

6. A challenge to this point of view

Of course, we mustn't let our zeal in pursuit of truth blind us to the fact that truth is complicated and elusive. Let me digress a moment to discuss a challenge to the point of view that expansion proofs provide the essential reasons why wffs are valid.

Once one has a proof in virtually any format, one can almost always construct another proof of the same theorem by adding certain redundancies and irrelevancies, and this is certainly true of expansion proofs. However, quite aside from this, one cannot ignore the fact that certain theorems have several fundamentally different expansion proofs, which express fundamentally different ideas.

Examples of two proofs of a theorem I call THM76 are in Figures 6-1 and 6-2. Do we regard such a theorem as two different theorems which happen to manifest themselves as the same wff, like multiple roots of an equation? One can start with two different tautologies, disguise them in different ways, and somehow end up with the same wff. This seems a remarkable coincidence.

Figure 6-1: First proof of THM76

(1)	1	$\vdash \forall P_{o\iota} .P\ Y_\iota \supset P\ X_\iota$	Hyp
(2)	1	$\vdash [\lambda Z_\iota .\sim R_{o\iota}\ Z]\ Y_\iota \supset [\lambda Z .\sim R\ Z]\ X_\iota$	UI: $[\lambda Z.\sim R\ Z]$ 1
(3)	1	$\vdash \sim R_{o\iota}\ Y_\iota \supset \sim R\ X_\iota$	Lambda: 2
(4)	1	$\vdash R_{o\iota}\ X_\iota \supset R\ Y_\iota$	RuleP: 3
(5)	1	$\vdash \forall R_{o\iota} .R\ X_\iota \supset R\ Y_\iota$	UGen: R 4
(6)		$\vdash \forall P_{o\iota}\ [P\ Y_\iota \supset P\ X_\iota] \supset \forall R_{o\iota} .R\ X \supset R\ Y$	Deduct: 5

Figure 6-2: Second proof of THM76

(1)	1	$\vdash \forall P_{o\iota} .P\ Y_\iota \supset P\ X_\iota$	Hyp
(2)	1	$\vdash [\lambda Z_\iota .R_{o\iota}\ Z \supset R\ Y_\iota]\ Y \supset [\lambda Z .R\ Z \supset R\ Y]\ X_\iota$	UI: $[\lambda Z.R\ Z \supset R\ Y]$ 1
(3)	1	$\vdash R_{o\iota}\ Y_\iota \supset R\ Y \supset .R\ X_\iota \supset R\ Y$	Lambda: 2
(4)	1	$\vdash R_{o\iota}\ X_\iota \supset R\ Y_\iota$	RuleP: 3
(5)	1	$\vdash \forall R_{o\iota} .R\ X_\iota \supset R\ Y_\iota$	UGen: R 4
(6)		$\vdash \forall P_{o\iota}\ [P\ Y_\iota \supset P\ X_\iota] \supset \forall R_{o\iota} .R\ X \supset R\ Y$	Deduct: 5

Maybe someday we will have a much deeper perspective which will allow us to approach automated theorem proving on a more abstract conceptual level. At present, we really don't have this, so let's turn to the question of how to prove theorems by searching for expansion proofs.

7. Introduction to searching for expansion proofs

The fundamental concepts of expansion proofs and connections are important basic tools which we can use in fashioning a general approach to automated theorem proving in higher-order logic. If one wants to construct an expansion proof for a theorem by working down from the theorem to a tautology, or from its negation to a contradiction, one has certain operations which can be applied. Ideally, the process of applying them should be guided not by blind search, but by an understanding of what makes a formula tautological or contradictory. Let's consider an example which will illustrate the general procedure of applying the operations. We'll try to provide some motivation for applying the operations the way we do, but considerable research is needed to devise a purely automatic procedure which would apply the operations this way reasonably early in its search process. Our purpose here is simply to illustrate the kind of syntactic analysis which is the basis for the kind of search procedure we have in mind.

We'll work with the dual tree, start with the negation of the theorem to be proven, and create an expansion proof with a contradictory deep formula. We shall represent the process of gradually creating the expansion proof by a sequence of vpforms, which show how the negation of the theorem can gradually be expanded into the contradictory deep formula. The reader will easily see how this sequence can be regarded as representing an expansion tree which evolves as we make expansions and selections, instantiate definitions, and substitute for variables which we introduced earlier.

Recall that THM112 is $\forall P_{o\iota}\, \exists M_{o(\iota\iota)}\, \forall G_{\iota\iota}\, \forall H_{\iota\iota}.\ M\,G \wedge M\,H \supset .\, M\,[G \circ H] \wedge \forall Y_{\iota}.P\,Y \supset P.G\,Y$.

~ THM112 is:

```
∃P
┌∀M                                                    ┐
│┌∃GH                                                 ┐│
││┌                  M G                             ┐││
│││                                                  │││
│││                  M H                             │││
│││                                                  │││
│││                      ∃Y                          │││
│││~ M [  G  o  H  ]      ┌    P Y     ┐             │││
│││                    v  │            │             │││
└└└                       └~ P [G Y]   ┘             ┘┘┘
```

Select P:

```
∀M
┌∃GH                                                ┐
│┌                  M G                             ┐│
││                                                  ││
││                  M H                             ││
││                                                  ││
││                               ∃Y                 ││
││~ M [  G  ∘  H  ]           ┌     P Y    ┐       ││
││                      v     │            │       ││
└└                            └~ P [G Y]   ┘       ┘┘
```

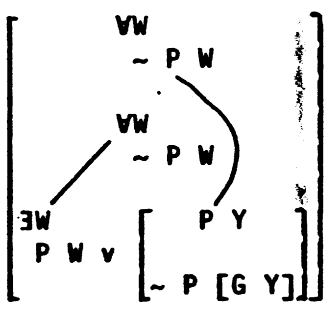
Since the head letters are P and M, and P is a constant, we must use the expansion operator which instantiates the quantifier ∀M. We need to span the vertical path on the right. If Y did not depend on M, we could instantiate M with $[\lambda F_{u} .\sim P_{o\iota} Y_{\iota}]$. This would create connections on both paths. Since Y does depend on M, we introduce a quantifier which we can instantiate after selecting Y by expanding on $M_{o(u)}$ with $[\lambda F_{u} \forall W_{\iota} .\sim P_{o\iota} W]$. We thus obtain:

```
∃GH
┌                  [λF ∀W.~ P W] G                        ┐
│                                                         │
│                  [λF ∀W.~ P W] H                        │
│                                                         │
│                                        ∃Y               │
│~ [λF ∀W.~ P W] [  G  ∘  H  ]          ┌    P Y    ┐     │
│                                  v    │           │     │
└                                       └~ P [G Y]  ┘     ┘
```

λ-normalize and select G, H, and Y. Note that these variables depend on M, but the term we substituted for M contains no free variables.

```
┌        ∀W                  ┐
│         ~ P W              │
│                            │
│        ∀W                  │
│         ~ P W              │
│                            │
┌∃W      ┌      P Y    ┐     ┐
│ P W v  │               │     │
└        └~ P [G Y]    ┘     ┘
```

In an expansion tree a connection is a pair of nodes of the tree, and the shallow formulas of these nodes need not be literals. We shall portray matings simply by drawing lines between formulas to represent the connections between the corresponding nodes. Note that the mating indicated by the lines in the vpform below spans both vertical paths through this wff.



Now we instantiate the top quantifier as dictated by the mating, i.e., by the substitution which makes mated pairs complementary.

```
[        ~ P Y              ]
[                           ]
[        ∀W                 ]
[         ~ P W             ]
[                           ]
[∃W         [      P Y    ] ]
[ P W ∨     [             ] ]
[           [~ P [G Y]    ] ]
```

We have reached a contradiction. Note that this is a correct proof, since in each case $[\lambda F_{\iota\iota}\ \forall W_{\iota} . \sim P_{o\iota}\ W]$ is a set of functions from P into P which is closed under composition. If P is empty, this is the set of all functions of the appropriate type; if P is nonempty, it is the empty set of functions.

Of course, the proof just given is somewhat trivial, so let's look for another proof. We return to the stage where we were ready to expand on ∀M.

```
∀M
 [∃GH                                               ]
 [ [                M G                           ] ]
 [ [                                              ] ]
 [ [                M H                           ] ]
 [ [                                              ] ]
 [ [                          ∃Y                  ] ]
 [ [~ M [  G  o  H  ]         [     P Y       ]   ] ]
 [ [                    ∨     [               ]   ] ]
 [ [                          [~ P [G Y]      ]   ] ]
```

This time let's try to mate the literal MG with the entire wff on the lower right. Consideration of the negation of this wff suggests that we instantiate ∀M with the term $[\lambda F_{\iota\iota}\ \forall W_{\iota} . \sim P_{o\iota}\ W \vee P.G\ W]$. However, we cannot quite do this, since G depends on M, so we use the term $[\lambda F_{\iota\iota}\ \forall W_{\iota} . \sim P_{o\iota}\ W \vee P.F\ W]$. Of course, this wff denotes the set of functions which map P into P; thus, this connection between parts of the wff has led us rather directly to an expansion term which contains the key idea in the most obvious proof of this theorem. When we make this expansion and λ-normalize we obtain:

```
∃GH
[           ∀W                                    ]
            [~ P W v P [G W]]

            ∀W
            [~ P W v P [H W]]

 ∃W                                   ∃Y
 [[            P W                 ]     [       P Y   ]]
 [[                                ]  v  [             ]]
 [[~ P [[  G  ∘  H   ] W]          ]     [~ P [G Y]    ]]
```

We rename some variables and select G, H, X, and Y:

```
[           ∀U                                    ]
            [~ P U v P [G U]]

            ∀W
            [~ P W v P [H W]]

[[            P X              ]     [       P Y   ]]
[[                             ]  v  [             ]]
[[~ P [[  G  ∘  H   ] X]       ]     [~ P [G Y]    ]]
```

Although our intuition tells us that we have already incorporated the expansion term representing the key idea for our proof into the expansion tree, there is still some work to be done. We can clearly mate the literals containing U with those containing Y in such a way as to span the paths passing through the wff on the lower right. We might try to mate [P X] with [~ P W], but this leaves some paths unspanned. We note that the lower-left literal is not yet being used in this mating. To try to find a mate for it, we need to duplicate one of the top two lines. We do this by quantifier duplication, which is really just the introduction of a new variable as an expansion term for the node of the expansion tree which corresponds to the quantifier. At this point the vpform we display will depart a little from the expansion tree it represents, since we'll display two quantifiers where there's really only one in the expansion tree. However, it's convenient to display quantifiers for the variables to keep track of where and how the variables are quantified. After duplicating the quantifier ∀U and instantiating the definition we obtain:

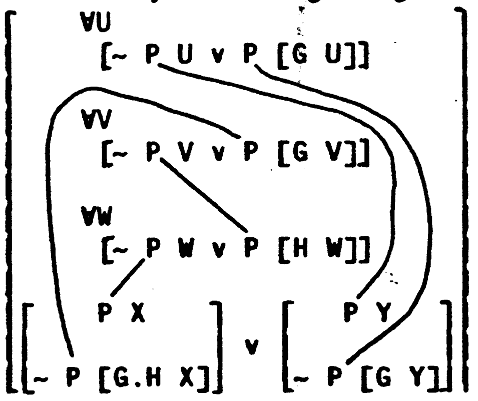
```
[     ∀U                               ]
      [~ P U v P [G U]]

      ∀V
      [~ P V v P [G V]]

      ∀W
      [~ P W v P [H W]]

[[      P X      ]     [      P Y    ]]
[[               ]  v  [             ]]
[[~ P [G.H X]    ]     [~ P [G Y]    ]]
```

Now let's try the following mating:



Instantiate the remaining universal quantifiers using the substitutions dictated by the mating:

```
┌       ~ P Y v P [G Y]          ┐
│                                │
│    ~ P [H X] v P [G.H X]       │
│                                │
│       ~ P X v P [H X]          │
│                                │
│┌    P X      ┐   ┌     P Y    ┐│
││             │ v │            ││
│└~ P [G.H X]  ┘   └~ P [G Y]   ┘│
```

This wff is contradictory.

We now have another example of a theorem with two proofs. This example illustrates a general method for constructing such theorems: assert the existence of some sort of entity, where many examples exist.

Of course, THM112 is fairly trivial, and we can't expect to guess the right expansion terms so easily in general. Nevertheless, this example illustrates the fact that the basic processes involved in constructing an expansion proof are expanding the formula and searching for an acceptable mating. The formula can be expanded by duplicating quantifiers and by instantiating quantifiers on higher-order variables. The matingsearch process involves building up a spanning set of connections which is *compatible*, i.e., a set for which there is a substitution which simultaneously makes each connection a complementary pair.

Connections, or matings, arise naturally as one seeks to find the tautology buried in a theorem. One knows that once one has a tautology, certain literals will be complementary. Once one has enough copies of all the appropriate literals and one decides which literals should be complementary, one can systematically generate the substitutions which will make them so by applying a unification algorithm. In working back to the tautology, the connections generate the appropriate substitutions.

8. General considerations about the search process

Let us consider certain aspects of the problem of searching for expansion proofs, and make some general suggestions about the design of a computer system for finding such proofs. Our purpose here is not to discuss the details of any existing system, but to bring relevant problems and questions into focus. We start by considering the kind of computer environment which will facilitate the process.

Since many difficult mathematical questions can be expressed as formulas of higher-order logic, it is clear that a useful theorem proving system in this realm will allow for human input, so it should be able to operate in a mixture of automatic and interactive modes. The system should be able to stop and restart gracefully, permitting human intervention and interaction in the process. It should be possible to instantiate definitions whenever this seems appropriate. To facilitate interaction, facilities should be available for displaying information and partial proofs in a variety of formats, and translating from one to another. The user should be able to control the level of detail that is displayed on the screen and recorded in various files.

A search procedure which is in principle complete will achieve early success only on relatively simple problems, and should be augmented by general heuristic search processes. We will never finish searching for, and experimenting with, heuristics to guide the decisions about what steps to take, so the system should be able to accommodate a variety of heuristics, and readily incorporate new heuristics. We will have both absolute and heuristic prescriptions and restrictions. The absolute ones are justified by metatheorems, the heuristic ones by our experience and intuition. The heuristic ones may conflict with each other, and we will need heuristic principles for resolving these conflicts.

The search for an expansion proof in higher-order logic may involve a number of simultaneous unbounded searches, so it will be particularly appropriate to use multiprocessors as suitable ones become available. (Note [15].) The search process should deal with data structures which are modified in systematic ways by processors which can work independently and interact in controlled and fruitful ways under the direction of a master processor. (For example, we should be able to expand the wff at any time and continue the matingsearch.)

We would really like to organize the search process so that we can imagine we are moving about in some well defined infinite search space. Our motion can be generated by arbitrary heuristics, and we always know what parts of the space we've already explored. We have methods for keeping track of parts of the space we need not explore because of things we have learned from earlier explorations. Of course, we would like to express all this information very concisely, and be able to access it very quickly and flexibly. Obviously, there are problems in designing such a system.

9. Matingsearch

The matingsearch process, which attempts to find an acceptable mating of the current formula, can be carried on by several processors working simultaneously.

One processor maintains the *connection graph* for the current formula, which is the set of connections which are candidates for inclusion in the mating. In order to be in the connection graph, a pair of nodes must share some vertical path, and must have the potential of becoming complementary under some substitution. Associated with each connection is the currently known information about its unification problem. Connections may be removed from the connection graph if they are eventually discovered to be incompatible.

Unification problems associated with sets of connections will be analyzed by processors which apply Huet's unification algorithm [23]. Ultimately the unification processors should accommodate any constraints on unification terms which the user wishes to impose and recognize special cases and certain unification problems where processing is guaranteed to terminate (such as those of first-order logic). There should be a special processor to recognize when certain nodes in the unification search tree are subsumed by others. (Such subsumptions actually occur with surprising frequency.)

Certain processors, which we call *matingsearch processors*, will build up sets of connections from the connection graph in an attempt to construct an acceptable mating. Unless an acceptable mating is quickly found, most of the useful information acquired in this process concerns what will *not* work, i.e., which sets of connections are incompatible. A number of matingsearch processors can profitably operate simultaneously if each contributes to, and makes use of, a collection of incompatible sets of connections which we shall call the *failure record*. When the formula is expanded, the matingsearch processors may have to reexamine what must be done to achieve an acceptable mating, but the information embodied in the failure record remains useful.

The failure record can be constantly augmented not only by incompatibilities discovered by the matingsearch processors, but also by analyses of symmetries in the formula and other methods. Symmetries may occur in the original theorem, and are always introduced when quantifiers are duplicated. They can cause significant redundancy in the search process if they are ignored.

Ideally, one of the matingsearch processors will apply a search procedure which is in principle complete, but others may apply heuristic procedures which are incomplete but likely to achieve success more quickly in favorable cases. Matingsearch heuristics can be based on the need to establish connections between nodes in such a way that an acceptable mating can eventually be achieved, and to discover incompatibilities quickly.

10. Expansion

Let's consider the problem of expanding a wff (i.e., introducing or changing expansion terms in such a way as to enlarge the deep formula of the expansion tree). In a multiprocessor environment, one or more processors could work on this while others work on matingsearch.

One way of expanding a wff is by quantifier duplication. As mentioned above, this operation involves adding an expansion term consisting of a new variable at an expansion node. (Of course, by a new variable we mean one that does not already occur in any wff associated with the expansion tree.) In first-order logic this is the only expansion operation that is necessary; once we have duplicated the quantifiers sufficiently, all we need to do (in principle) is to make an exhaustive search of the possible matings, applying the unification algorithm to generate the required substitution terms. In higher-order logic, however, quantifier duplication does not suffice. Indeed, the need to expand the formula by instantiating quantifiers with terms which cannot be generated by unification of existing subformulas is one of the vexing problems which distinguishes higher-order logic from first-order logic.

Other methods of expanding a wff in higher-order logic involve making substitutions for higher-order variables, so let's take a moment to consider what we mean by applying a substitution to a variable in an expansion tree. We shall assume that the shallow formula of the expansion tree contains no free variables. Also, since we wish to regard symbols as variables only if it is appropriate to substitute for them, we shall regard selected parameters as constants, although Miller calls them selected variables. Thus, free variables can be introduced in an expansion tree only in expansion terms, and if we compute all deep and shallow formulas of nodes (except for the shallow formula of the root) from the expansion terms and selected parameters, we can substitute for a free variable simply by substituting for its free occurrences in expansion terms.

Of course, we shall always wish to work with expansion trees in which selections are done right, so we assume without further discussion that new expansion terms will be introduced and substitutions will be applied in such a way as to guarantee this. If Skolemization is used this is generally automatic, but if it is not one may need to rename certain selected parameters and forbid certain substitutions so that the imbedding relation remains irreflexive.

Naturally, one can generate expansion terms incrementally by using substitutions which introduce single connectives and quantifiers in a general way, thus achieving the effects of Huet's splitting rules [22]. It turns out that in the context of the search process we envision, projections such as those used in [23] are also needed. We shall refer to such substitutions for a single variable as *primitive substitutions*. Examples of primitive substitutions for certain variables may be seen in Figures 10-1, 10-2, 10-3, and 10-4, where the terms to be substituted for the variable are given. Note that they are determined by the types of the variables for which the substitution is to be made. A substitution term has the form $\lambda x^1 ... \lambda x^n B_o$; we shall refer to the wff B_o as its *body*.

Figure 10-1: Primitive substitutions for $R_{o\alpha(o\beta)(o\alpha\xi)}$:

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}.\sim\ R^{1}_{o\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}.\ \ R^{2}_{o\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z\ \land\ R^{3}_{o\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}.\ \ R^{4}_{o\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z\ \lor\ R^{5}_{o\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}\ \exists W_{\varphi}\ R^{6}_{o\varphi\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z\ W$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}\ \forall W_{\varphi}\ R^{7}_{o\varphi\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z\ W$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}\ Y.R^{8}_{\beta\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ \lambda Z_{\alpha}\ X\ [R^{9}_{\xi\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z].R^{10}_{\alpha\alpha(o\beta)(o\alpha\xi)}\ X\ Y\ Z$

$\lambda X_{o\alpha\xi}\ \lambda Y_{o\beta}\ X.R^{11}_{\xi(o\beta)(o\alpha\xi)}\ X\ Y$

Before discussing primitive substitutions further, let's see how THM112 can be proved quite systematically by using them. We shall ignore all the incorrect choices which a real search procedure would probably make before finding the correct ones.

Recall that THM112 is:

$\forall P_{o\iota}\ \exists M_{o(\iota\iota)} \forall G_{\iota\iota} \forall H_{\iota\iota}.\ M\,G \land M\,H \supset M\,[G \circ H] \land \forall Y_{\iota}.P\,Y \supset P.G\,Y$
THM112 after negating and selecting P:

```
∀M
┌∃GH                                             ┐
│┌                    M G                       ┐│
││                                              ││
││                    M H                       ││
││                                              ││
││                               ∃Y             ││
││~ M [  G  ∘  H  ]           ┌    P Y       ┐ ││
││                         v  │              │ ││
└└                            └~  P [G Y]    ┘ ┘┘
```

Figure 10-2: Primitive substitutions for $M_{o(u)}$:

$\lambda F_{u} . \sim M^{1}_{o(u)} \; F$

$\lambda F_{u} . M^{2}_{o(u)} \; F \wedge M^{3}_{o(u)} \; F$

$\lambda F_{u} . M^{4}_{o(u)} \; F \vee M^{5}_{o(u)} \; F$

$\lambda F_{u} \; \exists W_{\delta} \; M^{1}_{o\delta(u)} \; F \; W$

$\lambda F_{u} \; \forall W_{\delta} \; M^{1}_{o\delta(u)} \; F \; W$

Special case of the above:

$\lambda F_{u} \; \forall W_{\iota} \; M^{1}_{o\iota(u)} \; F \; W$

Expand on $M_{o(u)}$ with the primitive substitution term $\lambda F_{u} \forall W_{\iota} M^{1}_{o\iota(u)} F W$:

$$\exists GH \left[\begin{array}{l} \lbrack \lambda F \;\; \forall W \;\; M1 \;\; F \;\; W \rbrack \; G \\ \lbrack \lambda F \;\; \forall W \;\; M1 \;\; F \;\; W \rbrack \; H \\ \sim \lbrack \lambda F \;\; \forall W \;\; M1 \;\; F \;\; W \rbrack \lbrack \; G \circ H \; \rbrack \vee \exists Y \left[\begin{array}{l} P \; Y \\ \sim P \lbrack G \; Y \rbrack \end{array}\right] \end{array}\right]$$

λ-normalize:

$$\exists GH \left[\begin{array}{l} \forall W \\ \quad M1 \; G \; W \\ \forall W \\ \quad M1 \; H \; W \\ \exists W \\ \sim M1 \lbrack \; G \circ H \; \rbrack \; W \vee \exists Y \left[\begin{array}{l} P \; Y \\ \sim P \lbrack G \; Y \rbrack \end{array}\right] \end{array}\right]$$

Figure 10-3: Primitive substitutions for $M^1_{o\iota(u)}$:

$\lambda F_u\ \lambda W_\iota .\sim\ M^2_{o\iota(u)}\ F\ W$

$\lambda F_u\ \lambda W_\iota . M^2_{o\iota(u)}\ F\ W \wedge M^3_{o\iota(u)}\ F\ W$

$\lambda F_u\ \lambda W_\iota . M^2_{o\iota(u)}\ F\ W \vee M^3_{o\iota(u)}\ F\ W$

$\lambda F_u\ \lambda W_\iota\ \exists U_\epsilon\ M^2_{o\epsilon\iota(u)}\ F\ W\ U$

$\lambda F_u\ \lambda W_\iota\ \forall U_\epsilon\ M^2_{o\epsilon\iota(u)}\ F\ W\ U$

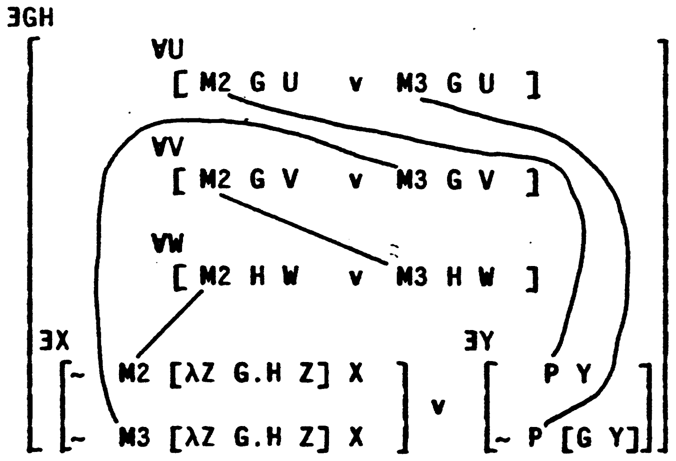
Expand the formula again by substituting for $M^1_{o\iota(u)}$ the primitive substitution term $[\lambda F_u\ \lambda W_\iota . M^2_{o\iota(u)}\ F\ W \ \vee M^3_{o\iota(u)}\ F\ W]$ and λ-normalize:

```
∃GH
┌             ∀W                                 ┐
│              [ M2 G W   v   M3 G W ]           │
│                                                │
│             ∀W                                 │
│              [ M2 H W   v   M3 H W ]           │
│                                                │
│∃W                               ∃Y             │
││┌~   M2 [  G  ∘  H  ] W ┐       ┌    P Y    ┐  │
│││                       │  v    │           │  │
└└~   M3 [  G  ∘  H  ] W ┘       └~ P [G Y]  ┘  ┘
```

Next we shall duplicate a quantifier, rename bound variables, and eliminate the definition.

```
∃GH
┌          ∀U                               ┐
│           [ M2 G U   v  M3 G U  ]         │
│                                           │
│          ∀V                               │
│           [ M2 G V   v  M3 G V  ]         │
│                                           │
│          ∀W                               │
│           [ M2 H W   v  M3 H W  ]         │
│                                           │
│∃X                            ∃Y           │
││┌~  M2 [λZ G.H Z] X  ┐       ┌    P Y   ┐ │
│││                    │  v    │          │ │
└└~  M3 [λZ G.H Z] X  ┘       └~ P [G Y] ┘ ┘
```

Now consider the following mating:



Let us first make the substitutions for M2 and M3 dictated by the mating above.

```
∃GH
┌ ∀U                                                                    ┐
│  [[λF λZ.~ P Z] G U    v  [λF λZ P.F Z] G U  ]                        │
│                                                                       │
│ ∀V                                                                    │
│  [[λF λZ.~ P Z] G V    v  [λF λZ P.F Z] G V  ]                        │
│                                                                       │
│ ∀W                                                                    │
│  [[λF λZ.~ P Z] H W    v  [λF λZ P.F Z] H W  ]                        │
│                                                                       │
│∃X                                                   ∃Y                │
│ ┌~ [λF λZ.~ P Z] [λZ G.H Z] X  ┐     ┌    P Y    ┐                   │
│ │                              │  v  │           │                   │
└ └~ [λF λZ P.F Z] [λZ G.H Z] X  ┘     └~ P [G Y]  ┘                   ┘
```

Note that the terms substituted for M2 and M3 contained no free variables, since P is a constant. Normalize and select G, H, X, and Y:

```
┌  ∀U                                      ┐
│   [~ P U   v P [G U  ]]                  │
│                                          │
│  ∀V                                      │
│   [~ P V   v P [G V  ]]                  │
│                                          │
│  ∀W                                      │
│   [~ P W   v P [H W  ]]                  │
│                                          │
│┌    P X       ┐   ┌    P Y   ┐           │
││              │ v │          │           │
└└~ P [G.H X   ]┘   └~ P [G Y] ┘           ┘
```

Now we expand on the universally quantified variables as dictated by the mating.

$$\begin{bmatrix} \sim P\ Y \lor P\ [G\ Y] \\ \sim P\ [H\ X\ ] \lor P\ [G.H\ X\ ] \\ \sim P\ X \lor P\ [H\ X\ ] \\ \begin{bmatrix} P\ X \\ \sim P\ [G.H\ X\ ] \end{bmatrix} \lor \begin{bmatrix} P\ Y \\ \sim P\ [G\ Y] \end{bmatrix} \end{bmatrix}$$

This wff is contradictory.

You might wonder why we list projections as primitive substitutions. Do we really need them? Let's have an example to show that projections are indeed needed.

THM104 is:

$\forall U_\alpha\ \forall V_\alpha.\mathrm{UNITSET}\ U = \mathrm{UNITSET}\ V \supset U = V$

The definition of UNITSET is $[\lambda W_\alpha\ \lambda Z_\alpha.Z = W]$. Thus $[\mathrm{UNITSET}\ W_\alpha]$, which might be abbreviated as $\{W_\alpha\}$, is $[\lambda Z_\alpha.Z = W]$, which denotes the set of all Z such that Z = W. The definition of = is $[\lambda X_\alpha\ \lambda Y_\alpha\ \forall Q_{o\alpha}.Q\ X \supset Q\ Y]$. The definition of UNITSET with the definition of = instantiated is $[\lambda W_\alpha\ \lambda Z_\alpha\ \forall Q_{o\alpha}.Q\ Z \supset Q\ W]$.

Nnf of the negation of THM104:

$$\exists U V \begin{bmatrix} \mathrm{UNITSET}\ U = \mathrm{UNITSET}\ V \\ \sim U = V \end{bmatrix}$$

Instantiate the definition of = :

$$\exists U V \begin{bmatrix} \forall P \\ \quad [\sim P\ [\mathrm{UNITSET}\ U] \lor P\ [\mathrm{UNITSET}\ V]] \\ \exists Q \begin{bmatrix} Q\ U \\ \sim Q\ V \end{bmatrix} \end{bmatrix}$$

Instantiate the definition of UNITSET:

```
∃UV
┌∀P                                    ┐
│ [~ P [λZ.Z = U] v P [λZ.Z = V]]      │
│                                      │
│              ∃Q                      │
│              ┌ Q U ┐                 │
│              │     │                 │
└              └~ Q V┘                 ┘
```

Instantiate the definition of =:

```
∃UV
┌∀P                                                  ┐
│ [~ P [λZ ∀R.R Z ⊃ R U] v P [λZ ∀S.S Z ⊃ S V]]      │
│                                                    │
│                        ∃Q                          │
│                        ┌ Q U ┐                     │
│                        │     │                     │
└                        └~ Q V┘                     ┘
```

Select U, V, and Q:

```
┌∀P                                                  ┐
│ [~ P [λZ ∀R.R Z ⊃ R U] v P [λZ ∀S.S Z ⊃ S V]]      │
│                                                    │
│                        Q U                         │
│                                                    │
└                        ~ Q V                       ┘
```

Since U, V, and Q are constants, there is nothing to do but expand on $P_{o(o\alpha)}$. Clearly, we cannot derive a contradiction from the top line alone (since it just says that [UNITSET U] = [UNITSET V]), so we must apply some operation Φ to the top line which will create some useful interaction between the top line and the two bottom literals. Note that modulo alphabetic changes of bound variables, the top line has the form [~LU ∨ LV]. Clearly Φ[~LU ∨ LV] must contain the literals ~QU and QV. Since [~LU ∨ LU] and [~LV ∨ LV] are useless tautologies, the result of applying Φ to either of these cannot be the same as Φ[~LU ∨ LV]. Therefore, the occurrences of U and V in the top line must actually contribute to the creation of at least one of the literals ~QU and QV. This cannot be accomplished by applying the primitive substitutions in Figure 10-4 which introduce quantifiers and connectives (possibly with iterations) and then applying the substitution induced by some mating. Thus, something else must be done. We apply the projection in Figure 10-4, and obtain the wff

$$[\sim [\lambda S_{o\alpha}\ S.P^{8}_{\alpha(o\alpha)}\ S]\ [\lambda Z_{\alpha}\ \forall R_{o\alpha}.R\ Z \supset R\ U_{\alpha}]$$
$$\vee\ [\lambda S\ S.P^{8}\ S].\lambda Z\ \forall S_{o\alpha}.S\ Z \supset S\ V_{\alpha}]\ \wedge\ .Q_{o\alpha}\ U \wedge \sim Q\ V$$

which we can λ-normalize to

Figure 10-4: Primitive substitutions for $P_{o(o\alpha)}$:

$\lambda S_{o\alpha} . \sim P^1_{o(o\alpha)}\ S$

$\lambda S_{o\alpha} . P^2_{o(o\alpha)}\ S \land P^3_{o(o\alpha)}\ S$

$\lambda S_{o\alpha} . P^4_{o(o\alpha)}\ S \lor P^5_{o(o\alpha)}\ S$

$\lambda S_{o\alpha}\ \exists X_\varepsilon\ P^6_{o\varepsilon(o\alpha)}\ S\ X$

$\lambda S_{o\alpha}\ \forall X_\varepsilon\ P^7_{o\varepsilon(o\alpha)}\ S\ X$

This one is a projection:

$\lambda S_{o\alpha}\ S . P^8_{\alpha(o\alpha)}\ S$

$$\begin{array}{l} [\ \sim \forall R_{o\alpha}\ [R\ [P^8_{\alpha(o\alpha)} . \lambda Z_\alpha\ \forall R . R\ Z \supset R\ U_\alpha] \supset R\ U] \\ \quad \lor \forall S_{o\alpha} . S\ [P^8 . \lambda Z\ \forall S . S\ Z \supset S\ V_\alpha] \supset S\ V] \land . Q_{o\alpha}\ U \land \sim Q\ V \end{array}$$

To abbreviate this formula we give the names M and N to the arguments of P^8. Note that these have no free variables, since U and V are constants. We now write the formula as

$[\sim \forall R_{o\alpha}\ [R\ [P^8_{\alpha(o\alpha)}\ M_{o\alpha}] \supset R\ U_\alpha] \lor \forall S_{o\alpha} . S\ [P^8\ N_{o\alpha}] \supset S\ V_\alpha] \land . Q_{o\alpha}\ U \land \sim Q\ V$

The nnf of this wff can be displayed as:

$$\left[\begin{array}{l} \exists R \\ \left[\begin{array}{l} R\ [P8\ M\ ] \\ \\ \sim R\ U \end{array}\right] \lor \begin{array}{l} \forall S \\ \ [\sim S\ [P8\ N\ ] \lor S\ V] \\ \end{array} \\ \\ \qquad\qquad\qquad Q\ U \\ \\ \qquad\qquad\qquad \sim Q\ V \end{array}\right]$$

Select R, and consider the following mating:

```
[[R [P8 M ]]    ∀S                        ]
[[ \       ]  v   [~ S [P8 N ] v S V]     ]
[[  ~ R U  ]        /               /     ]
[                  /              /       ]
[                Q U            /         ]
[                             /           ]
[               ~ Q V                     ]
```

From here on the mating dictates what is to be done. Substitute $\lambda H_{o\alpha} U_{\alpha}$ for P^8.

```
[[R [[λH U] M ]]    ∀S                            ]
[[             ]  v   [~ S [[λH U] N ] v S V]     ]
[[    ~ R U    ]                                  ]
[                                                 ]
[                       Q U                       ]
[                                                 ]
[                      ~ Q V                      ]
```

λ-normalize:

```
[[ R U ]    ∀S                ]
[[     ]  v   [~ S U v S V]   ]
[[~ R U]                      ]
[                             ]
[          Q U                ]
[                             ]
[         ~ Q V               ]
```

Instantiate S:

```
[[ R U ]                      ]
[[     ]  v   [~ Q U v Q V]   ]
[[~ R U]                      ]
[                             ]
[          Q U                ]
[                             ]
[         ~ Q V               ]
```

This wff is contradictory. Examination of the proof above shows that the expansion term finally substituted for $P_{o(o\alpha)}$ is $[\lambda S_{o\alpha}.S\ U_{\alpha}]$. Thus, the idea underlying the proof can be expressed as follows: suppose [UNITSET U] = [UNITSET V]; therefore [UNITSET U] U implies that [UNITSET V] U; the conclusion that U = V follows easily using the definition of UNITSET.

In Huet's paper [22] on constrained resolution, Huet introduces "splitting rules" which clearly introduce connectives and quantifiers, but one might wonder how constrained resolution handles THM104, where a projection must be introduced if one uses the method above. Actually, when using constrained resolution one generates a clause with the constraint that the literal $P_{o(o\alpha)}\ [\lambda Z_\alpha\ \forall S_{o\alpha}.S\ Z \supset S\ V_\alpha]$ must be unified with an expression of the form $\forall M_{o\alpha}[H_{o(o\alpha)}\ M]$, and the unifying substitution provides the projection for P.

Of course, generating expansion terms incrementally by using primitive substitutions as illustrated above is a rather primitive procedure, and additional methods are clearly needed. The most significant work on this problem which has been published so far is Bledsoe's work on set variables [16] [17]. His approach is based on the very sensible idea that it is often possible to construct expansion terms in a systematic way from expressions which are already present in the theorem to be proved. He gives specific basic rules and combining rules for doing this in certain contexts. With the aid of these rules his theorem prover is able to perform quite impressively on certain examples. However, much research remains to be done on this problem.

Let's think some more about how to use primitive substitutions.

Of course, one can choose the connectives and quantifiers to be introduced by primitive substitutions in a variety of ways. If one uses a minimal complete set such as $\{\sim, \wedge, \forall\}$, the process of generating expansion terms incrementally may be unduly arduous, so it may be better to use $\{\sim, \wedge, \vee, \forall, \exists\}$. As Sunil Issar has pointed out, in most contexts we may let the bodies of expansion terms be in nnf, so negations need be introduced only sparingly. Indeed, the matingsearch process can decide which literals require negations, so negations which will have only atomic scope in the wff need not be introduced by primitive substitutions. However, we cannot always assume that expansion terms can be in nnf; if a variable which will be replaced by an expansion term in the deep formula occurs as an argument of some constant symbol, then that expansion term may need to have whatever form will make the literal containing it complementary to some other literal. In such cases we would hope to generate the expansion term by direct unification of the atoms rather than by primitive substitutions.

In general, there will be infinitely many primitive substitutions for a variable, since some of the substitution terms will contain expressions of the form $\forall x_\alpha B_o$, where α is an arbitrary type symbol. The problem of how to choose the type symbol α is deep and significant. Of course, a natural thing to do is to use type variables for type symbols and determine what to substitute for these later. This is a problem which invites extensive research. At the very least, it suggests the need for a unification algorithm for expressions of typed λ-calculus containing type variables. For now we shall simply assume that the infinitely many primitive substitutions, and other expressions generated from them, will be represented in some finite way. In actual mathematical practice it is rather rare to use quantifiers on variables whose types are much more complicated than those in the theorem being proved, so until we are much more ambitious it should suffice to simply use a fairly small set of simple types in place of the infinite set of all types.

Of course, it is desirable to limit the use of primitive substitutions as much as possible. Actually, they may usefully be applied only to certain variables in a wff, which we shall call *eligible* (for primitive substitutions).

We shall call the first proper symbol of the λ-normal form of an atom the *head* of that atom, and call a variable which is the head of some atom of a wff a *head variable* of that wff. Only head variables need be eligible. Also, if a variable occurs only at the heads of atoms and these atoms are all positive or all negative, we can make all these atoms reduce to tautologies or contradictions by a simple substitution, so there is no need for such a variable to be eligible.

Usually an eligible variable will occur as head of both positively and negatively occurring atoms of the wff. In this case both conjunctive and disjunctive primitive substitutions for that variable will enlarge the number of paths through the wff, and an analysis of the costs and benefits to the matingsearch process of this enlargement may be useful.

The fact that it may be appropriate to apply several different primitive substitutions to the same variable, at least until the matingsearch process determines which substitutions are useful, makes it necessary to consider how this should be done in an expansion tree. We may assume that the processors which expand the formula work with a copy of the formula to which no substitutions are applied except those involved in the expansion process, and that the substitutions generated by the matingsearch process are not actually applied until all necessary expansions have been made. We may also assume that variables introduced as expansion terms, or as free variables in substitution terms, are all new variables which are distinct from one another. Therefore, it is easy to see that no variable will occur free in more than one expansion term. The result of applying the primitive substitutions $\theta^1, \ldots, \theta^n$ to the eligible variable v in an expansion tree may be described as follows. Let t be the expansion term in which v occurs free, and N be the expansion node from which the arc labeled with t descends. Create n variants $t^1, \ldots, t^n$ of t by substituting new and distinct variables for the free variables of t other than v, so that these variants share no free variables except for v. Now add n new arcs descending from node N labeled with expansion terms $(\theta^1 t^1)$, ..., and $(\theta^n t^n)$, respectively. Note that the original arc labeled with t remains in the tree; we have no way of knowing whether it is appropriate to apply primitive substitutions to v or not, so we keep one copy of v without change.

We visualize a search procedure in which variables are instantiated only by primitive substitutions for head variables and by the unifier associated with a mating. It's natural to ask whether such a procedure is complete in an appropriate sense.

In principle, we don't need to backtrack if we make inappropriate expansions of the wff, since this simply enlarges the search space. This is similar to deriving redundant clauses in resolution, though perhaps more costly. However, if the wff is repeatedly expanded in a blind and unmotivated way, one will quickly generate an unmanageably large search space for the matingsearch process. Of course, such situations are well known to researchers in this field. In general, we would like the matingsearch process to motivate the particular expansions of the wff which are made. The development of heuristics to guide the process of expanding the wff is a major area for research. It is important to understand how appropriate expansions can be used to create the literals needed for the connections of an acceptable mating.

One quite simple expansion procedure is based on the idea that one should search first for expansion proofs in which the formula has not been expanded very much. Thus, the matingsearch processors should work on copies of the expansion tree to which only one expansion option has been applied until all such options have been explored, then on copies to which only two expansion options have been applied, etc. (More generally, one could assign weights to combinations of expansion options, and minimize these weights.) The names of the nodes in these copies can always be taken from the master expansion tree maintained by the expansion processors, so consistent contributions to the failure record can be made by these processors working on different fragments of the master tree.

11. Conclusion

Many problems arise in connection with the automation of higher-order logic. By formulating these precisely and dealing with them systematically, we can hope to eventually make progress in this important realm. There's lots of room for good research here.

12. References

1. Peter B. Andrews, *Resolution in Type Theory,* Journal of Symbolic Logic 36 (1971), 414-432.

2. Peter B. Andrews, *Resolution and the Consistency of Analysis,* Notre Dame Journal of Formal Logic 15 (1974), 73-84.

3. Peter B. Andrews, *Refutations by Matings,* IEEE Transactions on Computers C-25 (1976), 801-807.

4. Peter B. Andrews, "Transforming Matings into Natural Deduction Proofs," in *5th Conference on Automated Deduction,* edited by W. Bibel and R. Kowalski, Les Arcs, France, Lecture Notes in Computer Science 87, Springer-Verlag, 1980, 281-292.

5. Peter B. Andrews, *Theorem Proving via General Matings,* Journal of the ACM 28 (1981), 193-214.

6. Peter B. Andrews, Dale A. Miller, Eve Longini Cohen, Frank Pfenning, "Automating Higher-Order Logic," in *Automated Theorem Proving: After 25 Years,* edited by W. W. Bledsoe and D. W. Loveland, Contemporary Mathematics series, vol. 29, American Mathematical Society, 1984, 169-192.

7. Peter B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof,* Academic Press, 1986.

8. Peter B. Andrews, "Connections and Higher-Order Logic," in *8th International Conference on Automated Deduction,* edited by Jorg H. Siekmann, Oxford, England, Lecture Notes in Computer Science 230, Springer-Verlag, 1986, 1-4.

9. Wolfgang Bibel and J. Schreiber, "Proof search in a Gentzen-like system of first-order logic," in *International Computing Symposium 1975,* edited by E. Gelenbe and D. Potier, North-Holland, Amsterdam, 1975, 205-212.

10. Wolfgang Bibel, *Tautology Testing with a Generalized Matrix Reduction Method,* Theoretical Computer Science 8 (1979), 31-44.

11. Wolfgang Bibel, *On Matrices with Connections,* Journal of the ACM 28 (1981), 633-645.

12. Wolfgang Bibel, *Automated Theorem Proving*, Vieweg, Braunschweig, 1982.

13. Wolfgang Bibel, *A Comparative Study of Several Proof Procedures*, Artificial Intelligence **18** (1982), 269-293.

14. Wolfgang Bibel, *Matings in Matrices*, Communications of the ACM **26** (1983), 844-852.

15. Wolfgang Bibel and Bruno Buchberger, *Towards a Connection Machine for Logical Inference*, Future Generations Computer Systems **1** (1984-1985), 177-185.

16. W. W. Bledsoe. A Maximal Method for Set Variables in Automatic Theorem Proving. In *Machine Intelligence 9*, Ellis Harwood Ltd., Chichester, and John Wiley & Sons, 1979, pp. 53-100.

17. W. W. Bledsoe, "Using Examples to Generate Instantiations of Set Variables," in *Proceedings of IJCAI-83, Karlsruhe, Germany*, Aug 8-12, 1983, 892-901.

18. Alonzo Church, *A Formulation of the Simple Theory of Types*, Journal of Symbolic Logic **5** (1940), 56-68.

19. Martin Davis, "Eliminating the Irrelevant from Mechanical Proofs," in *Experimental Arithmetic, High Speed Computing and Mathematics*, Proceedings of Symposia in Applied Mathematics XV, American Mathematical Society, 1963, 15-30.

20. Burton Dreben, Peter Andrews, and Stal Aanderaa, *False Lemmas in Herbrand*, Bulletin of the American Mathematical Society **69** (1963), 699-706.

21. Burton Dreben and John Denton, *A Supplement to Herbrand*, Journal of Symbolic Logic **31** (1966), 393-398.

22. Gêrard P. Huet, "A Mechanization of Type Theory," in *Proceedings of the Third International Joint Conference on Artificial Intelligence*, IJCAI, 1973, 139-146.

23. Gerard P. Huet, *A Unification Algorithm for Typed λ-Calculus*, Theoretical Computer Science **1** (1975), 27-57.

24. D. C. Jensen and T. Pietrzykowski, *Mechanizing ω-Order Type Theory Through Unification*, Theoretical Computer Science **3** (1976), 123-171.

25. Dale A. Miller. *Proofs in Higher-Order Logic*, Ph.D. Thesis, Carnegie Mellon University, 1983. 81 pp.

26. Dale A. Miller, "Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs," in *7th International Conference on Automated Deduction*, edited by R. E. Shostak, Napa, California, USA, Lecture Notes in Computer Science 170, Springer-Verlag, 1984, 375-393.

27. Dale A. Miller, *A Compact Representation of Proofs*, Studia Logica **46** (1987), 347-370.

28. Frank Pfenning, "Analytic and Non-analytic Proofs," in *7th International Conference on Automated Deduction*, edited by R. E. Shostak, Napa, California, USA, Lecture Notes in Computer Science 170, Springer-Verlag, 1984, 394-413.

29. Frank Pfenning. *Proof Transformations in Higher-Order Logic*, Ph.D. Thesis, Carnegie Mellon University, 1987. 156 pp.

30. Dag Prawitz. Advances and Problems in Mechanical Proof Procedures. In *Machine Intelligence 4*, Edinburgh University Press, 1969, pp. 59-71.

31. Dag Prawitz, "A proof procedure with matrix reduction," in *Symposium on Automatic Demonstration, Versailles, France*, edited by M. Laudet, D. Lacombe, L. Nolin, and M. Schutzenberger, Lecture Notes in Mathematics 125, Springer-Verlag, 1970, 207-214.

32. Robert E. Shostak, *Refutation Graphs,* Artificial Intelligence 7 (1976), 51-64.