HIGHER-ORDER HORN CLAUSES

GOPALAN NADATHUR
Duke University, Durham, North Carolina

DALE MILLER
University of Pennsylvania, Philadelphia, Pennsylvania

Abstract: A generalization of Horn clauses to a higher-order logic is described and examined as a basis for logic programming. In qualitative terms, these higher-order Horn clauses are obtained from the first-order ones by replacing first-order terms with simply typed λ-terms and by permitting quantification over all occurrences of function symbols and some occurrences of predicate symbols. Several proof-theoretic results concerning these extended clauses are presented. One result shows that although the substitutions for predicate variables can be quite complex in general, the substitutions necessary in the context of higher-order Horn clauses are tightly constrained. This observation is used to show that these higher-order formulas can specify computations in a fashion similar to first-order Horn clauses. A complete theorem proving procedure is also described for the extension. This procedure is obtained by interweaving higher-order unification with backchaining and goal reductions, and constitutes a higher-order generalization of SLD-resolution. These results have a practical realization in the higher-order logic programming language called λProlog.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory — syntax; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic — logic programming; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving — logic programming

General Terms: Languages, Theory

Additional Key Words and Phrases: Higher-order logic, higher-order unification, Horn clauses, Prolog, SLD-resolution

The work of G. Nadathur was funded in part by NSF grant MCS-82-19196 and a Burroughs contract. The work of D. Miller was funded by NSF grant CCR-87-05596 and DARPA grant N000-14-85-K-0018.

Authors' current addresses: G. Nadathur, Computer Science Department, Duke University, Durham, NC 27706; D. Miller, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-6389.

1. Introduction

A principled analysis of the nature and role of higher-order notions within logic programming appears to be absent from the literature on this programming paradigm. Some attempts, such as those in [34], have been made to realize higher-order features akin to those in functional programming languages, and have even been incorporated into most existing versions of the language Prolog. These attempts are, however, unsatisfactory from two perspectives. First, they have relied on the use of *ad hoc* mechanisms and hence are at variance with one of the strengths of logic programming, namely its basis in logic. Second, they have not taken full cognizance of the difference between the functional and logic programming paradigms and, consequently, of potential differences between higher-order notions in these paradigms.

Towards filling this lacuna, this paper initiates investigations into a logical basis for higher-order features in logic programming. The principal concern here is that of describing an extension to Horn clauses [33], the basis of languages such as Prolog [32], by using a higher-order logic. The use of the term "extension" clearly signifies that there is some character of Horn clauses that is to be retained. This character may most conveniently be enunciated in the context of a generalized version of Horn clauses that is, in some senses, closer to actual realizations of logic programming. Letting A represent an atomic formula, we identify *goal* formulas as those given by the rule

$$G ::= A \mid G_1 \vee G_2 \mid G_1 \wedge G_2 \mid \exists x G,$$

and *definite* sentences as the universal closure of atomic formulas and of formulas of the form $G \supset A$; the symbol G is used in each case as a syntactic variable for a goal formula. These formulas are related to Horn clauses in the following sense: Within the framework of classical first-order logic, the negation of a goal formula is equivalent to a set of negative Horn clauses and, similarly, a definite sentence is equivalent to a set of positive Horn clauses. Now, if $\mathcal{P}$ is a set of definite sentences and $\vdash$ denotes provability in (classical) first-order logic, then the following properties may be noted:

(i) $\mathcal{P} \vdash \exists x G$ only if there is a term t such that $\mathcal{P} \vdash G[t/x]$, where $G[t/x]$ represents the result of substituting t for all the free occurrences of x in G.

(ii) $\mathcal{P} \vdash G_1 \vee G_2$ only if $\mathcal{P} \vdash G_1$ or $\mathcal{P} \vdash G_2$.

(iii) $\mathcal{P} \vdash G_1 \wedge G_2$ only if $\mathcal{P} \vdash G_1$ and $\mathcal{P} \vdash G_2$.

(iv) If A is an atomic formula, then $\mathcal{P} \vdash A$ only if either (a) A is a substitution instance of a formula in $\mathcal{P}$, or (b) there is a substitution instance of the form $G \supset A$ of a definite sentence in $\mathcal{P}$ such that $\mathcal{P} \vdash G$.

While the converse of each property above follows from the meanings of the logical connectives and quantifiers, these properties themselves are a consequence of the special structure of goal formulas and definite sentences. The importance of these properties is in the role they play in the computational interpretation accorded to these formulas. Logic programming is based on construing a collection, $\mathcal{P}$, of definite sentences as a program and a goal formula, G, as a query. The idea of a computation in this context is that of constructing a proof of the existential closure of G from $\mathcal{P}$ and, if this process is successful, of extracting from this proof a substitution instance of G that is provable from $\mathcal{P}$. The consistency of this view is apparently dependent on (i). In a more fine-grained analysis, (i)–(iv) collectively support a feature of importance to logic programming, namely the ability to construe each formula as contributing to the specification of a *search*, the nature of the search being described through the use of logical connectives and quantifiers. Thus, (ii) and (iii) permit the interpretation of the propositional connectives $\vee$ and $\wedge$ as primitives for specifying non-deterministic *or* and *and* branches in a search, and (i) warrants the conception of the existential quantifier as the means for describing an infinite (non-deterministic) *or* branch where the branches are parameterized by the set of terms. Similarly, (iv) permits us to interpret a definite sentence as partially defining a procedure: For instance, the formula $G \supset A$ corresponds to the description that an attempt to solve a procedure whose name is the predicate head of A may be made by trying to solve the goal G. As a final observation, we see that (i)–(iv) readily yield a proof procedure, closely related to *SLD-resolution* [2], that, in fact, forms the computational machinery for realizing this programming paradigm.

The properties discussed above thus appear to play a central role in the context of logic programming, and it is desirable to retain these while extending the formulas underlying this programming paradigm. This paper provides one such extension. The formulas described here may, in an informal sense, be characterized as those obtained from first-order goal formulas and definite sentences by supplanting first-order terms with the terms of a typed λ-calculus and by permitting quantification over function and predicate symbols. These formulas provide for higher-order features of two distinct kinds within logic programming. The first arises out of the presence of predicate variables. Given the correspondence between predicates and procedures in logic programming, this facilitates the writing of procedures that take other procedures as arguments, a style of programming often referred to as *higher-order programming*. Occurrences of predicate variables are restricted and so also are the appearances of logical connectives in terms, but the restrictions are well motivated from a programming perspective. They may, in fact, be informally understood as follows. First, the name of a procedure defined by a definite sentence, *i.e.* the head of A in a formula of the form $G \supset A$, must not be a variable. Second, only those logical connectives that may appear in the top-level logical structure of a goal formula are permitted in terms; the picture here is that when a predicate variable in the body of a procedure declaration is instantiated, the result is expected to be a legitimate goal formula or query. The quantification over predicate variables that is permitted and the corresponding enrichment to the term structure are however sufficient to allow a direct emulation within logic programming of various higher-order functions (such as the *map* and *reduce* functions of Lisp) that have been found to be useful in the functional programming context.

The second, and truly novel, feature of our extension is the provision of λ-terms as data structures. There has been a growing interest in recent years in programming environments in which complex syntactic objects such as formulas, programs and proofs can be represented and manipulated easily [6, 11, 31]. In developing environments of this kind, programming languages that facilitate the representation and manipulation of these kinds of objects play a fundamental role. As is evident from the arguments provided elsewhere [16, 22, 30, 28], the representation of objects involving the notion of binding, *i.e.* objects such as formulas, programs and proofs, is best achieved through the use of a term language based on the λ-calculus. The task of reasoning about such objects in turn places an emphasis on a programming paradigm that provides primitives for examining the structure of λ-terms and that also supports the notion of search in an intrinsic way. Although the logic programming paradigm is a natural choice from the latter perspective, there is a need to enrich the data structures of a language such as Prolog before it is genuinely useful as a "metalanguage". The analysis in this paper provides for a language with such an enrichment and consequently leads to a language that potentially has several novel applications. Detailed experiments in some of the application realms show that this potential is in fact borne out. The interested reader may refer, for instance, to [7, 13, 20, 22, 25, 29] for the results of these experiments.

It should be mentioned that the extension to first-order Horn clauses described in this paper is not the only one possible that preserves the spirit of properties (i)–(iv). The primary aim here is that of examining the nature and role of higher-order notions within logic programming and this underlies the focus on enriching only the nature of quantification within (first-order) Horn clauses. It is possible, however, to consider other enrichments to the logical structure of these formulas, perhaps after explaining what it means to preserve the spirit of properties (i)–(iv) if a richer set of connectives is involved. Such a task is undertaken in [23]. Briefly, a proof-theoretic criterion is presented therein for determining when a logical language provides the basis for logic programming and this is used to describe a family of extensions to first-order Horn clauses. The "richest" extension described in [23] replaces definite sentences by a class of formulas called *higher-order hereditary Harrop* or *hohh* formulas. The higher-order definite sentences of this paper are a subclass of the latter class of formulas. Further, the use of *hohh* formulas provides for notions such as modules, abstract data types and lexical scoping in addition to higher-order features within logic programming. There is, however, a significant distinction to be made between the *theory* of the higher-order Horn clauses presented in this paper and of the *hohh* formulas presented in [23]. The distinction, expressed informally, is that the programming interpretation of higher-order definite sentences accords well with classical logic, whereas a shift to intuitionistic logic is required in the context of *hohh* formulas. This shift in semantic commitment may not be acceptable in some applications of logic programming. Despite this difference, it is to be noted that several of the proof-theoretic techniques presented in this paper have been generalized and utilized in [23]. It is in fact a perceived generality of these techniques that justifies their detailed presentation here.

This rest of this paper is organized as follows. In the next section we describe the higher-order logic used in this paper, summarizing several necessary logical notions in the process. The axiomatization presented here for the logic is in the style of Gentzen [9]. The use of a sequent calculus, although unusual in the literature on logic programming (see [4] for an exception), has several advantages. One advantage is the simplification of proof-theoretic discussions that we hope this paper demonstrates. In another direction, it is the casting of our arguments within such a calculus that has been instrumental in the discovery of the "essential" character of logic programming and thereby in the description of further logical extensions to it [18, 23]. In Section 3, we outline the classes of formulas within this logic that are our generalizations to first-order goal formulas and definite sentences. Section 4 is devoted to showing that these formulas satisfy properties (i)–(iv) above when $\vdash$ is interpreted as provability in the higher-order logic. The main problem here is the presence of predicate variables; substitutions for these kinds of variables may, in general, alter the logical structure of formulas in which they appear and thus complicate the nature of proofs. Fortunately, as we show in the first part of Section 4, predicate variable substitutions can be tightly constrained in the context of our higher-order formulas. This result is then used to show that these formulas provide a satisfactory basis for a logic programming language. In Section 5 we describe a theorem-proving procedure that provides the basis for an interpreter for such a language. This procedure interweaves higher-order unification [15] with backchaining and goal reductions and constitutes a higher-order generalization to SLD-resolution. These results have been used to describe a logic programming language called λProlog. A presentation of this language is beyond the scope of this paper but may be found in [21, 25, 27].

2. A Higher-Order Logic

The higher-order logic used in this paper is derived from Church's formulation of the simple theory of types [5] principally by the exclusion of the axioms concerning infinity, choice, extensionality and description. Church's logic is particularly suited to our purposes since it is obtained by superimposing logical notions over the calculus of λ-conversion. Our omission of certain axioms is based on a desire for a logic that generalizes first-order logic by providing a stronger notion of a variable and a term, but at the same time encompasses only the most primitive logical notions that are relevant in this context; only these notions appear to be of consequence from the perspective of computational applications. Our logic is closely related to that of [1], the only real differences being the inclusion of η-conversion as a rule of inference and the incorporation of a larger number of propositional connectives and quantifiers as primitives. We describe this logic below, simultaneously introducing the logical vocabulary used in the rest of the paper.

The Language. The language used is based on a typed λ-calculus. The types in this context are determined by a set $\mathcal{S}$ of *sorts*, that contains at least the sort o and one other sort, and by a set $\mathcal{C}$ of *type constructors* each member of which has a unique positive arity: The class of types is then the smallest collection that includes (i) every sort, (ii) $(c\,\alpha_1\,\ldots\,\alpha_n)$, for every $c \in \mathcal{C}$ of arity n and every $\alpha_1, \ldots, \alpha_n$ that are types, and (iii) $(\alpha \to \beta)$ for every α and β that are types. We refer to the types obtained by virtue of (i) and (ii) as *atomic types* and to those obtained by virtue of (iii) as *function* types. In an informal sense, each type may be construed as corresponding to a set of objects. Understood in this manner, $(\alpha_1 \to \alpha_2)$ corresponds to the collection of functions each of whose domain and range is determined by α_1 and α_2, respectively. In writing types, the convention that $\to$ associates to the right is often used to omit parentheses. In this paper, the letters α and β are used, perhaps with subscripts, as syntactic variables for types. An arbitrary type is occasionally depicted by an expression of the form $(\alpha_1 \to \cdots \to \alpha_n \to \beta)$ where β is assumed to be an atomic type. When a type is displayed in this form, we refer to $\alpha_1, \ldots, \alpha_n$ as its *argument* types and to β as its *target* type. The use of such a notation for atomic types is justified by the convention that the argument types may be an empty sequence.

We now assume that we are provided with a denumerable set, $\mathcal{V}ar_\alpha$, of *variables* for each type α, and with a collection of constants of arbitrary given types, such that the subcollection at each type α is denumerable and disjoint from $\mathcal{V}ar_\alpha$. The latter collection is assumed to contain at least one member of each type, and to include the following infinite list of symbols called the *logical* constants: $\top$ of type o, $\sim$ of type $o \to o$, $\wedge$, $\vee$ and $\supset$ of type $o \to o \to o$, and, for each α, Σ and Π of type $(\alpha \to o) \to o$. The remaining constants are referred to as *parameters*. The class of *formulas* or *terms* is then defined inductively as follows:

(i) A variable or a constant of type α is a formula of type α.

(ii) If x is a variable of type α_1 and F is a formula of type α_2 then $[\lambda x.F]$ is a formula of type $\alpha_1 \to \alpha_2$, and is referred to as an *abstraction* that *binds* x and whose *scope* is F.

(iii) If F_1 is a formula of type $\alpha_1 \to \alpha_2$ and F_2 is a formula of type α_1 then $[F_1\,F_2]$, referred to as the *application* of F_1 to F_2, is a formula of type α_2.

Certain conventions concerning formulas are employed in this paper. First, lower-case letters are used, perhaps with subscripts, to denote formulas that are variables; such a usage may occur at the level of either the object or the meta language. Similarly, upper-case letters are used to denote parameters at the object level and arbitrary formulas in the meta language. Second, in the interest of readability, the brackets that surround expressions formed by virtue of (ii) and (iii) above are often omitted. These may be restored by using the conventions that abstraction is right associative, application is left associative and application has smaller scope than abstraction. Finally, although each formula is specified only in conjunction with a type, the types of formulas are seldom explicitly mentioned. Such omissions are justified on the basis that the types are either inferable from the context or inessential to the discussion at hand.

The rules of formation serve to identify the well-formed subparts of each formula. Specifically, G is said to *occur* in, or to be a *subformula* of, F if (a) G is F, or (b) F is $\lambda x.F_1$ and G occurs in F_1, or (c) F is $[F_1\,F_2]$ and G occurs in either F_1 or F_2. An occurrence of a variable x in F is either *bound* or *free* depending on whether it is or is not an occurrence in the scope of an abstraction that binds x. x is a bound (free) variable of F if it has at least one bound (free) occurrence in F. F is a *closed* formula just in case it has no free variables. We write $\mathcal{F}(F)$ to denote the set of free variables of F. This notation is generalized to sets of formulas and sets of pairs of formulas in the following way: $\mathcal{F}(\mathcal{D})$ is $\bigcup\{\mathcal{F}(F) \mid F \in \mathcal{D}\}$ if $\mathcal{D}$ is a set of formulas and $\bigcup\{\mathcal{F}(F_1) \cup \mathcal{F}(F_2) \mid \langle F_1, F_2\rangle \in \mathcal{D}\}$ if $\mathcal{D}$ is a set of pairs of formulas.

The type o has a special significance. Formulas of this type correspond to propositions, and a formula of type $\alpha_1 \to \cdots \to \alpha_n \to o$ is a predicate of n arguments whose i^{th} argument is of type α_i. In accordance with the informal interpretation of types, predicates may be thought of as representing sets of n-tuples or relations. The logical constants are intended to be interpreted in the following manner: $\top$ corresponds to the tautologous proposition, the *(propositional) connectives* $\sim$, $\vee$, $\wedge$, and $\supset$ correspond, respectively, to negation, disjunction, conjunction, and implication, and the family of constants Σ and Π are, respectively, existential and universal quantifiers, viewed as propositional functions of propositional functions. There are certain notational conventions pertaining to the logical constants that find use below. First, disjunction, conjunction and implication are written as infix operations; *e.g.* $[\vee\, F\, G]$ is usually written as $[F \vee G]$. Second, the expressions $[\exists x.F]$ and $[\forall x.F]$ serve as abbreviations for $[\Sigma\, \lambda x.F]$ and $[\Pi\, \lambda x.F]$; these abbreviations illustrate the use of Σ and Π along with abstractions to create the operations of existential and universal quantification familiar in the context of first-order logic. Finally $\bar{x}$ is sometimes used as an abbreviation for a sequence of variables $x_1, \ldots, x_n$. In such cases, the expression $\exists \bar{x}.F$ serves as a shorthand for $\exists x_1. \;\ldots\; \exists x_n.F$. Similar interpretations are to be bestowed upon $\forall \bar{x}.F$ and $\lambda \bar{x}.F$.

The Calculus of λ-Conversion. In the interpretation intended for the language, λ is to correspond to the abstraction operation and juxtaposition to the operation of function application. These intentions are formalized by the rules of λ-conversion. To define these rules, we need the operation of replacing all free occurrences of a variable x in the formula F by a formula G of the same type as x. This operation is denoted by $S^x_G\, F$ and may be made explicit as follows:

(i) If F is a variable or a constant, then $S^x_G\, F$ is G if F is x and F otherwise.

(ii) If F is of the form $\lambda y.C$, then $S^x_G\, F$ is F if y is x and $\lambda y.S^x_G\, F$ otherwise.

(iii) If F is of the form $[C\, D]$, then $S^x_G\, F = [(S^x_G\, C)\,(S^x_G\, D)]$.

In performing this operation of replacement, there is the danger that the free variables of G become bound inadvertently. The term "G is *free for x in F*" describes the situations in which the operation is logically correct, *i.e.* those situations where x does not occur free in the scope of an abstraction in F that binds a free variable of G. The rules of *α-conversion*, *β-conversion* and *η-conversion* are then, respectively, the following operations on formulas:

(1) Replacing a subformula $\lambda x.F$ by $\lambda y.S^x_y\, F$ provided y is free for x in F and y is not free in F.

(2) Replacing a subformula $[\lambda x.F]\, G$ by $S^x_G\, F$ provided G is free for x in F and vice versa.

(3) Replacing a subformula $\lambda x.[F\, x]$ by F provided x is not free in F and vice versa.

The rules above, collectively referred to as the λ-conversion rules, are used to define the following relations on formulas.

2.1. Definition. F *λ-conv* (*β-conv*, $\equiv$) G just in case there is a sequence of applications of the λ-conversion (respectively α- and β-conversion, α-conversion) rules that transforms F into G.

The three relations thus defined are evidently equivalence relations. They correspond, in fact, to notions of equality between formulas based on the following informal interpretation of the λ-conversion rules: α-conversion asserts that the choice of name for the variable bound by an abstraction is unimportant, β-conversion relates an application to the result of evaluating the application, and η-conversion describes a weak notion of extensionality for formulas. In this paper we use the strongest of these notions, *i.e.* we consider F and G equal just in case F λ-conv G. There are certain distinctions to be made between formulas by omitting the rules of η-conversion, but we feel that these are not important in our context.

A formula F is said to be a *β-normal form* if it does not have a subformula of the form $[\lambda x.A]\, B$, and a *λ-normal form* if, in addition, it does not have a subformula of the form $\lambda x.[A\, x]$ with x not occurring free in A. If F is a β-normal form (λ-normal form) and G β-conv (λ-conv) F, then F is said to be a β-normal (λ-normal) form of G. From the Church-Rosser Theorem, described in, *e.g.*, [3] for a λ-calculus without type symbols but applicable to the language under consideration as well, it follows that a β-normal (λ-normal) form of a formula is unique up to a renaming of bound variables. Further, it is known [1, 8] that a β-normal form exists for every formula in the typed λ-calculus; this may be obtained by repeatedly replacing subformulas of the form $[\lambda x.A]\,B$ by $S^x_B\,A$, preceded, perhaps, by some α-conversion steps. Such a formula may be converted into a λ-normal form by replacing each subformula of the form $\lambda x.[A\,x]$ where x does not occur free in A by A. In summary, any formula F may be converted into a λ-normal form that is unique up to α-conversions. We denote such a form by $\lambda norm(F)$. Occasionally, we need to talk of a unique normal form and, in such cases, we use $\rho(F)$ to designate what we call the *principal normal form* of F. Determining this form essentially requires a naming convention for bound variables and a convention such as that in [1] will suffice for our purposes.

The existence of a λ-normal form for each formula provides a mechanism for determining whether two formulas are equal by virtue of the λ-conversion rules. These normal forms also facilitate the discussion of properties of formulas in terms of a representative for each of the equivalence classes that has a convenient structure. In this context, we note that a β-normal form is a formula that has the structure

$$\lambda x_1.\ \ldots\ \lambda x_n.[A\,F_1\ \ldots\ F_m]$$

where A is a constant or variable, and, for $1 \leq i \leq m$, F_i also has the same form. We refer to the sequence $x_1, \ldots, x_n$ as the *binder*, to A as the *head* and to $F_1, \ldots, F_m$ as the *arguments* of such a formula; in particular instances, the binder may be empty, and the formula may also have no arguments. Such a formula is said to be *rigid* if its head, *i.e.* A, is either a constant or a variable that appears in the binder, and *flexible* otherwise. A formula having the above structure is also a λ-normal form if F_m is not identical to x_n and, further, each of the F_is also satisfy this constraint. In subsequent sections, we shall have use for the structure of λ-normal forms of type o. To describe this, we first identify an *atom* as a λ-normal form whose leftmost symbol that is not a bracket is either a variable or a parameter. Then, a λ-normal form of type o is one of the following: (i) $\top$, (ii) an atom, (iii) $\sim F$, where F is a λ-normal form of type o, (iv) $[F \vee G]$, $[F \wedge G]$, or $[F \supset G]$ where F and G are λ-normal forms of type o, or (v) $\Sigma\, P$ or $\Pi\, P$, where P is a λ-normal form.

Substitutions. A *substitution* is a set of the form $\{\langle x_i, F_i\rangle \mid 1 \leq i \leq n\}$, where, for $1 \leq i \leq n$, each x_i is a distinct variable and F_i is a formula in principal normal form of the same type as, but distinct from, x_i; this substitution is a substitution *for* $\{x_1, \ldots, x_n\}$, and its *range* is $\{F_1, \ldots, F_n\}$. A substitution may be viewed as a type preserving mapping on variables that is the identity everywhere except the points explicitly specified. This mapping may be extended to the class of all formulas in a manner consistent with this view: If $\theta = \{\langle x_i, F_i\rangle \mid 1 \leq i \leq n\}$ and G is any formula, then

$$\theta(G) = \rho([\lambda x_1. \ldots \lambda x_n.G]\, F_1 \ldots F_n)$$

This definition is independent of the order in which we take the pairs from θ. Further, given our notion of equality between formulas, the application of a substitution to a formula G is evidently a formalization of the idea of replacing the free occurrences of $x_1, \ldots, x_n$ in G simultaneously by the formulas $F_1, \ldots, F_n$.

We need certain terminology pertaining to substitutions, and we summarize these here. A formula F is said to be an *instance* of another formula G if it results from applying a substitution to G. The *restriction* of a substitution θ to a set of variables $\mathcal{V}$, denoted by $\theta \uparrow \mathcal{V}$, is given as follows

$$\theta \uparrow \mathcal{V} = \{\langle x, F\rangle \mid \langle x, F\rangle \in \theta \text{ and } x \in \mathcal{V}\}.$$

It is evident that $\theta(G) = (\theta \uparrow \mathcal{F}(G))(G)$. The *composition* of two substitutions θ_1 and θ_2, written as $\theta_1 \circ \theta_2$, is precisely the composition of θ_1 and θ_2 when these are viewed as mappings: $\theta_1 \circ \theta_2(G) = \theta_1(\theta_2(G))$. Two substitutions, θ_1 and θ_2 are said to be *equal* relative to a set of variables $\mathcal{V}$ if it is the case that $\theta_1 \uparrow \mathcal{V} = \theta_2 \uparrow \mathcal{V}$; this relationship is denoted by $\theta_1 =_{\mathcal{V}} \theta_2$. θ_1 is said to be *less general than* θ_2 relative to $\mathcal{V}$, a relationship denoted by $\theta_1 \preceq_{\mathcal{V}} \theta_2$, if there is a substitution σ such that $\theta_1 =_{\mathcal{V}} \sigma \circ \theta_2$. Finally, we shall sometimes talk of the result of applying a substitution to sets of formulas and to sets of pairs of formulas. In the first case, we mean the set that results from applying the substitution to each formula in the set, and, in the latter case, we mean the set of pairs that results from the application of the substitution to each element in each pair.

The Formal System. The notion of derivation used in this paper is formalized by means of the sequent calculus *LKH* that is a higher-order extension to the logistic classical calculus *LK* of [9]. A *sequent* within the calculus *LKH* is an expression of the form

$$F_1, \ldots, F_n \longrightarrow G_1, \ldots, G_m$$

where $n \geq 0$, $m \geq 0$, and, for $1 \leq i \leq n$ and $1 \leq j \leq m$, F_i and G_j are formulas; the listing $F_1, \ldots, F_n$ is the *antecedent* of the sequent, and $G_1, \ldots, G_m$ forms its *succedent*. The *initial sequents* or *axioms* of the calculus are $\longrightarrow \top$ and the sequents of the form $A \longrightarrow A'$ where A and A' are atomic formulas such that $A \equiv A'$. The *inference figures* are the arrangements of sequents that result from the schemata in Figures 2.1 and 2.2

by replacing (i) the Γs and Δs by finite sequences of formulas of type o, (ii) A and B by formulas of type o, (iii) A', in the schemata designated by λ, by a formula resulting by a sequence of λ-conversions from the formula that replaces A in the lower sequent of the schema, and, finally, (iv) P, C, and y in the schemata designated by Σ−IS, Π−IS, Σ−IA, and Π−IA by, respectively, a formula of type $\alpha \to o$, a formula of type α, and a parameter or variable of type α that does not occur free in any formulas substituted into the lower sequent, for some choice of α. An inference figure is classified as a *structural* or an *operational* one, depending on whether it results from a schema in Figure 2.1 or 2.2. In the operational inference figures, we designate the formula substituted for the expression containing the logical constant as the *principal* formula of the figure. Some operational inference figures contain two upper sequents, and these are referred to, respectively, as the *left* and *right* upper sequents of the figure.

Thinning: *in the antecedent* *in the succedent*

$$\frac{\Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, A}$$

Contraction: *in the antecedent* *in the succedent*

$$\frac{A, A, \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta_1, F, \Delta_2}$$

Interchange: *in the antecedent* *in the succedent*

$$\frac{\Gamma_1, B, A, \Gamma_2 \longrightarrow \Delta}{\Gamma_1, A, B, \Gamma_2 \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta_1, A, B, \Delta_2}{\Gamma \longrightarrow \Delta_1, B, A, \Delta_2}$$

λ: *in the antecedent* *in the succedent*

$$\frac{A', \Gamma \longrightarrow \Delta}{A, \Gamma \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Delta, A'}{\Gamma \longrightarrow \Delta, A}$$

Figure 2.1: The *LKH* Structural Inference Figure Schemata

Intrinsic to a sequent calculus is the notion of a *derivation* or *proof figure.* These are tree-like arrangements of sequents that combine to form inference figures such that (i) each sequent, with the exception of one called the *endsequent*, is the upper sequent of exactly one inference figure, (ii) each sequent is the lower sequent of at most one inference figure and those sequents that are not the lower sequents of any inference figure are initial sequents. Such an arrangement constitutes a derivation *for* its endsequent. If there is a

$$\frac{\Gamma \longrightarrow \Delta, A}{\sim A, \Gamma \longrightarrow \Delta}\ \sim\!-\mathrm{IA} \qquad \frac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \sim A}\ \sim\!-\mathrm{IS}$$

$$\frac{A, \Gamma \longrightarrow \Delta \qquad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta}\ \vee-\mathrm{IA}$$

$$\frac{\Gamma \longrightarrow \Delta, A}{\Gamma \longrightarrow \Delta, A \vee B}\ \vee-\mathrm{IS} \qquad \frac{\Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \vee B}\ \vee-\mathrm{IS}$$

$$\frac{A, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta}\ \wedge-\mathrm{IA} \qquad \frac{B, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta}\ \wedge-\mathrm{IA}$$

$$\frac{\Gamma \longrightarrow \Delta, A \qquad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}\ \wedge-\mathrm{IS}$$

$$\frac{\Gamma_1 \longrightarrow \Delta_1, A \qquad B, \Gamma_2 \longrightarrow \Delta_2}{A \supset B, \Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2}\ \supset-\mathrm{IA}$$

$$\frac{A, \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \supset B}\ \supset-\mathrm{IS}$$

$$\frac{\rho([P\, y]), \Gamma \longrightarrow \Delta}{\Sigma\, P, \Gamma \longrightarrow \Delta}\ \Sigma-\mathrm{IA} \qquad \frac{\Gamma \longrightarrow \Delta, \rho([P\, C])}{\Gamma \longrightarrow \Delta, \Sigma\, P}\ \Sigma-\mathrm{IS}$$

$$\frac{\rho([P\, C]), \Gamma \longrightarrow \Delta}{\Pi\, P, \Gamma \longrightarrow \Delta}\ \Pi-\mathrm{IA} \qquad \frac{\Gamma \longrightarrow \Delta, \rho([P\, y])}{\Gamma \longrightarrow \Delta, \Pi\, P}\ \Pi-\mathrm{IS}$$

Figure 2.2: The *LKH* Operational Inference Figure Schemata

derivation for $\Gamma \longrightarrow A$, where A is a formula, then A is said to be *provable* from Γ. A *path* in a derivation is a sequence of sequents whose first member is the endsequent and whose last member is an initial sequent, and of which every member except the first is the upper sequent of an inference figure whose lower sequent is the preceding member. The *height* of the proof figure is the length of the longest path in the figure. Each sequent occurrence† in a derivation is on a unique path, and we refer to the number of sequents that precede it on this path as its *distance* from the endsequent. The notion of a derivation is relativized to particular sequent calculi by the choice of initial sequents and inference figures, and we use this observation in Section 4. Our current focus is on the calculus LKH, and we intend unqualified uses of the term "derivation" below to be read as LKH-derivation.

It is of interest to note that if η-conversion is added as a rule of inference to the system $\mathcal{T}$ of [1], then the resulting system is equivalent to the calculus LKH in the sense described in [9]. Specifically, let us say that the *associated formula* of the sequent $\Gamma \longrightarrow \Delta$ is $\wedge\Gamma \supset \vee\Delta$ if neither the antecedent nor the succedent is empty, $\vee\Delta$ if only the antecedent is empty, $\wedge\Gamma \supset p \wedge {\sim}p$ if only the succedent is empty, and $p \wedge {\sim}p$ if the antecedent and the succedent are both empty; p is a designated propositional variable here, and $[\wedge\Gamma]$ and $[\vee\Delta]$ are to be read as conjunctions and disjunctions of the formulas in Γ and Δ respectively. It is then the case that a derivation exists for $\Gamma \longrightarrow \Delta$ if and only if its associated formula is a theorem of $\mathcal{T}$ with the rule of η-conversion; we assume here that the symbols $\wedge$, $\supset$ and Σ are introduced via abbreviations in $\mathcal{T}$.

The reader familiar with [9] may notice several similarities between the calculi LK and LKH. One difference between these is the absence of the *Cut* inference figure in LKH. This omission is justified by the cut-elimination result for the version of higher-order logic under consideration [1]. Another, apparently superficial, difference is in the use in LKH of λ-conversion to capture the notion of substitution in inference figures pertaining to quantifiers. The only significant difference, then is in the richer syntax of formulas and the presence of the λ inference figures in LKH. We note in particular that the presence of predicate variables in formulas of LKH enables substitutions to change their logical structure. As a result, it is possible to describe several complex derivations in a concise form in this higher-order logic. However, this facet makes the task of constructing satisfactory proof procedures for this logic a difficult one. In fact, as we shall see shortly, considerable care must be taken even in enunciating and verifying the proof-theoretic properties of our higher-order formulas.

3. Higher-Order Definite Sentences and Goal Formulas

Using the higher-order logic of the previous section, the desired generalizations to first-order definite sentences and goal formulas may be identified. Intrinsic to this identification is the notion of a positive formula. As made precise by the following definition, these are the formulas in which the symbols $\sim$, $\supset$ and Π do not appear.

† The qualification "occurrence" will henceforth be assumed implicitly where necessary.

3.1. Definition. The class of positive formulas, $\mathcal{PF}$, is the smallest collection of formulas such that (i) each variable and each constant other than $\sim$, $\supset$ and Π is in $\mathcal{PF}$, and (ii) the formulas $\lambda x.A$ and $[A\,B]$ are in $\mathcal{PF}$ if A and B are in $\mathcal{PF}$. The *Positive Herbrand Universe*, $\mathcal{H}^+$, is the collection of all λ-normal formulas in $\mathcal{PF}$, and the *Herbrand Base*, $\mathcal{HB}$, is the collection of all closed formulas in $\mathcal{H}^+$.

As will become apparent shortly, $\mathcal{HB}$ in our context plays the same role as the *Herbrand Universe* does in the context of other discussions of logic programming: it is the domain of terms that is used in describing the results of computations.

3.2. Definition. A *higher-order goal formula* is a formula of type o in $\mathcal{H}^+$. A *positive atom* is an atomic goal formula, *i.e.* an atom in $\mathcal{H}^+$. A *higher-order definite formula* is any formula of the form $\forall \bar{x}.G \supset A$ where G is a goal formula, A is a rigid positive atom; in particular, $\bar{x}$ may be an empty sequence, *i.e.* the quantification may be absent. Such a formula is a higher-order definite *sentence* if it is closed, *i.e.* if $\bar{x}$ contains all the variables free in G and A. The qualification "higher-order" used in this definition is intended to distinguish the formulas defined from the first-order formulas of the same name and may be omitted if, by so doing, no confusion should arise.

A formula is a positive atom if it is either $\top$ or of the form $[A\,F_1\,\ldots\,F_n]$ where A is a parameter or a variable and, for $1 \leq i \leq n$, F_i is a positive formula, and is a rigid positive atom if, in the latter case, A is also a parameter. It is easily verified that a goal formula must be one of the following: (i) a positive atom, (ii) $A \vee B$ or $A \wedge B$, where A and B are goal formulas, or (iii) $\Sigma\,P$ where P is a predicate in $\mathcal{H}^+$; in the last case, we observe that it is equivalent to a formula of the form $\exists x.G$ where G is a goal formula. Thus, we see that the top-level logical structure of definite sentences and goal formulas in the higher-order setting is quite similar to those in the first-order context. The first-order formulas are, in fact, contained in the corresponding higher-order formulas under an implicit encoding that essentially assigns types to the first-order terms and predicates. To be precise, if i is a sort other that o, the encoding assigns the type i to variables and constants, the type $i \rightarrow \cdots \rightarrow i \rightarrow i$, with $n+1$ occurrences of i, to each n-ary function symbol, and the type $i \rightarrow \cdots \rightarrow i \rightarrow o$, with n occurrences of i, to each n-ary predicate symbol. Looked at differently, our formulas contain within them a many-sorted version of first-order definite sentences and goal formulas. However, they do embody a genuine generalization to the first-order formulas in that they may contain complex terms that are constructed by the use of abstractions and applications and, further, may also include quantifications over variables that correspond to functions and predicates. The following examples serve to illustrate these additional facets.

3.3. Example. Let *list* be a 1-ary type constructor and let *int* be a sort. Further, let *nil* and *cons* be parameters of type $(list\,int)$ and $int \rightarrow (list\,int) \rightarrow (list\,int)$ respectively, and let *mapfun* be a parameter of type $(int \rightarrow int) \rightarrow (list\,int) \rightarrow (list\,int) \rightarrow o$. Then the two formulas below are definite sentences:

$$\forall f.[\top \supset [mapfun\; f\; nil\; nil]],$$
$$\forall f.\forall x.\forall l_1.\forall l_2.[[mapfun\; f\; l_1\; l_2] \supset [mapfun\; f\; [cons\; x\; l_1]\; [cons\; [f\; x]\; l_2]]];$$

f is evidently a function variable in these formulas. If 1, 2, and and g are parameters of type *int*, *int*, and $int \rightarrow int \rightarrow int$ respectively, the following is a goal formula:

$$\exists l.[mapfun\; [\lambda x.[g\; x\; 1]]\; [cons\; 1\; [cons\; 2\; nil]]\; l].$$

Observe that this formula contains the higher-order term $\lambda x.[g\; x\; 1]$.

3.4. Example. Let *primrel* be a parameter of type $(i \rightarrow i \rightarrow o) \rightarrow o$ and let *rel*, *wife*, *mother*, *jane*, and *mary* be parameters of appropriate types. The following are then definite sentences:

$$[\top \supset [mother\; jane\; mary]],\; [\top \supset [wife\; john\; jane]],$$
$$[\top \supset [primrel\; mother]],\; [\top \supset [primrel\; wife]],\; \forall r.[[primrel\; r] \supset [rel\; r]],$$
$$\forall r.\forall s.[[[primrel\; r] \wedge [primrel\; s]] \supset [rel\; [\lambda x.\lambda y.\exists z.[[r\; x\; z] \wedge [s\; z\; y]]]]].$$

Observe that the last definite sentence contains the predicate variables r and s. Further, the complex term that appears as the argument of the predicate *rel* in this definite sentence contains an existential quantifier and a conjunction. The formula

$$\exists r.[[rel\; r] \wedge [r\; john\; mary]]$$

is a goal formula in this context. It is a goal formula in which a predicate variable occurs "extensionally", *i.e.* in a position where a substitution made for it can affect the top-level logical structure of the formula.

In the next section, we examine some of the properties of higher-order definite sentences and goal formulas. To preview the main results there, we introduce the following definition.

3.5. Definition. A substitution φ is a positive substitution if its range is contained in $\mathcal{H}^+$. It is a closed positive substitution if its range is contained in $\mathcal{HB}$.

Now let $\mathcal{P}$ be a finite collection of definite sentences, and let G be a goal formula whose free variables are contained in the listing $\bar{x}$. We shall see, then, that $\exists \bar{x}.G$ is provable from $\mathcal{P}$ just in case there is a closed positive substitution φ for $\bar{x}$ such that $\varphi(G)$ is provable from $\mathcal{P}$. This observation shall also facilitate the description of a simple proof procedure that may be used to extract substitutions such as φ. From these observations it will follow that our definite sentences and goal formulas provide the basis for a generalization to the programming paradigm of first-order logic.

4. Properties of Higher-Order Definite Sentences

As observed in Section 3, one difference between the top-level logical structure of first-order and higher-order goal formulas is that, in the higher-order context, predicate variables may appear as the heads of atomic goals. A consequence of this difference is that some of the proofs that can be constructed for goal formulas from definite sentences in the higher-order setting have quite a different character from any in the first-order case. An illustration of this fact is provided by the following derivation of the goal formula $\exists y.[P\,y]$ from the definite sentence $\forall x.[x \supset [P\,A]]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[P\,B] \longrightarrow [P\,B]}{[P\,B] \longrightarrow \exists y.[P\,y]}\ \Sigma\text{−IS}}{\longrightarrow \exists y.[P\,y], \sim[P\,B]}\ \sim\text{−IS} \qquad \dfrac{[P\,A] \longrightarrow [P\,A]}{[P\,A] \longrightarrow \exists y.[P\,y]}\ \Sigma\text{−IS}}{\sim[P\,B] \supset [P\,A] \longrightarrow \exists y.[P\,y], \exists y.[P\,y]}\ \supset\text{−IA}}{\sim[P\,B] \supset [P\,A] \longrightarrow \exists y.[P\,y]}\ \text{Contraction}}{\forall x.[x \supset [P\,A]] \longrightarrow \exists y.[P\,y]}\ \Pi\text{−IA}$$

It is easily observed that the top-level logical structure of a first-order formula remains invariant under any substitutions that are made for its free variables. It may then be seen that in a derivation whose endsequent contains only first-order definite formulas in the antecedent and only first-order goal formulas in the succedent, every other sequent must have only definite formulas and atoms in the antecedent and goal formulas in the succedent. This is, however, not the case in the higher-order context, as illustrated by the above derivation. Consider, for instance, the penultimate sequent in this derivation that is reproduced below:

$$\sim[P\,B] \supset [P\,A] \longrightarrow \exists y.[P\,y]. \qquad (*)$$

The formula $\sim[P\,B] \supset [P\,A]$ that appears in the antecedent of this sequent is neither a definite formula nor an atom. Looking closely at this sequent also reveals why this might be a cause for concern from the perspective of our current endeavor. Although this sequent has a derivation, we observe that there is no term t such that

$$\sim[P\,B] \supset [P\,A] \longrightarrow [P\,t]$$

has a derivation. Now, if all derivations of

$$\forall x.[x \supset [P\,A]] \;\longrightarrow\; \exists y.[P\,y]$$

involve the derivation of $(*)$, or of sequents similar to $(*)$ in the sense just outlined, then there would be no proof of $\exists y.[P\,y]$ from $\exists x.[x \supset [P\,A]]$ from which an "answer" may be extracted. Thus, it would be the case that one of the properties critical to the computational interpretation of definite sentences is false.

We show in this section that problems of the sort alluded to in the previous paragraph do not arise, and that, in fact, higher-order definite sentences and goal formulas resemble the corresponding first-order formulas in several proof-theoretic senses. The method that we adopt in demonstrating this may be described as follows. Let us identify the following inference figure schemata

$$\frac{\rho(P\,C), \Gamma \;\longrightarrow\; \Delta}{\Pi\,P, \Gamma \;\longrightarrow\; \Delta}\;\Pi{-}\mathrm{IA}^{+}$$

$$\frac{\Gamma \;\longrightarrow\; \Delta, \rho(P\,C)}{\Gamma \;\longrightarrow\; \Delta, \Sigma\,P}\;\Sigma{-}\mathrm{IS}^{+}$$

where we expect only closed positive formulas, *i.e.* formulas from $\mathcal{HB}$, to be substituted for C. These are evidently subcases of $\Pi{-}$IA and $\Sigma{-}$IS. We shall show that if there is any derivation at all of a sequent $\Gamma \;\longrightarrow\; \Delta$ where Γ consists only of definite sentences and Δ consists only of closed goal formulas, then there is one in which every inference figure obtained from $\Pi{-}$IA or $\Sigma{-}$IS is also an instance of $\Pi{-}\mathrm{IA}^{+}$ and $\Sigma{-}\mathrm{IS}^{+}$ respectively. These schemata are of interest because in any of their instances, if the lower sequent has only closed positive atoms and definite sentences in the antecedent and closed goal formulas in the succedent then so too does the upper sequent. Derivations of the sort mentioned, therefore, bear structural similarities to those in the first-order case, a fact that may be exploited to show that higher-order definite sentences and goal formulas retain many of the computational properties of their first-order counterparts.

The result that we prove below is actually of interest quite apart from the purposes of this paper. The so-called cut-elimination theorems have been of interest in the context of logic because they provide an insight into the nature of deduction and often are the basis for its mechanization. In the context of first-order logic, for instance, this theorem leads to the *subformula* property: if a sequent has a derivation, then it has one in which every formula in any intermediate sequent is a subformula of some formula in the endsequent. Several other useful structural properties of deduction in the first-order context flow from this observation, and the traditional proof procedures for first-order logic are based on it. As is evident from the example at the beginning of this section, the subformula property does not hold (under any acceptable interpretation) for higher-order logic even though the logic admits a cut-elimination theorem; predicate terms containing connectives and quantifiers may be generalized upon in the course of a derivation, and thus intermediate sequents may have formulas whose structure cannot be predicted from the formulas in the final one. For this reason, the usefulness of cut-elimination as a mechanism for analyzing and automating deduction in higher-order logic has been generally doubted. However, Theorem 4.1 below shows that there is useful structural information about proofs in higher-order logic to be obtained from the cut-elimination theorem for this logic, and is one of few results of this sort. This theorem permits us to observe that in the restricted setting of higher-order Horn clauses proofs for every derivable sequent can be constructed without ever having to generalize on predicate terms containing the logical constants $\sim$, $\supset$, and Π. This observation in turn provides information about the structure of each sequent in a derivation and constitutes a restriction on substitution terms that is sufficient to enable the description of a complete theorem proving procedure for higher-order Horn clauses.

A Simplified Sequent Calculus. Part of the discussion above is given a precise form by the following theorem:

4.1. Theorem. *Let Γ be a sequence of formulas that are either definite sentences or closed positive atoms of type o, and let Δ be a sequence of closed goal formulas. Then $\Gamma \longrightarrow \Delta$ has an LKH-derivation only if it has one in which*

(i) the only inference figures that appear are Contraction, Interchange, Thinning, $\vee-$IS, $\wedge-$IS, $\Sigma-$IS, $\supset-$IA *and* $\Pi-$IA, *and*

(ii) each occurrence of the figure $\Pi-$IA *or* $\Sigma-$IS *is also an occurrence of the figure* $\Pi-\mathrm{IA}^+$ *or* $\Sigma-\mathrm{IS}^+$.

The proof of this theorem is obtained by describing a transformation from an arbitrary LKH-derivation of $\Gamma \longrightarrow \Delta$ into a derivation of the requisite kind. This transformation is performed by replacing the formulas in each sequent by what we might call their "positive correlates" and by removing certain parts of the derivation that become unnecessary as a result. The following definition describes the manner in which the formulas in the succedent of each sequent are transformed.

4.2. Definition. Let $x, y \in \mathcal{V}ar_o$ and, for each α, let $z_\alpha \in \mathcal{V}ar_{(\alpha\to o)}$. The function *pos* on formulas is then defined as follows:

(i) If F is a constant or a variable

$$pos(F) = \begin{cases} \lambda x.\top, & \text{if } F \text{ is } \sim; \\ \lambda x.\lambda y.\top, & \text{if } F \text{ is } \supset; \\ \lambda z_\alpha.\top, & \text{if } F \text{ is } \Pi \text{ of type } ((\alpha \to o) \to o); \\ F, & \text{otherwise.} \end{cases}$$

(ii) $pos([F_1\, F_2]) = [pos(F_1)\, pos(F_2)]$.

(iii) $pos(\lambda x.F) = \lambda x.pos(F)$.

Further, *pc* is the mapping on formulas defined as follows: If F is a formula, $pc(F) = \rho(pos(F))$.

From the definition above it follows easily that $\mathcal{F}(pc(F)) \subseteq \mathcal{F}(pos(F)) = \mathcal{F}(F)$. Thus if F is a closed formula, then $pc(F) \in \mathcal{HB}$. The properties of *pc* stated in the following lemma are similarly easily argued for; these properties will be used in the proof of the main theorem.

4.3. Lemma. *Let F be a λ-normal formula of type o.*

(i) If F is an atom, then $pc(F)$ is a positive atom.

(ii) If F is $\sim F_1$, $F_1 \supset F_2$, or $\Pi\, P$, then $pc(F) = \top$.

(iii) If F is $G * H$ where $*$ is either $\vee$ or $\wedge$, then $pc(F) = pc(G) * pc(H)$.

(iv) If F is $\Sigma\, P$, then $pc(F) = \Sigma\, pc(P)$

In the lemma below, we argue that *pc* and ρ commute as operations on formulas. This observation is of interest because it yields the property of *pc* that is stated in Corollary 4.5 and is needed in the proof of the Theorem 4.1.

4.4. Lemma. *For any formula F, $pc(\rho(F)) = \rho(pc(F))$.*

Proof. Given any formula B of the same type as x, an easy induction on the structure of a formula G verifies the following facts: If B is free for x in G, then $pos(B)$ is free for x in $pos(G)$ and $pos(S^x_B\, G) = S^x_{pos(B)}\, pos(G)$. Now, let us say that a formula H results *directly* from a formula G by an application of a rule of λ-conversion if the subformula of G that is replaced by virtue of one of these rules is G itself. We claim that if H does indeed result from G in such a manner, then $pos(H)$ also results from $pos(G)$ in a similar manner. This claim is easily proved from the following observations:

(i) If G is $\lambda x.G_1$ and H is $\lambda y.S^x_y\, G_1$, $pos(G)$ is $\lambda x.pos(G_1)$ and $pos(H)$ is $\lambda y.S^x_y\, pos(G_1)$. Further if y is free for x in G_1 then y is free for x in $pos(G_1)$.

(ii) If G is $[\lambda x.G_1]\, G_2$ and H is $S^x_{G_2}\, G_1$ then $pos(F)$ is $[[\lambda x.pos(G_1)]\, pos(G_2)]$ and $pos(H)$ is $S^x_{pos(G_2)}\, pos(G_1)$. Further if G_2 is free for x in G_1 then $pos(G_2)$ is free for x in $pos(G_1)$.

(iii) If G is of type $\alpha \to \beta$ then $pos(G)$ is also of type $\alpha \to \beta$. If G is $\lambda y.[H\, y]$ then $pos(G)$ is $\lambda y.[pos(H)\, y]$ and, further, $y \in \mathcal{F}(pos(H))$ only if $y \in \mathcal{F}(H)$.

Now let F' result from F by a rule of λ-conversion. By making use of the claim above and by inducing on the structure of F, it may be shown that $pos(F')$ results from $pos(F)$ by a similar rule. From this it follows that $pos(\rho(F))$ results from $pos(F)$ by the application of the rules of λ-conversion. Hence $\rho(pos(\rho(F))) = \rho(pos(F))$. Noting further that $\rho(pos(F)) = \rho(\rho(pos(F)))$, the lemma is proved. ▮

4.5. Corollary. *If P and C are formulas of appropriate types, then*

$$pc(\rho([P\,C])) = \rho([pc(P)\,pc(C)]).$$

Proof. The claim is evident from the following equalities:

$$\begin{aligned} pc(\rho([P\,C])) &= \rho(pc([P\,C])) && \text{by Lemma 4.4} \\ &= \rho([pos(P)\,pos(C)]) && \text{using definitions} \\ &= \rho([\rho(pos(P))\,\rho(pos(C))]) && \text{by properties of } \lambda\text{-conversion} \\ &= \rho([pc(P)\,pc(C)]) && \text{using definitions. ▮} \end{aligned}$$

While the mapping pc will adequately serve the purpose of transforming formulas in the succedent of each sequent in the proof of Theorem 4.1, it does not suffice as a means for transforming formulas in the antecedent. It may be observed that if F is a definite formula then $pc(F) = \top$. The transformation to be effected on formulas in the antecedent must be such that it preserves definite formulas. In order to describe such a mapping, we find it useful first to identify the following class of formulas that contains the class of definite formulas.

4.6. Definition. A formula of type o is an *implicational* formula just in case it has one of the following forms

(i) $F \supset A$, where F and A are λ-normal formulas and in addition A is a rigid atom, or

(ii) $\forall x.F$, where F is itself an implicational formula.

We now define a function on implicational formulas whose purpose is to transform these formulas into definite formulas.

4.7. Definition. The function pos_i on implicational formulas is defined as follows

(i) If F is $H \supset A$ then $pos_i(F) = pos(H) \supset pos(A)$.

(ii) If F is $\forall x.F_1$ then $pos_i(F) = \forall x.pos_i(F_1)$

If F is an implicational formula then $pc_i(F) = \rho(pos_i(F))$.

From the definition, it is evident that if F is a definite formula, $pos_i(F) = F$ and, hence, $pc_i(F) = \rho(F)$. While this is not true when F is an arbitrary implicational formula, it is clear that $pc_i(F)$ is indeed a definite formula. The following lemma states an additional property of pc_i that will be useful in the proof of Theorem 4.1.

4.8. Lemma. *If F is an implicational formula and x and C are, respectively, a variable and a formula of the same type, then*

$$pc_i(\rho([\lambda x.F]\,C)) = \rho([\lambda x.pc_i(F)]\,pc(C)).$$

Proof. An easy induction on the structure of F shows that $\rho([\lambda x.F]\,C)$ is an implicational formula. Hence pos_i is defined on this formula and, consequently, the left-hand side of the equality is defined. We now claim that

$$pos_i(\rho([\lambda x.F]\,C)) \ \lambda\text{-conv}\ [\lambda x.pos_i(F)]\,pos(C).$$

Given this claim, it is clear that

$$\rho(pos_i(\rho([\lambda x.F]\,C))) = \rho([\lambda x.pos_i(F)]\,pos(C),$$

and the lemma follows by observing that $\rho([\lambda x.A]\,B) = \rho([\lambda x.\rho(A)]\,\rho(B))$.

Thus, it only remains to show the claim. Given any implicational formula F_1, we observe that if $F_1 \equiv F_2$ then $pos_i(F_1) \equiv pos_i(F_2)$. In trying to show the claim, we may therefore assume that the bound variables of F are distinct from x and from the free variables of C. Making such an assumption, we now induce on the structure of F:

(a) F is of the form $H \supset A$. In this case $\rho([\lambda x.F]\,C) \equiv \rho(S^x_C\,H) \supset \rho(S^x_C\,A)$. Using the definition of pos_i and arguments similar to those in Lemma 4.4, we see that

$$pos_i(\rho(S^x_C\,H) \supset \rho(S^x_C\,A)) \quad \lambda\text{-conv} \quad S^x_{pos(C)}\,pos(H) \supset S^x_{pos(C)}\,pos(A)$$

The claim easily follows from these observations.

(b) F is of the form $\forall y.F'$. Since the bound variables of F are distinct from x and the free variables of C, we see that

$$\rho([\lambda x.F]\,C) \quad \equiv \quad \forall y.\rho([\lambda x.F']\,C).$$

Using the inductive hypothesis and the definitions of λ-conversion and of pos_i we see that

$$pos_i(\rho([\lambda x.F]\,C)) \quad \lambda\text{-conv} \quad \forall y.[[\lambda x.pos_i(F')]\,pos(C)].$$

Observing that $\mathcal{F}(C) = \mathcal{F}(pos(C))$, it is clear that

$$\forall y.[[\lambda x.pos_i(F')]\,pos(C)] \quad \lambda\text{-conv} \quad [\lambda x.[\forall y.pos_i(F')]]\,pos(C).$$

The claim is now apparent from the definition of pos_i. ∎

Using the two mappings pc and pc_i, the desired transformation on sequents may now be stated. This is the content of the following definition.

4.9. Definition. First, we extend pc_i to the class of all formulas of type o:

$$pc_o(F) = \begin{cases} pc_i(F), & \text{if } F \text{ is an implicational formula;} \\ pc(F), & \text{otherwise.} \end{cases}$$

The mapping pc_s on sequents is then defined as follows: $pc_s(\Gamma \longrightarrow \Delta)$ is the sequent that results by replacing each formula F in Γ by $pc_o(F)$ and each formula G in Δ by $pc(G)$.

We are now in a position to describe, in a precise manner, the transformation of LKH-derivations alluded to immediately after the statement of Theorem 4.1. We do this below, thereby proving the theorem.

Proof of Theorem 4.1 We assume initially that every formula in Γ and Δ is in principal normal form; we indicate how this requirement may be relaxed at the end. Now, a simple inductive argument on the heights of derivations convinces us of the following fact: If Θ' and Λ' are finite sequences of formulas that are obtained from Θ and Λ, respectively, by replacing each formula by one of its λ-normal forms, then $\Theta \longrightarrow \Lambda$ has a derivation only if $\Theta' \longrightarrow \Lambda'$ has a derivation in which the inference figure λ does not appear. Since every formula in Γ and Δ is in λ-normal form, we may assume that $\Gamma \longrightarrow \Delta$ has a derivation in which no use is made of the inference figure λ. Further, since every formula in Γ and Δ is closed, we may assume that each instance of the schemata Π−IA and Σ−IS in this derivation is obtained by substituting a closed formula for C; if a variable y does appear free in the formula substituted for C, then we replace each free occurrence of y in it and in the sequents in the derivation by a parameter of the same type as y that does not already appear in the derivation; it is easy to see that the result is still a derivation of the same kind, and that its endsequent is still $\Gamma \longrightarrow \Delta$. Let $\mathbf{T}$ be such a derivation. We show below that $\mathbf{T}$ can be transformed into a derivation satisfying the requirements of the theorem.

To define the transformation, we need to distinguish between what we call the *essential* and the *inessential* sequents in $\mathbf{T}$. A sequent (occurrence) is considered inessential if it appears above the lower sequent of a Π−IS, $\supset$−IS or $\sim$−IS inference figure along any path in the derivation. A sequent is essential if it is not inessential. We claim that every formula in the antecedent of an essential sequent is either a rigid atom or an implicational formula. Observe that from this claim it also follows that each essential sequent, except the endsequent, is the upper sequent of one of the figures *Contraction*, *Interchange*, *Thinning*, $\vee$−IS, $\wedge$−IS, Σ−IS, $\supset$−IA or Π−IA.

The claim is proved by inducing on the distance of an essential sequent from the endsequent. If this distance is 0, the claim is obviously true. Assuming then that the claim is true if the distance is d, we verify it for distance $d+1$. Given the inductive hypothesis, we only need to consider the cases when the sequent in question is the upper sequent of an inference figure in which there is a formula in the antecedent of an upper sequent that is not in the antecedent of the lower sequent, *i.e.* one of the figures $\sim$−IS, $\vee$−IA, $\wedge$−IA, $\supset$−IA, $\supset$−IS, Π−IA, and Σ−IA. The cases of $\sim$−IS and $\supset$−IS are ruled out, given that we are considering essential sequents. Also, since the antecedent of the lower sequent contains only implicational formulas and rigid atoms, the figure in question cannot be one of $\vee$−IA, $\wedge$−IA, or Σ−IA. The only cases that remain are $\supset$−IA and

Π−IA. In the first case, *i.e.* when the inference figure in question is

$$\frac{\Theta_1 \longrightarrow \Lambda_1, F \qquad\qquad G, \Theta_2 \longrightarrow \Lambda_2}{F \supset G, \Theta_1, \Theta_2 \longrightarrow \Lambda_1, \Lambda_2}$$

the principal formula must be an implicational formula, and G must therefore be a rigid atom. From this it is clear that the claim holds in this case. If the inference figure is Π−IA, *i.e.* of the form

$$\frac{\rho(P\,C), \Theta \longrightarrow \Lambda}{\Pi\, P, \Theta \longrightarrow \Lambda}$$

the principal formula must again be an implicational formula, and so P must be of the form $\lambda x.F$ where F is an implicational formula. But then it is easily seen that $\rho(P\,C)$ must be an implicational formula and the claim is verified.

Now let $e(\mathbf{T})$ be the structure that results from removing all the inessential sequents from $\mathbf{T}$; $e(\mathbf{T})$ is a derivation of $\Gamma \longrightarrow \Delta$ but for the fact that some of its leaf sequents are not axioms: Such sequents are of the form $\Xi \longrightarrow \Phi, P$, where P is $\sim F$, $\Pi\, B$ or $F \supset G$. Let $pe(\mathbf{T})$ be the result of replacing each $\Theta \longrightarrow \Lambda$ in $e(\mathbf{T})$ by $pc_s(\Theta \longrightarrow \Lambda)$. We claim that each pair of upper sequent(s) and lower sequent in $pe(\mathbf{T})$ is an instance of the same inference figure schema as the corresponding pair in $e(\mathbf{T})$. To show this, we consider each of the possible cases in $e(\mathbf{T})$ and check the corresponding pairs in $pe(\mathbf{T})$. The claim is easily verified if the pair is an instance of *Contraction*, *Interchange* or *Thinning*. The cases for $\vee$−IS and $\wedge$−IS are similarly clear, given Lemma 4.3. If the pair is an instance of $\supset$−IA, the principal formula is an implicational formula of the form $F \supset A$. Observing then that $pc_i(F \supset A) = pc(F) \supset pc(A)$, the claim follows in this case as well. If the pair from $e(\mathbf{T})$ is an instance of Σ−IS, *i.e.* of the form

$$\frac{\Theta \longrightarrow \Lambda, \rho(P\,C)}{\Theta \longrightarrow \Lambda, \Sigma\, P}$$

the claim follows from Lemma 4.3 and Corollary 4.5; $pc(\Sigma\, P) = \Sigma\, pc(P)$ and

$$pc(\rho(P\,C)) = \rho(pc(P)\, pc(C)).$$

We note further that, since C is a closed formula, $pc(C) \in \mathcal{HB}$ and so the corresponding figure in $pe(\mathbf{T})$ is actually an instance of Σ−IS^{+}. Finally, let the pair in $e(\mathbf{T})$ be an instance of Π−IA, *i.e.* of the form

$$\frac{\rho(P\,C), \Theta \longrightarrow \Lambda}{\Pi\, P, \Theta \longrightarrow \Lambda}$$

By our earlier observations $\Pi\, P$ is an implicational formula, and hence P is of the form $\lambda x.F$, where F is an implicational formula. Using Lemma 4.8, we see that

$$pc_i(\rho((\lambda x.F)\,C)) = \rho([\lambda x.pc_i(F)]\,pc(C)).$$

Noting now that $pc_i(\Pi\,[\lambda x.F]) \equiv [\Pi\,[\lambda x.pc_i(F)]]$ and that $pc(C) \in \mathcal{HB}$, it is clear that the corresponding pair in $pe(\mathbf{T})$ is also an instance of Π−IA, and in fact of Π−IA$^+$.

Given the forms of formulas in Γ and Δ, we observe that $pc_s(\Gamma \longrightarrow \Delta) = \Gamma \longrightarrow \Delta$. We also note that if A is an atomic formula and $A \equiv A'$, then $pc_o(A) = pc(A')$. Thus we may conclude, from the above considerations and Lemma 4.3, that $pe(\mathbf{T})$ would be a derivation of $\Gamma \longrightarrow \Delta$ of the sort required by the theorem but for the fact that some of its leaf sequents are of the form $\Theta \longrightarrow \Lambda, \top$. However, we may adjoin derivations of the form

$$\begin{array}{c} \longrightarrow\ \top \\ \hline\hline \Theta \longrightarrow \Lambda, \top \end{array} \quad \begin{array}{l}\text{sequence of Thinnings}\\ \text{and Interchanges}\end{array}$$

above sequents of this sort to obtain a genuine LKH-derivation that satisfies the theorem.

The above argument is adequate in the case that each formula in Γ and Δ is in principal normal form. If this is not the case, then we proceed as follows. First we construct a derivation of the requisite sort for $\Gamma' \longrightarrow \Delta'$, where Γ' and Δ' are obtained from Γ and Δ respectively by placing each formula in principal normal form. A simple inductive argument then suffices to show that this derivation may be converted into one for $\Gamma \longrightarrow \Delta$ by replacing some of the formulas in the derivation by formulas that they convert to via the rule of α-conversion.

4.10. Example. An illustration of the transformation described in the proof of Theorem 4.1 may be provided by considering the derivation presented at the beginning of this section. This would be transformed into the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\longrightarrow\ \top}{\longrightarrow\ \top, \exists y.[P\,y]}\ \text{Thinning}}{\longrightarrow\ \exists y.[P\,y], \top}\ \text{Interchange} \qquad \dfrac{[P\,A] \longrightarrow [P\,A]}{[P\,A] \longrightarrow \exists y.[P\,y]}\ \Sigma\text{−IS}}{\top \supset [P\,A] \longrightarrow \exists y.[P\,y], \exists y.[P\,y]}\ \supset\text{−IA}}{\top \supset [P\,A] \longrightarrow \exists y.[P\,y]}\ \text{Contraction}}{\forall x.[x \supset [P\,A]] \longrightarrow \exists y.[P\,y]}\ \Pi\text{−IA}$$

The content of Theorem 4.1 may be expressed by the description of a simplified sequent calculus for definite sentences and goal formulas. Let $LKHD$ be the subcalculus of LKH with exactly the same initial sequents but with inference figures restricted to being instances of the schemata *Contraction*, *Interchange*, *Thinning*, $\vee-$IS, $\wedge-$IS, $\Sigma-$IS$^+$, $\supset-$IA, and $\Pi-$IA$^+$. In the discussions below we shall be concerned with derivations for sequents of the form $\Gamma \longrightarrow \Delta$, where Γ is any finite sequence of definite sentences and closed positive atoms and Δ is a finite sequence of closed goal formulas. By virtue of Theorem 4.1 we see that such a sequent has an LKH-derivation exactly when it has an $LKHD$-derivation. In considering questions about derivations for such a sequent we may, therefore, restrict our attention to the calculus $LKHD$, and we implicitly do so below.

Proofs from Higher-Order Definite Sentences. We now use the preceding results to demonstrate that higher-order definite sentences and goal formulas retain the proof-theoretic properties of the corresponding first-order formulas that were discussed in Section 1. In this endeavor, we use the characteristics of the higher-order formulas observed in the two lemmas below. The first lemma subsumes the statement that a finite set of higher-order definite sentences is consistent.

4.11. Lemma. *If Γ is a finite sequence of definite sentences and closed positive atoms, then there can be no derivation for $\Gamma \longrightarrow$.*

Proof. Suppose the claim is false. Then there is a least h and a Γ of the requisite sort such that $\Gamma \longrightarrow$ has a derivation of height h. Since $\Gamma \longrightarrow$ is not an initial sequent, h is evidently not 1. Consider now by cases the inference figures of which $\Gamma \longrightarrow$ could be the lower sequent, *i.e.* the figures *Contraction*, *Thinning* and *Interchange* in the antecedent, $\Pi-$IA$^+$, and $\supset-$IA. In each of these cases it is easily observed that there must be a finite sequence of definite sentences and closed positive atoms Γ' such that $\Gamma' \longrightarrow$ has a derivation of height $< h$. This contradicts the leastness of h. ▮

The lemma below relates the notions of classical and intuitionistic provability of a goal formula from a set of definite sentences.

4.12. Lemma. *Let Γ be a finite sequence of definite sentences and closed positive atoms and let G be a closed goal formula. Then $\Gamma \longrightarrow G$ has a derivation only if it has one in which there is at most one formula in the succedent of each sequent.*

Proof. We claim that a slightly stronger statement is true: A sequent of the form $\Gamma \longrightarrow G_1, \ldots, G_n$, where Γ consists only of definite sentences and closed positive atoms and $G_1, \ldots, G_n$ are closed goal formulas, has a derivation only if there is an i such that $1 \leq i \leq n$ and $\Gamma \longrightarrow G_i$ has a derivation in which at most one formula appears in the succedent of each sequent.

The claim is proved by an induction on the heights of derivations for sequents of the sort hypothesized in it. If the height is 1, then $n = 1$ and the claim is obviously true. Let us, therefore, assume the height is $h + 1$ and consider the possible cases for the inference figure that appears at the end of the derivation. If this figure is a *Contraction*, *Thinning* or *Interchange* in the succedent, the claim follows directly from the inductive hypothesis; in the case of *Thinning*, we only need to observe that, by Lemma 4.11, $n > 1$. The cases of *Contraction*, *Thinning*, and *Interchange* in the antecedent, $\vee$−IS, $\wedge$−IS, Σ−IS$^+$ and Π−IA$^+$, also follow from the hypothesis with a little further argument. Consider, for instance the case when the figure is an $\wedge$−IS. The derivation at the end then has the following form:

$$\frac{\Gamma \longrightarrow G_1, \ldots, G_{n-1}, G_n^1 \qquad\qquad \Gamma \longrightarrow G_1, \ldots, G_{n-1}, G_n^2}{\Gamma \longrightarrow G_1, \ldots, G_{n-1}, G_n^1 \wedge G_n^2}$$

By the hypothesis, there must be a derivation of the requisite sort either for $\Gamma \longrightarrow G_i$ for some i, $1 \le i \le n-1$, or for both $\Gamma \longrightarrow G_n^1$ and $\Gamma \longrightarrow G_n^2$. In the former case the claim follows directly, and in the latter case we use the two derivations together with an $\wedge$−IS inference figure to construct a derivation of the requisite sort for $\Gamma \longrightarrow G_n^1 \wedge G_n^2$.

The only remaining case is that of $\supset$−IA, *i.e.* when the inference figure in question is of the form

$$\frac{\Gamma_1 \longrightarrow G_1, \ldots, G_k, G \qquad\qquad A, \Gamma_2 \longrightarrow G_{k+1}, \ldots, G_n}{G \supset A, \Gamma_1, \Gamma_2 \longrightarrow G_1, \ldots, G_n}$$

From Lemma 4.11 it follows that $k < n$. By the hypothesis, we see that there is a derivation of the requisite sort either for $\Gamma_1 \longrightarrow G_i$ for some i between 1 and k or for $\Gamma_1 \longrightarrow G$. In the former case, by adjoining a sequence of *Thinning* and *Interchange* figures below the derivation for $\Gamma_1 \longrightarrow G_i$ we obtain the required derivation for $G \supset A, \Gamma_1, \Gamma_2 \longrightarrow G_i$. In the latter case, using the induction hypothesis again we see that there is a derivation of the required sort for $A, \Gamma_2 \longrightarrow G_j$ for some j between $k+1$ and n. This derivation may be combined with the one for $\Gamma_1 \longrightarrow G$ to obtain the required derivation for $G \supset A, \Gamma_1, \Gamma_2 \longrightarrow G_j$. ∎

Lemmas 4.11 and 4.12 permit us to further restrict our sequent calculus in the context of definite sentences. To make the picture precise, let us assume that Γ is a finite sequence of definite sentences and closed positive atoms, and that G is a goal formula such that $\Gamma \longrightarrow G$ has a derivation. Then, using Lemma 4.12 we see that this sequent has a derivation in which there is no occurrence of *Contraction* or *Interchange* in the succedent. Using Lemma 4.11 it may be seen that such a derivation is also one in which the figure *Thinning* in the succedent does not appear. Thus, in considering questions about derivability for sequents of the sort described above, we may dispense with the structural inference figures pertaining to the succedent. This fact is made use of in the proof of Theorem 4.14 and in the discussions that follow it. We present this theorem after introducing a convenient notational convention.

4.13. Definition. Let D be the definite sentence $\forall \bar{x}.G \supset A$. Then $|D|$ denotes the set of all closed positive instances of $G \supset A$, *i.e.*

$$|D| = \{\varphi(G \supset A) \mid \varphi \text{ is a closed positive substitution for } \bar{x}\}.$$

This notation is extended to sets of definite sentences: If $\mathcal{P}$ is such a set,

$$|\mathcal{P}| = \bigcup\{|D| \;\mid\; D \in \mathcal{P}\}.$$

From this definition it readily follows that $|D|$ and $|\mathcal{P}|$ are both collections of definite sentences.

4.14. Theorem. *Let Γ be a finite sequence of definite sentences and closed positive atoms, and let G be a closed goal formula. Then there is a derivation for $\Gamma \longrightarrow G$ if and only if*

(i) G is $G_1 \wedge G_2$ and there are derivations for $\Gamma \longrightarrow G_1$ and $\Gamma \longrightarrow G_2$, or

(ii) G is $G_1 \vee G_2$ and there is a derivation for either $\Gamma \longrightarrow G_1$ or $\Gamma \longrightarrow G_2$, or

(iii) G is $\Sigma\, P$ and there is a $C \in \mathcal{HB}$ such that $\Gamma \longrightarrow \rho(P\,C)$ has a derivation, or

(iv) G is an atom and either G is $\top$, or $G \equiv A$ for some A in Γ, or, for some definite sentence D in Γ, $G' \supset A \in |D|$, $G \equiv A$, and there is a derivation for $\Gamma \longrightarrow G'$.

Proof. ($\supset$) As we have noted, there is a derivation for a sequent of the sort described in the Theorem only if there is one in which there are no structural figures pertaining to the succedent. An induction on the heights of such derivations now proves the theorem in this direction.

If $\Gamma \longrightarrow G$ has a derivation of height 1, then G is $\top$ or Γ is A and $G \equiv A$. In either case the theorem is true. For the case when the height is $h+1$, we consider each possibility for the last inference figure in the derivation. If it is one of $\wedge$–IS, $\vee$–IS or Σ–IS$^+$, the theorem is evidently true. If it is a *Contraction* in the antecedent, *i.e.* of the form

$$\frac{F, F, \Gamma \longrightarrow G}{F, \Gamma \longrightarrow G}$$

we see that the upper sequent is of the kind described in the theorem and, in fact, has a derivation of height h. A recourse to the induction hypothesis now completes the proof. For instance, assume that G is of the form $G_1 \vee G_2$. By the hypothesis, there is a derivation for $F, F, \Gamma \longrightarrow G_i$ for $i = 1$ or $i = 2$. By adjoining below this derivation a *Contraction* in the antecedent, we obtain a derivation for $F, \Gamma \longrightarrow G_i$. The analysis for the cases when G has a different structure follows an analogous pattern.

In the cases when the last inference figure is a *Thinning* or an *Interchange* in the antecedent, the argument is similar to that of *Contraction.* If the figure is Π–IA$^+$, *i.e.* of the form

$$\frac{\rho(P\,C), \Gamma \longrightarrow G}{\Pi\, P, \Gamma \longrightarrow G}$$

we see that $\rho(P\,C)$ and $\Pi\,P$ are both definite sentences and, further, that $|\rho(P\,C)| \subseteq |\Pi\,P|$. Applying the induction hypothesis to the upper sequent, the proof may now be completed by arguments similar to those outlined in the case of *Contraction* in the antecedent.

The only remaining case is that of $\supset$ –IA. In this case, by Lemma 4.11 we observe that the derivation at the end has the following form

$$\frac{\Gamma_1 \longrightarrow G' \qquad\qquad A, \Gamma_2 \longrightarrow G}{G' \supset A, \Gamma_1, \Gamma_2 \longrightarrow G}$$

The right upper sequent of this inference figure is evidently of the form required for the induction hypothesis. Once again using arguments similar to those in the case of *Contraction* in the antecedent, the proof may be completed in all cases except when $G \equiv A$. But if $G \equiv A$, we observe that the theorem is still true, since a derivation for $G' \supset A, \Gamma_1, \Gamma_2 \longrightarrow G'$ may be constructed by adjoining a sequence of *Thinning* and *Interchange* inference figures below the derivation for $\Gamma_1 \longrightarrow G'$.

($\subset$) The only case that needs explicit consideration here is that when, for some definite sentence D in Γ, there is a $G' \supset A \in |D|$ such that $G \equiv A$ and $\Gamma \longrightarrow G'$ has a derivation. In this case a derivation for $\Gamma \longrightarrow G$ may be constructed as follows:

$$\begin{array}{ll}
\dfrac{\Gamma \longrightarrow G' \qquad\qquad A \longrightarrow G}{G' \supset A, \Gamma \longrightarrow G} & \supset\text{–IA} \\
\hline\hline
D, \Gamma \longrightarrow G & \text{sequence of } \Pi\text{–IA}^{+} \\
\hline\hline
\Gamma \longrightarrow G & \begin{array}{l}\text{sequence of Interchanges} \\ \text{and a Contraction}\end{array}
\end{array}$$

∎

In Section 1 we outlined the proof-theoretic properties of first-order definite sentences and goal formulas that play a pivotal role in their use as a basis for programming. Theorem 4.14 demonstrates that our higher-order generalizations of these formulas retain these properties. Thus, if G is a higher-order goal formula whose free variables are included in the listing $x_1, \ldots, x_n$ and $\mathcal{P}$ is a finite set of higher-order definite sentences, we see from clause (iii) of Theorem 4.14 that $\exists x_1 \ldots \exists x_n.G$ is provable from $\mathcal{P}$ just in case there is a closed positive substitution φ for $x_1, \ldots, x_n$ such that $\varphi(G)$ is provable from $\mathcal{P}$. Hence, sets of higher-order definite sentences and higher-order goal formulas may be construed, respectively, as *programs* and *queries* in a manner exactly analogous to the first-order case. Furthermore, clauses (i) – (iii) show that $\wedge$, $\vee$ and the existential quantifier provide primitives for search specification in exactly the same way as in the first-order setting. Finally, by virtue of (iv), higher-order definite sentences provide the basis for defining nondeterministic procedures. Notice that in a definite sentence of form $\forall \bar{x}.(G \supset A)$, the head of A must be a parameter, and construing this formula as a procedure defining this head of A, therefore, makes good sense.

Theorem 4.14 also provides the skeleton for a procedure that may be used for determining whether a goal formula is provable from a set of definite sentences. In essence, clauses (i) – (iii) describe the means by which the search for a proof of a complex goal formula may be reduced to the search for proofs of a set of atoms. The search for a proof of an atomic goal may be progressed by "backchaining" on definite sentences in the manner indicated by clause (iv). A precise description of this proof procedure requires the explication of the notion of higher-order unification, and we undertake this task in the next section. We note, however, that the steps mentioned above must simplify the search in some manner if they are to be effective in finding a proof. That they do have this effect may be seen by associating a measure, indexed by a finite set of definite sentences, with each goal formula. For this purpose, we identify the notion of a *reduced path* in a derivation as a sequence of sequents that results by the removal of the lower sequents of structural inference figures from a path in the derivation; intuitively, the length of a reduced path is a count of the number of operational inference figures that appear along the corresponding path. Letting the *true height* of a derivation be the length of the longest reduced path in the derivation, the required measure may be defined as follows.

4.15. Definition. Let Γ be a finite sequence of definite sentences, and let G be a closed goal formula. Further, let k be the least among the true heights of derivations for $\Gamma \longrightarrow G$ in which there appear no structural inference figures pertaining to the succedent; if no such derivation exists, $k = \omega$. Then

$$\mu_\Gamma(G) = \begin{cases} 2^k, & \text{if } k < \omega; \\ \omega, & \text{otherwise.} \end{cases}$$

The measure is extended to be relative to a finite set, $\mathcal{P}$, of definite sentences by defining $\mu_\mathcal{P}(G) = \mu_\Gamma(G)$ where Γ is a listing of the members of $\mathcal{P}$. This extension is clearly independent of the particular listing chosen.

The properties of this measure that are of interest from the perspective of describing a proof procedure are stated in the following Lemma.

4.16. Lemma. *Let $\mathcal{P}$ be a finite set of definite sentences and let G be a closed goal formula that is provable from $\mathcal{P}$. Then $\mu_\mathcal{P}(G) > 0$ and $\mu_\mathcal{P}(G) < \omega$. Further,*

(i) If G is an atom other than $\top$ then there is a $G' \supset G \in |\mathcal{P}|$ such that $\mu_\mathcal{P}(G') < \mu_\mathcal{P}(G)$.

(ii) If G is $G_1 \vee G_2$ then $\mu_\mathcal{P}(G_i) < \mu_\mathcal{P}(G)$ for $i = 1$ or $i = 2$.

(iii) If G is $G_1 \wedge G_2$ then $\mu_\mathcal{P}(G_1) + \mu_\mathcal{P}(G_2) < \mu_\mathcal{P}(G)$.

(iv) If G is ΣP then for some $C \in \mathcal{HB}$ it is the case that $\mu_\mathcal{P}(\rho(P\,C)) < \mu_\mathcal{P}(\Sigma P)$.

Proof. $\mu_{\mathcal{P}}(G)$ is obviously greater that 0. Since G is provable from $\mathcal{P}$, it is also obvious that $\mu_{\mathcal{P}}(G) < \omega$. Now let Γ be a finite sequence of definite sentences such that $\Gamma \longrightarrow G$ has a derivation of true height h. A reexamination of the proof of Lemma 4.14 reveals the following facts: If G is $G_1 \vee G_2$, then $\Gamma \longrightarrow G_i$ has a derivation of true height $< h$ for $i = 1$ or $i = 2$. If G is $G_1 \wedge G_2$, then $\Gamma \longrightarrow G_i$ has a derivation of true height $< h$ for $i = 1$ and $i = 2$. If G is $\Sigma\, P$, then there is a $C \in \mathcal{HB}$ such that $\Gamma \longrightarrow \rho(P\, C)$ has a derivation of true height $< h$. If G is an atom, then there is a $G' \supset G \in |\mathcal{P}|$ such that $\Gamma \longrightarrow G'$ has a derivation of true height $< h$. From these observations, the rest of the lemma follows easily.

5. Searching for Proofs from Definite Sentences

We now turn to the task of describing a procedure for determining whether there is a proof for the existential closure of a goal formula from a set of definite sentences. As already noted, the description of such a procedure requires a consideration of the problem of unifying two higher-order formulas. This problem has been studied by several researchers, and in most extensive detail by [15]. In the first part of this section, we summarize this problem and detail those aspects of its solution in [15] that are pertinent to our current endeavor. We then introduce the notion of a $\mathcal{P}$-derivation. $\mathcal{P}$-derivations are a generalization to the higher-order context of the notion of SLD-derivations described in [2] and prevalent in most discussions of first-order definite sentences. At one level, they are intended to be syntactic objects for demonstrating the existence of a proof for a goal formula and our discussions show their correctness from this perspective. At another level, they are intended to provide a basis for an actual proof procedure — a symbol manipulating procedure that searches for $\mathcal{P}$-derivations would constitute an interpreter for a programming paradigm that is based on higher-order definite sentences — and we explore some of their properties that are pertinent to the description of such a procedure.

The Higher-Order Unification Problem. Let us call a pair of formulas of the same type a *disagreement pair*. A *disagreement set* is then a finite set, $\{\langle F_i, H_i\rangle \mid 1 \leq i \leq n\}$, of disagreement pairs, and a *unifier* for the set is a substitution σ such that, for $1 \leq i \leq n$, $\sigma(F_i) = \sigma(H_i)$. The *higher-order unification problem* is then the following: Given a disagreement set, we desire to determine whether it has unifiers, and to explicitly provide a unifier if it does have one.

The problem described above is a generalization of the well-known unification problem for first-order terms. The higher-order unification problem has several properties that are, in a certain sense, divergent from those of the problem in the first-order case. For instance, the question of whether a unifier exists for an arbitrary disagreement set in the higher-order context is an undecidable question [10, 14, 17], whereas the corresponding question for first-order terms is decidable. As another example, it has been shown [12] that most general unifiers do not always exist for unifiable higher-order disagreement pairs. Despite these characteristics of the problem, a systematic search can be made for unifiers of a given disagreement set, and we discuss this aspect below.

Huet, in [15], describes a procedure for determining the existence of unifiers for a given disagreement set and shows that, whenever unifiers do exist, the procedure may be used to provide some of them. The basis for this procedure is in the fact that there are certain disagreement sets for which at least one unifier may easily be provided and, similarly, there are other disagreement sets for which it is easily manifest that no unifiers can exist. Given an arbitrary disagreement set, the procedure then attempts to reduce it to a disagreement set of one of these two kinds. This reduction proceeds by an iterative use of two kinds of simplifying functions, called SIMPL and MATCH, on disagreement sets. Since our notion of a $\mathcal{P}$-derivation uses these functions in an intrinsic way, we devote some effort to describing them below.

In presenting the functions SIMPL and MATCH and in analyzing their properties, the normal form for formulas that is introduced by the following definition is useful.

5.1. Definition. A β-normal form $\tilde{F} = \lambda x_1.\dots\lambda x_n.[H\, A_1\, \dots\, A_m]$ is said to be a *$\beta\eta$-long* form if the type of H is of the form $\alpha_1 \to \cdots \to \alpha_m \to \alpha_0$, where α_0 is an atomic type, and, for $1 \le i \le m$, A_i is also a $\beta\eta$-long form. If F is a formula such that F λ-conv $\tilde{F}$, then $\tilde{F}$ is said to be a *$\beta\eta$-long form* of F. Given a $\beta\eta$-long form $F = \lambda\bar{x}.[H\, A_1 \dots A_m]$, a count of the number of occurrences of applications in F is provided by the following recursively defined measure on F:

$$\xi(F) = m + \sum_{i=1}^{m} \xi(A_i).$$

It is clear that every formula has a $\beta\eta$-long form; such a form may be obtained by first converting the formula to a β-normal form, and then performing a sequence of η-expansions. We shall write $\tilde{F}$ below to denote a $\beta\eta$-long form of a formula F. The formula thus denoted is ambiguous only up to a renaming of bound variables. To see this, let $F_1 = \lambda x_1 \dots \lambda x_m.[H\, A_1\, \dots\, A_r]$ and $F_2 = \lambda y_1.\dots\lambda y_n.[H'\, B_1\, \dots\, B_s]$ be two $\beta\eta$-long forms such that F_1 λ-conv F_2. Observing that F_1 and F_2 must have the same λ-normal forms, it is clear that $\lambda x_1 \dots \lambda x_m.H \equiv \lambda y_1.\dots\lambda y_n.H'$, and hence $m = n$ and $r = s$. Furthermore, for all i, $1 \le i \le r$, it must be the case that $\lambda x_1 \dots \lambda x_m.A_i$ λ-conv $\lambda y_1 \dots \lambda y_m.B_i$. A simple argument by induction on the measure in Definition 5.1 then shows that $F_1 \equiv F_2$. This observation permits us to extend the measure ξ to arbitrary formulas. For any formula F, we may define $\xi(F) = \xi(\tilde{F})$. Given the uniqueness of $\beta\eta$-long forms up to α-conversions and the fact that $\xi(F_1) = \xi(F_2)$ for any $\beta\eta$-long forms F_1 and F_2 such that $F_1 \equiv F_2$, it follows that this extension of ξ is well-defined.

Given any formula F and any substitution σ, it is apparent that $\sigma(F) = \sigma(\tilde{F})$. The interest in the $\beta\eta$-long form of representation of formulas stems from the fact that the effects of substitutions on formulas in this form can be analyzed easily. As an instance of this, we observe the following lemma that is the basis of the first phase of simplification in the search for unifiers for a given disagreement set. In this lemma, and in the rest of this section, we use the notation $\mathcal{U}(\mathcal{D})$ to denote the set of unifiers for a disagreement set $\mathcal{D}$. A proof of this lemma is contained in [15] and also in [25].

5.2. Lemma. *Let $F_1 = \lambda\bar{x}[H_1\,A_1\,\ldots\,A_r]$ and $F_2 = \lambda\bar{x}[H_2\,B_1\,\ldots\,B_s]$ be two rigid $\beta\eta$-long forms of the same type. Then $\sigma \in \mathcal{U}(\{\langle F_1, F_2\rangle\})$ if and only if*

(i) $H_1 = H_2$ (and, therefore, $r = s$), and

(ii) $\sigma \in \mathcal{U}(\{\langle\lambda\bar{x}.A_i, \lambda\bar{x}.B_i\rangle \mid 1 \leq i \leq r\})$.

Let us say that F is rigid (flexible) just in case $\tilde{F}$ is rigid (flexible), and let us refer to the arguments of $\tilde{F}$ as the arguments of F. If F_1 and F_2 are two formulas of the same type, it is evident that $\beta\eta$-long forms of F_1 and F_2 must have binders of the same length. Furthermore, we may, by a sequence of α-conversions, arrange their binders to be identical. If F_1 and F_2 are both rigid, then Lemma 5.2 provides us a means for either determining that F_1 and F_2 have no unifiers or for reducing the problem of finding unifiers for F_1 and F_2 to that of finding unifiers for the arguments of these formulas. This, in fact, is the nature of the simplification effected on a given unification problem by the function SIMPL.

5.3. Definition. The function SIMPL on sets of disagreement pairs is defined as follows:

(1) If $\mathcal{D} = \emptyset$ then SIMPL$(\mathcal{D}) = \emptyset$.

(2) If $\mathcal{D} = \{\langle F_1, F_2\rangle\}$, and

  (a) if F_1 is a flexible formula then SIMPL$(\mathcal{D}) = \mathcal{D}$; otherwise

  (b) if F_2 is a flexible formula then SIMPL$(\mathcal{D}) = \{\langle F_2, F_1\rangle\}$;

  (c) otherwise F_1 and F_2 are both rigid formulas. Let $\lambda\bar{x}.[C_1\,A_1\,\ldots\,A_r]$ and $\lambda\bar{x}.[C_2\,B_1\,\ldots\,B_s]$ be $\beta\eta$-long forms for F_1 and F_2. If $C_1 \neq C_2$ then SIMPL$(\mathcal{D}) = \mathbf{F}$; otherwise SIMPL$(\mathcal{D}) =$ SIMPL$(\{\langle\lambda\bar{x}.A_i, \lambda\bar{x}.B_i\rangle \mid 1 \leq i \leq r\})$.

(3) Otherwise $\mathcal{D}$ has at least two members. Let $\mathcal{D} = \{\langle F_i, G_i\rangle \mid 1 \leq i \leq n\}$.

  (a) If SIMPL$(\{\langle F_i, G_i\rangle\}) = \mathbf{F}$ for some i then SIMPL$(\mathcal{D}) = \mathbf{F}$;

  (b) Otherwise SIMPL$(\mathcal{D}) = \bigcup_{i=1}^{n}$ SIMPL$(\{\langle F_i, G_i\rangle\})$.

Clearly, SIMPL transforms a given disagreement set into either the marker $\mathbf{F}$ or a disagreement set consisting solely of "flexible-flexible" or "flexible-rigid" formulas. By an abuse of terminology, we shall regard $\mathbf{F}$ as a disagreement set that has no unifiers. The intention, then, is that SIMPL transforms the given set into a simplified set that has the same unifiers. The following lemma shows that SIMPL achieves this purpose in a finite number of steps.

5.4. Lemma. SIMPL *is a total computable function on sets of disagreement pairs. Further, if $\mathcal{D}$ is a set of disagreement pairs then $\sigma \in \mathcal{U}(\mathcal{D})$ if and only if* $\mathrm{SIMPL}(\mathcal{D}) \neq \mathbf{F}$ *and* $\sigma \in \mathcal{U}(\mathrm{SIMPL}(\mathcal{D}))$.

Proof. We define a measure ψ on sets of disagreement pairs in the following fashion. If $\mathcal{D} = \{\langle F_i, G_i \rangle \mid 1 \leq i \leq n\}$, then

$$\psi(\mathcal{D}) = n + \sum_{i=1}^{n} \xi(F_i).$$

The lemma follows from Lemma 5.2 by an induction on this measure. ▮

The first phase in the process of finding unifiers for a given disagreement set $\mathcal{D}$ thus consists of evaluating $\mathrm{SIMPL}(\mathcal{D})$. If the result of this is $\mathbf{F}$, $\mathcal{D}$ has no unifiers. On the other hand, if the result is a set that is either empty or has only flexible-flexible pairs, at least one unifier can be provided easily for the set, as we shall see in the proof of Theorem 5.14; such a set is, therefore, referred to as a *solved set*. If the set has at least one flexible-rigid pair, then a substitution for the head of the flexible formula needs to be considered so as to make the heads of the two formulas in the pair identical. There are essentially two kinds of "elementary" substitutions that may be employed for this purpose. The first kind of substitution is the one that makes the head of the flexible formula "imitate" that of the rigid formula. In the context of first-order terms this is, in fact, the only kind of substitution that needs to be considered. If the head of the flexible formula is a higher-order variable, however, there is also another possibility. This is that of "projecting" one of the arguments of the flexible formula into the head position, in the hope that the head of the resulting formula becomes identical to the head of the rigid one or may be made so by a subsequent substitution. There are, thus, a set of substitutions, each of which may be investigated separately as a component of a complete unifier. The purpose of the function MATCH that is defined below is to produce these substitutions.

5.5. Definition. Let $\mathcal{V}$ be a set of variables, let F_1 be a flexible formula, let F_2 be a rigid formula of the same type as F_1, and let $\lambda\bar{x}.[f\, A_1 \,\ldots\, A_r]$, and $\lambda\bar{x}.[C\, B_1 \,\ldots\, B_s]$ be $\beta\eta$-long forms of F_1 and F_2. Further, let the type of f be $\alpha_1 \to \cdots \to \alpha_r \to \beta$, where β is primitive and, for $1 \leq i \leq r$, let w_i be a variable of type α_i. The functions IMIT, PROJ, and MATCH are then defined as follows:

(i) If C is a variable (appearing also in $\bar{x}$), then $\mathrm{IMIT}(F_1, F_2, \mathcal{V}) = \emptyset$; otherwise

$$\mathrm{IMIT}(F_1, F_2, \mathcal{V}) = \{\{\langle f, \lambda w_1 \,\ldots\, \lambda w_r.[C\; [h_1\, w_1 \,\ldots\, w_r] \ldots [h_s\, w_1 \ldots\, w_r]]\rangle\}\},$$

where $h_1, \ldots, h_s$ are variables of appropriate types not contained in $\mathcal{V} \cup \{w_1, \ldots, w_r\}$.

(ii) For $1 \leq i \leq r$, if α_i is not of the form $\beta_1 \to \ldots \to \beta_t \to \beta$ then $\mathrm{PROJ}_i(F_1, F_2, \mathcal{V}) = \emptyset$; otherwise,

$$\mathrm{PROJ}_i(F_1, F_2, \mathcal{V}) = \{\{\langle f, \lambda w_1 \ldots \lambda w_r.[w_i\ [h_1\ w_1 \ldots\ w_r] \ldots [h_t\ w_1 \ldots\ w_r]]\rangle\}\},$$

where $h_1, \ldots, h_t$ are variables of appropriate type not contained in $\mathcal{V} \cup \{w_1, \ldots, w_r\}$.

(iii) $\mathrm{MATCH}(F_1, F_2, \mathcal{V}) = \mathrm{IMIT}(F_1, F_2, \mathcal{V}) \cup (\bigcup_{1 \leq i \leq r} \mathrm{PROJ}_i(F_1, F_2, \mathcal{V}))$.

The purpose of MATCH is to suggest a set of substitutions that may form "initial segments" of unifiers and, in this process, bring the search for a unifier closer to resolution. To describe the sense in which MATCH achieves this purpose precisely, we need the following measure on substitutions:

5.6. Definition. Let $\varphi = \{\langle f_i, T_i\rangle \mid 1 \leq i \leq n\}$ be a substitution. We define a measure on φ as follows:

$$\pi(\varphi) = n + \sum_{i=1}^{n} \xi(T_i).$$

The correctness of MATCH is now stated in the lemma below. We omit a proof of this lemma, referring the interested reader to [15] or [25].

5.7. Lemma. *Let $\mathcal{V}$ be a set of variables, let F_1 be a flexible formula and let F_2 be a rigid formula of the same type as F_1. If there is a substitution $\sigma \in \mathcal{U}(\{\langle F_1, F_2\rangle\})$ then there is a substitution $\varphi \in \mathrm{MATCH}(F_1, F_2, \mathcal{V})$ and a corresponding substitution σ' such that*

(i) $\sigma =_{\mathcal{V}} \sigma' \circ \varphi$, *and*

(ii) $\pi(\sigma') < \pi(\sigma)$.†

A unification procedure may now be described based on an iterative use of SIMPL and MATCH. A procedure that searches for a $\mathcal{P}$-derivation, a notion that we describe next, actually embeds such a unification procedure within it.

$\mathcal{P}$-Derivations. Let the symbols $\mathcal{G}$, $\mathcal{D}$, θ and $\mathcal{V}$, perhaps with subscripts, denote sets of formulas of type o, disagreement sets, substitutions and sets of variables, respectively. The relation of being "$\mathcal{P}$-derivable from" between tuples of the form $\langle \mathcal{G}, \mathcal{D}, \theta, \mathcal{V}\rangle$ is defined in the following manner.

† This lemma may actually be strengthened: If $f \in \mathcal{V}$, then there is exactly one φ corresponding to each σ.

5.8. Definition. Let $\mathcal{P}$ be a set of definite sentences. We say a tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ is *$\mathcal{P}$-derivable* from the tuple $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ if $\mathcal{D}_1 \neq \mathbf{F}$ and, in addition, one of the following situations holds:

(1) (*Goal reduction step*) $\theta_2 = \emptyset$, $\mathcal{D}_2 = \mathcal{D}_1$, and there is a goal formula $G \in \mathcal{G}_1$ such that

   (a) G is $\top$ and $\mathcal{G}_2 = \mathcal{G}_1 - \{G\}$ and $\mathcal{V}_2 = \mathcal{V}_1$, or

   (b) G is $G_1 \wedge G_2$ and $\mathcal{G}_2 = (\mathcal{G}_1 - \{G\}) \cup \{G_1, G_2\}$ and $\mathcal{V}_2 = \mathcal{V}_1$, or

   (c) G is $G_1 \vee G_2$ and, for $i = 1$ or $i = 2$, $\mathcal{G}_2 = (\mathcal{G}_1 - \{G\}) \cup \{G_i\}$ and $\mathcal{V}_2 = \mathcal{V}_1$, or

   (d) G is $\Sigma\, P$ and for some variable $y \notin \mathcal{V}_1$ it is the case that $\mathcal{V}_2 = \mathcal{V}_1 \cup \{y\}$ and $\mathcal{G}_2 = (\mathcal{G}_1 - \{G\}) \cup \{\lambda norm(P\, y)\}$.

(2) (*Backchaining step*) Let $G \in \mathcal{G}_1$ be a rigid positive atom, and let $D \in \mathcal{P}$ be such that $D \equiv \forall x_1. \ldots \forall x_n. G' \supset A$ for some sequence of variables $x_1, \ldots, x_n$ for which no $x_i \in \mathcal{V}_1$. Then $\theta_2 = \emptyset$, $\mathcal{V}_2 = \mathcal{V}_1 \cup \{x_1, \ldots, x_n\}$, $\mathcal{G}_2 = (\mathcal{G}_1 - \{G\}) \cup \{G'\}$, and $\mathcal{D}_2 = \mathrm{SIMPL}(\mathcal{D}_1 \cup \{\langle G, A \rangle\})$.

(3) *(Unification step)* $\mathcal{D}_1$ is not a solved set and for some flexible-rigid pair $\langle F_1, F_2 \rangle \in \mathcal{D}_1$, either $\mathrm{MATCH}(F_1, F_2, \mathcal{V}_1) = \emptyset$ and $\mathcal{D}_2 = \mathbf{F}$, or there is a $\sigma \in \mathrm{MATCH}(F_1, F_2, \mathcal{V}_1)$ and it is the case that $\theta_2 = \sigma$, $\mathcal{G}_2 = \sigma(\mathcal{G}_1)$, $\mathcal{D}_2 = \mathrm{SIMPL}(\sigma(\mathcal{D}_1))$, and, if $\sigma = \{\langle x, T \rangle\}$, $\mathcal{V}_2 = \mathcal{V}_1 \cup \mathcal{F}(T)$.

Let us call a finite set of goal formulas a *goal set*, and a disagreement set that is $\mathbf{F}$ or consists solely of pairs of positive formulas a *positive disagreement set*. If $\mathcal{G}_1$ is a goal set and $\mathcal{D}_1$ is a positive disagreement set then it is clear, from an inspection of the above definition, the definitions 5.3 and 5.5, and the fact that a positive formula remains a positive formula under a positive substitution, that $\mathcal{G}_2$ is a goal set and $\mathcal{D}_2$ a positive disagreement set for any tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$.

5.9. Definition. Let $\mathcal{G}$ be a goal set. Then we say that a sequence $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle_{1 \leq i \leq n}$ is a $\mathcal{P}$-derivation sequence for $\mathcal{G}$ just in case $\mathcal{G}_1 = \mathcal{G}$, $\mathcal{V}_1 = \mathcal{F}(\mathcal{G}_1)$, $\mathcal{D}_1 = \emptyset$, $\theta_1 = \emptyset$, and, for $1 \leq i < n$, $\langle \mathcal{G}_{i+1}, \mathcal{D}_{i+1}, \theta_{i+1}, \mathcal{V}_{i+1} \rangle$ is $\mathcal{P}$-derivable from $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle$.

From our earlier observations, and an easy induction on the length of the sequence, it is clear that in a $\mathcal{P}$-derivation sequence for a goal set $\mathcal{G}$ each $\mathcal{G}_i$ is a goal set and each $\mathcal{D}_i$ is a positive disagreement set. We make implicit use of this observation in our discussions below. In particular, we intend unqualified uses of the symbols $\mathcal{G}$ and $\mathcal{D}$ to be read as syntactic variables for goal sets and positive disagreement sets, respectively.

A $\mathcal{P}$-derivation sequence $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle_{1 \leq i \leq n}$ terminates, *i.e.* is not contained in a longer sequence, if

(a) $\mathcal{G}_n$ is either empty or is a goal set consisting solely of flexible atoms and $\mathcal{D}_n$ is either empty or consists solely of flexible-flexible pairs, or

(b) $\mathcal{D}_n = \mathbf{F}$.

In the former case we say that it is a *successfully terminated* sequence.

5.10. Definition. A $\mathcal{P}$-derivation sequence, $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle_{1 \le i \le n}$, for $\mathcal{G}$ that is a successfully terminated sequence is called a *$\mathcal{P}$-derivation of $\mathcal{G}$* and $\theta_n \circ \cdots \circ \theta_1$ is called its *answer substitution.* If $\mathcal{G} = \{G\}$ then we also say that the sequence is a $\mathcal{P}$-derivation of G.

5.11. Example. Let $\mathcal{P}$ be the set of definite sentences in Example 3.3. Further, let f_1 be a variable of type $int \to int$ and let G be the goal formula

$$[mapfun\, f_1\, [cons\, 1\, [cons\, 2\, nil]]\, [cons\, [g\, 1\, 1]\, [cons\, [g\, 1\, 2]\, nil]]].$$

Then the tuple $\langle \mathcal{G}_1, \mathcal{D}_1, \emptyset, \mathcal{V}_1 \rangle$ is $\mathcal{P}$-derivable from $\langle \{G\}, \emptyset, \emptyset, \{f_1\} \rangle$ by a backchaining step, if

$$\begin{aligned}
\mathcal{V}_1 &= \{f_1, f_2, l_1, l_2, x\}, \\
\mathcal{G}_1 &= \{[mapfun\, f_2\, l_1\, l_2]\}, \quad \text{and} \\
\mathcal{D}_1 &= \{\langle f_1, f_2 \rangle, \langle x, 1 \rangle, \langle [f_1\ x], [g\ 1\ 1] \rangle, \langle l_1, [cons\ 2\ nil] \rangle, \langle l_2, [cons\ [g\ 1\ 2]\ nil] \rangle\},
\end{aligned}$$

where f_2, l_1, l_2, and x are variables. Similarly, if

$$\begin{aligned}
\mathcal{V}_2 &= \mathcal{V}_1 \cup \{h_1, h_2\}, \\
\mathcal{G}_2 &= \{[mapfun\, f_2\, l_1\, l_2]\}, \\
\theta_2 &= \{\langle f_1, \lambda w.[g\, [h_1\, w][h_2\, w]] \rangle\}, \quad \text{and} \\
\mathcal{D}_2 &= \{\langle l_1, [cons\, 2\, nil] \rangle, \langle l_2, [cons\, [g\, 1\, 2]\, nil] \rangle, \langle x, 1 \rangle, \\
&\qquad \langle [h_1\, x], 1 \rangle, \langle [h_2\, x], 1 \rangle, \langle f_2, \lambda w.[g\, [h_1\, w]\, [h_2\, w]] \rangle\},
\end{aligned}$$

then the tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \emptyset, \mathcal{V}_1 \rangle$ by a unification step. It is, in fact, obtained by picking the flexible-rigid pair $\langle [f_1\, x], [g\, 1\, 1] \rangle$ from $\mathcal{D}_1$ and using the substitution provided by IMIT for this pair. If the substitution provided by PROJ$_1$ was picked instead, we would obtain the tuple $\langle \mathcal{G}_2, \mathbf{F}, \{\langle f_1, \lambda w.w \rangle\}, \mathcal{V}_1 \rangle$.

There are several $\mathcal{P}$-derivations of G, and all of them have the same answer substitution: $\{\langle f_1, \lambda w.[g\, w\, 1] \rangle\}$.

5.12. Example. Let $\mathcal{P}$ be a set of definite sentences that contains the definite sentence $\forall x.[x \supset [P\, A]]$, where P and A are parameters of type $int \to o$ and int, respectively. Then, the following sequence of tuples constitutes a $\mathcal{P}$-derivation of $\exists y.[P\, y]$:

$$\langle \{\exists y.[P\ y]\}, \emptyset, \emptyset, \emptyset \rangle,\ \langle \{[P\ y]\}, \emptyset, \emptyset, \{y\} \rangle,\ \langle \{x\}, \{\langle y, A \rangle, \emptyset, \{y, x\} \rangle,\ \langle \{x\}, \emptyset, \{\langle y, A \rangle, \{y, x\} \rangle.$$

Notice that this is a successfully terminated sequence, even though the final goal set contains a flexible atom. We shall see, in Theorem 5.14, that a goal set that contains only flexible atoms can be "solved" rather easily. In this particular case, for instance, the final goal set may be solved by applying the substitution $\{\langle x, \top\rangle\}$ to it.

As mentioned at the beginning of this section, a $\mathcal{P}$-derivation of a goal formula G is intended to be an object that demonstrates the provability of G from the set of definite sentences $\mathcal{P}$. Our next endeavor, culminating in the Theorems 5.14 and 5.18, is to show that this notion is true to our intention. In the process, we shall see that a $\mathcal{P}$-derivation of G encodes enough information to make it possible to extract the result of a computation. We shall also observe some properties of $\mathcal{P}$-derivations that are of interest from the perspective of constructing a procedure that searches for such a derivation of a goal formula.

5.13. Lemma. *Let $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2\rangle$ be* $\mathcal{P}$-derivable *from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1\rangle$, and let $\mathcal{D}_2 \neq \mathbf{F}$. Further let $\sigma \in \mathcal{U}(\mathcal{D}_2)$ be a positive substitution such that every closed positive instance of the formulas in $\sigma(\mathcal{G}_2)$ is provable from $\mathcal{P}$. Then*

(i) $\sigma \circ \theta_2 \in \mathcal{U}(\mathcal{D}_1)$, and

(ii) every closed positive instance of the formulas in $\sigma \circ \theta_2(\mathcal{G}_1)$ is provable from $\mathcal{P}$.

Proof. The lemma is proved by considering the cases in Definition 5.9.

A goal reduction or a backchaining step. In these cases $\theta_2 = \emptyset$ and so $\sigma \circ \theta_2 = \sigma$. Further, in a goal reduction step $\mathcal{D}_2 = \mathcal{D}_1$, and in a backchaining step $\mathcal{D}_1 \subseteq \mathcal{D}_2$. From these observations and the assumptions in the lemma, it follows that (i) is true. Similarly, it is clear that all the closed positive instances of $\sigma \circ \theta_2(G)$ of each $G \in \mathcal{G}_1$ that is also an element of $\mathcal{G}_2$ are provable from $\mathcal{P}$. To verify (ii), therefore, we only need to establish the truth of the previous statement for the case when the $G \in \mathcal{G}_1$ is not in $\mathcal{G}_2$, and we do this below by considering the possibilities for such a G.

In the case that G is $\top$, the argument is obvious. If G is $G_1 \vee G_2$, then

$$\sigma \circ \theta_2(G) = \sigma \circ \theta_2(G_1) \vee \sigma \circ \theta_2(G_2) = \sigma(G_1) \vee \sigma(G_2).$$

Thus the closed positive instances of $\sigma \circ \theta_2(G)$ are of the form $G' \vee G''$ where G' and G'' are closed positive instances of $\sigma(G_1)$ and $\sigma(G_2)$ respectively. Noting that either $\sigma(G_1)$ or $\sigma(G_2)$ is an element of $\sigma(\mathcal{G}_2)$, the argument may be completed using the assumptions in the lemma and Theorem 4.14. A similar argument may be provided for the case when G is $G_1 \wedge G_2$.

Before considering the remaining cases, we define a substitution that is parameterized by a substitution and a sequence of variables. Let c_α be an arbitrary parameter of type α. If $\bar{y}$ is a sequence of variables and δ is a substitution, then

$$\delta_{\bar{y}} = \{\langle x, c_\alpha\rangle \mid x \text{ is a variable of type } \alpha \text{ such that } x \in \mathcal{F}(\delta \circ \sigma(y_i)) \text{ for some } y_i \text{ in } \bar{y}\}.$$

We note that if F is a positive formula all of whose free variables are included in the list $\bar{y}$ and if δ is a positive substitution, then $\delta_{\bar{y}} \circ \delta \circ \sigma(F)$ is a closed positive formula.

Now let the G under consideration be $\Sigma\, P$. Any closed positive instance of $\sigma \circ \theta_2(G)$ may be written in the form $\Sigma\,(\delta \circ \sigma(P))$ for a suitable δ; note that then $\delta \circ \sigma(P)$ must itself be a closed positive formula. From Definition 5.9, we see that, for some y, $\lambda norm(P\,y) \in \mathcal{G}_2$ and hence $\sigma(P\,y) \in \sigma(\mathcal{G}_2)$. It is easily seen that $\delta_y \circ \delta \circ \sigma(P\,y)$ is a closed positive instance of $\sigma(P\,y)$ and is, therefore, provable from $\mathcal{P}$. But now we observe that

$$\delta_y \circ \delta \circ \sigma(P\,y) = \rho([\delta \circ \sigma(P)\ \delta_y \circ \sigma \circ \delta(y)]).$$

Using Theorem 4.14 it then follows that $\Sigma\,(\delta \circ \sigma(P))$, is provable from $\mathcal{P}$. The choice of δ having been arbitrary, we may thus conclude that any closed positive instance of $\sigma \circ \theta_2(G)$ is provable from $\mathcal{P}$.

The only other case to consider is when G is removed by a backchaining step. In this case, by Definition 5.9 and Lemma 5.4, there must be a $D \in \mathcal{P}$ such that

$$D \equiv \forall \bar{x}.G' \supset A, \quad G' \in \mathcal{G}_2, \quad \text{and} \quad \sigma(G) = \sigma(A).$$

Once again, we observe that any closed positive instance of $\sigma \circ \theta_2(G)$ may be written as $\delta \circ \sigma(G)$ for a suitably chosen δ. Now $\delta_{\bar{x}} \circ \delta \circ \sigma(G' \supset A)$ is a closed positive instance of $G' \supset A$, and hence is a member of $|\mathcal{P}|$. Further,

$$\delta_{\bar{x}} \circ \delta \circ \sigma(G' \supset A) = \delta_{\bar{x}} \circ \delta \circ \sigma(G') \supset \delta_{\bar{x}} \circ \delta \circ \sigma(A) = \delta_{\bar{x}} \circ \delta \circ \sigma(G') \supset \delta \circ \sigma(G).$$

Finally, $\delta_{\bar{x}} \circ \delta \circ \sigma(G')$ is evidently a closed positive instance of $\sigma(G')$,and is therefore provable from $\mathcal{P}$. Using these facts in conjunction with Theorem 4.14 we may now conclude that $\delta \circ \sigma(G)$ is provable from $\mathcal{P}$.

A unification step. We note first that $\mathcal{D}_2 \neq \mathbf{F}$. Hence, in either of these cases, it follows from Lemma 5.4 that if $\sigma \in \mathcal{U}(\mathcal{D}_2)$ then $\sigma \in \mathcal{U}(\theta_2(\mathcal{D}_1))$. But then, it is easy to see that $\sigma \circ \theta_2 \in \mathcal{U}(\mathcal{D}_1)$. Since $\mathcal{G}_2 = \theta_2(\mathcal{G}_1)$ it is evident that every closed instance of a goal formula in $\sigma \circ \theta_2(\mathcal{G}_1)$ is also a closed instance of a goal formula in $\sigma(\mathcal{G}_2)$. From this the second part of the lemma is obvious. ∎

5.14. Theorem. (Soundness of $\mathcal{P}$-derivations) *Let $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle_{1 \le i \le n}$ be a* $\mathcal{P}$-derivation *of G, and let θ be its answer substitution. Then there is a positive substitution σ such that*

(i) $\sigma \in \mathcal{U}(\mathcal{D}_n)$, and

(ii) all the closed positive instances of the goal formulas in $\sigma(\mathcal{G}_n)$ are provable from $\mathcal{P}$.

Further, if σ is a positive substitution satisfying (i) and (ii), then every closed positive instance of $\sigma \circ \theta(G)$ is provable from $\mathcal{P}$.

Proof. The second part of the theorem follows easily from Lemma 5.13 and a backward induction on i, the index of each tuple in the given $\mathcal{P}$-derivation sequence. For the first part we exhibit a substitution — that is a simple modification of the one in Lemma 3.5 in [15] — and then show that it satisfies the requirements.

Let $h_\alpha \in \mathcal{V}ar_\alpha$ be a chosen variable for each atomic type α. Then for each type α we identify a formula $\hat{E}_\alpha$ in the following fashion:

(a) If α is o, then $\hat{E}_\alpha = \top$.

(b) If α is an atomic type other than o, then $\hat{E}_\alpha = h_\alpha$.

(c) If α is the function type $\beta_1 \to \cdots \to \beta_k \to \beta$ where β is an atomic type, then $\hat{E}_\alpha = \lambda x_1.\ \ldots \lambda x_k.\hat{E}_\beta$, where, for $1 \le i \le k$, x_i is a variable of type β_i that is distinct from h_{β_i}.

Now let $\gamma = \{\langle y, \hat{E}_\alpha\rangle \mid y \in \mathcal{V}ar_\alpha\}$. Finally, letting $\mathcal{V} = \mathcal{F}(\mathcal{G}_n) \cup \mathcal{F}(\mathcal{D}_n)$, we define $\sigma = \gamma \uparrow \mathcal{V}$.

We note that any goal formula in $\mathcal{G}_n$ is of the form $[P\, C_1\, \ldots\, C_n]$ where P is a variable whose type is of the form $\alpha_1 \to \cdots \to \alpha_n \to o$. From this it is apparent that if $G \in \mathcal{G}_n$ then any ground instance of $\sigma(G)$ is identical to $\top$. Thus, it is clear that σ satisfies (ii). If $\mathcal{D}_n$ is empty then $\sigma \in \mathcal{U}(\mathcal{D}_n)$. Otherwise, let $\langle F_1, F_2\rangle \in \mathcal{D}_n$. Since F_1 and F_2 are two flexible formulas, it may be seen that $\sigma(F_1)$ and $\sigma(F_2)$ are of the form $\lambda y_1^1. \ldots \lambda y_{m_1}^1.\hat{E}_{\beta_1}$, and $\lambda y_1^2. \ldots \lambda y_{m_2}^2.\hat{E}_{\beta_2}$ respectively, where β_i is a primitive type and $\hat{E}_{\beta_i} \notin \{y_1^i, \ldots, y_{m_i}^i\}$ for $i = 1, 2$. Since F_1 and F_2 have the same types and substitution is a type preserving mapping, it is clear that $\beta_1 = \beta_2$, $m_1 = m_2$ and, for $1 \le i \le m_1$, y_i^1 and y_i^2 are variables of the same type. But then evidently $\sigma(F_1) = \sigma(F_2)$. ∎

In order to show a converse of the above theorem, we need the observation contained in the following lemma that may be verified by a routine inspection of Definition 5.9.

5.15. Lemma. *Let $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2\rangle$ be $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1\rangle$ and let $\mathcal{D} \neq \mathbf{F}$. Then $\mathcal{V}_1 \subseteq \mathcal{V}_2$ and if $\mathcal{F}(\mathcal{G}_1) \cup \mathcal{F}(\mathcal{D}_1) \subseteq \mathcal{V}_1$, then $\mathcal{F}(\mathcal{G}_2) \cup \mathcal{F}(\mathcal{D}_2) \subseteq \mathcal{V}_2$.*

We also need a measure of complexity corresponding to a goal set and a unifier. In defining such a measure, we use those introduced in Definitions 4.15 and 5.6.

5.16. Definition.

(i) Let $\mathcal{G}$ be a set of closed goal formulas. Then $\nu_{\mathcal{P}}(\mathcal{G}) = \sum_{G \in \mathcal{G}} \mu_{\mathcal{P}}(G)$.

(ii) Let $\mathcal{G}$ be a set of goal formulas and let σ be a positive substitution such that each formula in $\sigma(\mathcal{G})$ is closed. Then $\kappa_{\mathcal{P}}(\mathcal{G}, \sigma) = \langle \nu_{\mathcal{P}}(\sigma(\mathcal{G})), \pi(\sigma)\rangle$.

(iii) $\prec$ is the lexicographic ordering on the collection of pairs of natural numbers, *i.e.* $\langle m_1, n_1\rangle \prec \langle m_2, n_2\rangle$ if either $m_1 < m_2$ or $m_1 = m_2$ and $n_1 < n_2$.

If $\mathcal{G}$ is a finite set of closed goal formulas such that each member of $\mathcal{G}$ is provable from

$\mathcal{P}$, then it is easily seen that $\nu_{\mathcal{P}}(\mathcal{G}) < \omega$. We make implicit use of this fact in the proof of the following lemma.

5.17. Lemma. *Let $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ be a tuple that is not a terminated $\mathcal{P}$-*derivation *sequence and for which $\mathcal{F}(\mathcal{G}_1) \cup \mathcal{F}(\mathcal{D}_1) \subseteq \mathcal{V}_1$. Further, let there be a positive substitution $\sigma_1 \in \mathcal{U}(\mathcal{D}_1)$ such that, for each $G_1 \in \mathcal{G}_1$, $\sigma_1(G_1)$ is a closed goal formula that is provable from $\mathcal{P}$. Then there is a tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ that is $\mathcal{P}$-*derivable *from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ and a positive substitution σ_2 such that*

(i) $\sigma_2 \in \mathcal{U}(\mathcal{D}_2)$,

(ii) $\sigma_1 =_{\mathcal{V}_1} \sigma_2 \circ \theta_2$,

(iii) for each $G_2 \in \mathcal{G}_2$, $\sigma_2(G_2)$ is a closed goal formula that is provable from $\mathcal{P}$, and

(iv) $\kappa_{\mathcal{P}}(\mathcal{G}_2, \sigma_2) \prec \kappa_{\mathcal{P}}(\mathcal{G}_1, \sigma_1)$.

*In addition, when there are several tuples that are $\mathcal{P}$-*derivable *from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$, such a tuple and such a substitution exist for every choice of (1) the kind of step, (2) the goal formula in a goal reduction or backchaining step, and (3) the flexible-rigid pair in a unification step.*

Proof. Since $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ is not a terminated $\mathcal{P}$-derivation sequence, it is clear that there must be a tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ that is $\mathcal{P}$-derivable from it. We consider below the various ways in which such a tuple may result to show that there must exist a tuple, and a corresponding substitution, satisfying the requirements of the lemma. From this argument it will also be evident that this is the case no matter how the choices mentioned in the lemma are exercised.

Goal reduction step. If there is tuple that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ by such a step, then there must be a goal formula in $\mathcal{G}_1$ of the form $\top$, $G_1 \vee G_2$, $G_1 \wedge G_2$, or $\Sigma\, P$. Let us consider the first three cases first. In these cases, we let

$$\mathcal{D}_2 = \mathcal{D}_1, \quad \theta_2 = \emptyset, \quad \mathcal{V}_2 = \mathcal{V}_1, \quad \text{and} \quad \sigma_2 = \sigma_1.$$

Since $\sigma_1 \in \mathcal{U}(\mathcal{D}_1)$, it is obvious that $\sigma_2 \in \mathcal{U}(\mathcal{D}_2)$. Further, $\sigma_1 =_{\mathcal{V}_1} \sigma_2 \circ \theta_2$; in fact, $\sigma_1 = \sigma_2 \circ \theta_2$. Now we consider each of the cases in turn to provide a value for $\mathcal{G}_2$ that, together with the assignments provided above, meets the requirements of the lemma.

(a) If $\top \in \mathcal{G}_1$, then let $\mathcal{G}_2 = \mathcal{G}_1 - \{\top\}$. $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ is obviously a tuple that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$. The observations above show that this tuple and σ_2 satisfy conditions (i) and (ii) in the lemma. That (iii) is true follows from the facts that $\mathcal{G}_2 \subseteq \mathcal{G}_1$ and $\sigma_2 = \sigma_1$. Observing additionally that $\mu_{\mathcal{P}}(\top) > 0$, (iv) follows.

(b) Let $G_1 \vee G_2 \in \mathcal{G}_1$. We note here that

$$\sigma_1(G_1 \vee G_2) = \sigma_1(G_1) \vee \sigma_1(G_2) = \sigma_2(G_1) \vee \sigma_2(G_2),$$

and, further, that $\sigma_2(G_1)$ and $\sigma_2(G_2)$ are closed goal formulas. Using Theorem 4.14 and the assumption that $\sigma_1(G_1 \vee G_2)$ is provable from $\mathcal{P}$, we see that $\sigma_2(G_i)$ is provable from $\mathcal{P}$ for $i = 1$ or $i = 2$. Further, by Lemma 4.16, it is the case that for the same i

$$\mu_{\mathcal{P}}(\sigma_1(G_i)) < \mu_{\mathcal{P}}(\sigma_1(G_1 \vee G_2)).$$

Setting

$$\mathcal{G}_2 = (\mathcal{G}_1 - \{G_1 \vee G_2\}) \cup \{G_i\},$$

we obtain a tuple that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ and that together with σ_2 satisfies the requirements in the lemma.

(c) If $G_1 \wedge G_2 \in \mathcal{G}_1$, let $\mathcal{G}_2 = (\mathcal{G}_1 - \{G_1 \wedge G_2\}) \cup \{G_1, G_2\}$. By arguments similar to those in (b), it follows that $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$ and σ_2 meet the requirements in the lemma.

The only case remaining is when $\Sigma\, P \in \mathcal{G}_1$. Here we choose a variable y such that $y \notin \mathcal{V}_1$, and let

$$\mathcal{G}_2 = (\mathcal{G}_1 - \{\Sigma\, P\}) \cup \{\lambda norm(P\, y)\},\ \mathcal{D}_2 = \mathcal{D}_1,\ \theta_2 = \emptyset, \text{and } \mathcal{V}_2 = \mathcal{V}_1 \cup \{y\}.$$

Evidently $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ is a tuple that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1 \rangle$. Now if we let $P' = \sigma_1(P)$, we see that $\Sigma\, P' \in \sigma_1(\mathcal{G}_1)$. Thus, by assumption, P' is a closed positive formula and $\Sigma\, P'$ is provable from $\mathcal{P}$. From Theorem 4.14 and Lemma 4.16 it follows that there is a closed positive formula C such that $\rho(P'\, C)$ is provable from $\mathcal{P}$ and, in fact, $\mu_{\mathcal{P}}(\rho(P'\, C)) < \mu_{\mathcal{P}}(\Sigma\, P')$. Setting

$$\sigma_2 = \sigma_1 \circ \{\langle y, C \rangle\},$$

we see that $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2 \rangle$ and σ_2 meet the requirements in the lemma: Since $y \notin \mathcal{V}_1$ and $\mathcal{F}(\mathcal{D}_1) \subseteq \mathcal{V}_1$, $\sigma_2 \in \mathcal{U}(\mathcal{D}_2)$ and $\sigma_1 =_{\mathcal{V}_1} \sigma_2$. Since $\mathcal{F}(\mathcal{G}_1) \subseteq \mathcal{V}_1$, (iii) is satisfied for each $G \in \mathcal{G}_2$ that is also in $\mathcal{G}_1$. For the only other $G \in \mathcal{G}_2$, *i.e.* $\lambda norm(P\, y)$, it is apparent that $\sigma_2(G) = \rho(P'\, C)$ and so (iii) is satisfied in this case too. From these observations and the fact that $\mu_{\mathcal{P}}(\rho(P'\, C)) < \mu_{\mathcal{P}}(\sigma_1(\Sigma\, P))$, (iv) also follows.

Backchaining step. For this step to be applicable, there must be a rigid positive atom $G \in \mathcal{G}_1$. Let $G_a = \sigma_1(G)$. By assumption, G_a is a closed positive atom that is provable from $\mathcal{P}$. Therefore, by Theorem 4.14, there must be a formula $G'' \supset G_a \in |\mathcal{P}|$ such that G'' is provable from $\mathcal{P}$; in fact, by Lemma 4.16, $\mu_{\mathcal{P}}(G'') < \mu_{\mathcal{P}}(G_a)$. Since $G'' \supset G_a \in |\mathcal{P}|$, there must be a $D \in \mathcal{P}$ such that

$$D \equiv \forall x_1 \ldots \forall x_n . G' \supset A,$$

where $x_1, \ldots, x_n$ are not members of $\mathcal{V}_1$, and a positive substitution φ for $\{x_1, \ldots, x_n\}$ such that $G_a = \varphi(A)$ and $G'' = \varphi(G')$. Now, setting

$$\mathcal{G}_2 = (\mathcal{G}_1 - \{G\}) \cup \{G'\}, \qquad \theta_2 = \emptyset,$$

$$\mathcal{D}_2 = \mathrm{SIMPL}(\mathcal{D}_1 \cup (\{\langle G, A\rangle\}), \qquad \mathcal{V}_2 = \mathcal{V}_1 \cup \{x_1, \ldots, x_n\},$$

we obtain the tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2\rangle$ that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1\rangle$. Letting $\sigma_2 = \sigma_1 \circ \varphi$, we see that this tuple and σ_2 also meet the requirements in the lemma: Since $x_i \notin \mathcal{V}_1$ for $1 \leq i \leq n$,

$$\sigma_1 =_{\mathcal{V}_1} \sigma_2 =_{\mathcal{V}_1} \sigma_2 \circ \theta_2.$$

Thus (ii) is satisfied. Also, since $\mathcal{F}(\mathcal{D}_1) \subseteq \mathcal{V}_1$, $\sigma_2 \in \mathcal{U}(\mathcal{D}_1)$. Noting that $\varphi(A)$ is a closed formula and that $\mathcal{F}(G) \subseteq \mathcal{V}_1$,

$$\sigma_2(G) = \sigma_1(G) = \varphi(A) = \sigma_2(A).$$

From these observations and Lemma 5.4, it is clear that $\sigma_2 \in \mathcal{U}(\mathcal{D}_2)$, *i.e.* (i) is satisfied. Now, since $\mathcal{F}(\mathcal{G}_1) \subseteq \mathcal{V}_1$, $\sigma_2(G_1) = \sigma_1(G_1)$ for each $G_1 \in \mathcal{G}_2$ that is also in $\mathcal{G}_1$. Thus (iii) is true by assumption for such a G_1. For the only other formula in $\mathcal{G}_2$, *i.e.* G', this follows by observing that

$$\sigma_2(G') = \varphi(G') = G'';$$

G'' is by assumption provable from $\mathcal{P}$. Finally, (iv) follows by observing that

$$\mu_{\mathcal{P}}(\sigma_2(G')) = \mu_{\mathcal{P}}(G'') < \mu_{\mathcal{P}}(G_a) = \mu_{\mathcal{P}}(\sigma_1(G)).$$

Unification step. For this case to be applicable, there must be a flexible-rigid pair in $\mathcal{D}_1$. Let $\langle F_1, F_2\rangle$ be an arbitrary such pair. By Lemma 5.7, there is a (positive) substitution $\varphi \in \mathrm{MATCH}(F_1, F_2, \mathcal{V}_1)$ and a (positive) substitution δ such that $\sigma_1 =_{\mathcal{V}_1} \delta \circ \varphi$ and $\pi(\delta) < \pi(\sigma_1)$. Setting

$$\mathcal{G}_2 = \varphi(\mathcal{G}_1), \quad \mathcal{D}_2 = \mathrm{SIMPL}(\varphi(\mathcal{D}_1)), \quad \theta_2 = \varphi,$$

and choosing $\mathcal{V}_2$ appropriately, we see that there is a tuple $\langle \mathcal{G}_2, \mathcal{D}_2, \theta_2, \mathcal{V}_2\rangle$ that is $\mathcal{P}$-derivable from $\langle \mathcal{G}_1, \mathcal{D}_1, \theta_1, \mathcal{V}_1\rangle$. Letting $\sigma_2 = \delta$ we see easily that the other requirements of the lemma are also satisfied: Since $\mathcal{F}(\mathcal{D}_1) \subseteq \mathcal{V}_1$, it is clear that

$$\sigma_1(\mathcal{D}_1) = \sigma_2 \circ \theta_2(\mathcal{D}_1) = \sigma_2(\theta_2(\mathcal{D}_1)).$$

Noting that $\sigma_1 \in \mathcal{U}(\mathcal{D}_1)$, (i) follows from Lemma 5.4. (ii) is evidently true. Since $\mathcal{F}(\mathcal{G}_1) \subseteq \mathcal{V}_1$, we see that

$$\sigma_1(\mathcal{G}_1) = \sigma_2 \circ \theta_2(\mathcal{G}_1) = \sigma_2(\theta_2(\mathcal{G}_1)) = \sigma_2(\mathcal{G}_2).$$

That every $G_2 \in \sigma_2(\mathcal{G}_2)$ is a closed goal formula that is provable from $\mathcal{P}$ now follows trivially from the assumptions. Finally

$$\kappa_{\mathcal{P}}(\mathcal{G}_2, \sigma_2) \prec \kappa_{\mathcal{P}}(\mathcal{G}_1, \sigma_1)$$

since $\nu_{\mathcal{P}}(\sigma_2(\mathcal{G}_2)) = \nu_{\mathcal{P}}(\sigma_1(\mathcal{G}_1))$ and $\pi(\sigma_2) < \pi(\sigma_1)$. ▮

5.18. Theorem. (Completeness of $\mathcal{P}$-derivations) *Let φ be a closed positive substitution for the free variables of G such that $\varphi(G)$ is provable from $\mathcal{P}$. Then there is a $\mathcal{P}$-*derivation *of G with an answer substitution θ such that $\varphi \preceq_{\mathcal{F}(G)} \theta$.*

Proof. From Lemmas 5.17 and 5.15 and the assumption of the theorem, it is evident that there is a $\mathcal{P}$-derivation sequence $\langle \mathcal{G}_i, \mathcal{D}_i, \theta_i, \mathcal{V}_i \rangle_{1 \le i}$ for $\{G\}$ and a sequence of substitutions σ_i such that

(i) $\sigma_1 = \varphi$,

(ii) σ_{i+1} satisfies the equation $\sigma_i =_{\mathcal{V}_i} \sigma_{i+1} \circ \theta_{i+1}$,

(iii) $\sigma_i \in \mathcal{U}(\mathcal{D}_i)$, and

(iv) $\kappa_{\mathcal{P}}(\mathcal{G}_{i+1}, \sigma_{i+1}) \prec \kappa_{\mathcal{P}}(\mathcal{G}_i, \sigma_i)$.

From (iv) and the definition of $\prec$ it is clear that the sequence must terminate. From (iii) and Lemmas 5.4 and 5.7 it is evident, then, that it must be a successfully terminated sequence, *i.e.* a $\mathcal{P}$-derivation of G. Using (i), (ii) and Lemma 5.15, an induction on the length n of the sequence then reveals that $\varphi \preceq_{\mathcal{V}_1} \theta_n \circ \cdots \circ \theta_1$. But $\mathcal{F}(G) = \mathcal{V}_1$ and $\theta_n \circ \cdots \circ \theta_1$ is the answer substitution for the sequence. ▮

$\mathcal{P}$-derivations, thus, provide the basis for describing the proof procedure that we desired at the outset. Given a goal formula G, such a procedure starts with the tuple $\langle \{G\}, \emptyset, \emptyset, \mathcal{F}(G) \rangle$ and constructs a $\mathcal{P}$-derivation sequence. If the procedure performs an exhaustive search, and if there is a proof of G from $\mathcal{P}$, it will always succeed in constructing a $\mathcal{P}$-derivation of G from which a result may be extracted. A breadth-first search may be inappropriate if the procedure is intended as an interpreter for a programming language based on our definite sentences. By virtue of Lemma 5.17, we see that there are certain cases in which the procedure may limit its choices without adverse effects. The following choices are, however, critical:

(i) Choice of disjunct in a goal reduction step involving a disjunctive goal,

(ii) Choice of definite sentence in a backchaining step, and

(iii) Choice of substitution in a unification step.

When it encounters such choices, the procedure may, with an accompanying loss of completeness, perform a depth-first search with backtracking. The particular manner in which to exercise these choices is very much an empirical question, a question to be settled only by experimentation.

6. Conclusion

In this paper we have concerned ourselves with the provision of higher-order features within logic programming. An approach that has been espoused elsewhere in this regard is to leave the basis of first-order logic programming languages unchanged and to provide some of the functionality of higher-order features through special mechanisms built into the interpreter. This approach is exemplified by the presence of "extra-logical" predicates such as *univ* and *functor* in most current implementations of Prolog [32]. While this approach has the advantage that usable "higher-order" extensions may be provided rapidly, it has the drawbacks that the logical basis of the resulting language is no longer clear and, further, that the true nature and utility of higher-order features within logic programming is obscured.

We have explored an alternative approach based on strengthening the underlying logic. In a precise sense, we have abstracted out those properties of first-order Horn clauses that appear to be essential to their computational interpretation, and have described a class of higher-order formulas that retain these properties. Towards realizing a higher-order logic programming language based on these formulas, we have also discussed the structure of a theorem-proving procedure for them. These results have been used elsewhere [21, 25] in the description of a language called λProlog. Although space does not permit a detailed discussion of this language, it needs to be mentioned that an experimental implementation for it exists, and has in fact been widely distributed. This implementation has, among other things, provided us with insights into the practical aspects of the trade-offs to be made in designing an actual theorem-proving procedure based on the discussions in Section 5 [25]. Its existence has also stimulated research into applications of the truly novel feature of the extension discussed in this paper: the use of λ-terms as data structures in a logic programming language.

This work has suggested several questions of both a theoretical and a practical nature, some of which are currently being examined. One theoretical question that has been addressed is that of providing for a stronger use of logical connectives within logic programming. Our approach in this regard has been to understand the desired "search" semantics for each logical connective and to then identify classes of formulas within appropriately chosen proofs systems that permit a match between the declarative and search-related meanings for the connectives. One extension along these lines to the classical theory of Horn clauses is provided by the intuitionistic theory of *hereditary Harrop formulas.* The first-order version of these formulas is presented in [18, 19] and the higher-order version is discussed in detail in [23]. These formulas result from allowing certain occurrences of universal quantifiers and implications into goals and program clauses and provide the means for realizing new notions of abstractions within logic programming. The higher-order version of hereditary Harrop formulas has been incorporated into the current version of λProlog [27] and has provided significant enrichments to it as a programming language.

A second theoretical issue is the provision of a richer term language within λProlog. The use of simply typed λ-terms has turned out to be a limiting factor in the programming context, and we have therefore incorporated a form of polymorphism inspired by ML [11, 24]. A complete theoretical analysis for this extension is, however, yet to be provided. Further, there is reason to believe that a term language that permits an explicit quantification over types, *e.g.* the one discussed in [8], may be a better choice in this context. In a similar vein, a richer term language like the one provided in [30], may also be considered as the basis for the data structures of λProlog.

Among the practical questions, an important one that is being addressed is the description of an efficient implementation for a λProlog-like language [26]. The key pursuit in this respect is to devise data structures for λ-terms that will support reasonable implementations of the reduction mechanism of functional programming on the one hand, and of the unification and backchaining mechanisms of logic programming on the other. Another issue of interest is the harnessing of the richness added to the logic programming paradigm by the use of λ-terms as data structures. As already indicated, ongoing research has been focused on exploiting such a language in areas that include theorem proving, type inference, program transformation, and computational linguisitics.

Acknowledgements. This paper is based on Nadathur's doctoral dissertation. The authors gratefully acknowledge helpful comments provided on this work by Jean Gallier, Andre Scedrov and Richard Statman.

References

[1] Andrews, P. B. Resolution in type theory. *Journal of Symbolic Logic* 36 (1971) 414 – 432.

[2] Apt, K. R., and van Emden, M. H. Contributions to the theory of logic programming. *J. ACM* 29, 3 (1982) 841 – 862.

[3] Barendregt, H.P. *The Lambda Calculus: Its Syntax and Semantics*, North Holland Publishing Co., 1981.

[4] Bowen, K. A. Programming with full first-order logic. *Machine Intelligence 10*, Hayes, J. E., Michie, D. and Pao, Y-H, eds., Halsted Press, 1982, 421 – 440.

[5] Church, A. A formulation of the simple theory of types. *Journal of Symbolic Logic* 5 (1940) 56 – 68.

[6] Constable, R. L. *et. al. Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, 1986.

[7] Felty, A., and Miller, D. Specifying theorem provers in a higher-order logic programming language. Proceedings of the 9th International Conference on Automated Deduction, 1988, Springer-Verlag, 61 – 80.

[8] Fortune, S., Leivant, D., and O'Donnell, M. The expressiveness of simple and second-order type structures. *J. ACM* 30, 1 (1983) 151 – 185.

[9] Gentzen, G. Investigations into logical deduction. *The Collected Papers of Gerhard Gentzen*, Szabo, M. E., ed., North-Holland Publishing Co., 1969, 68 – 131.

[10] Goldfarb, W. D. The undecidability of the second-order unification problem. *Theoretical Computer Science* 13 (1981) 225 – 230.

[11] Gordon, M., Milner, A., and Wadsworth, C. *Edinburgh LCF: A Mechanized Logic of Computation, LNCS 78*, Springer-Verlag, 1972.

[12] Gould, W. E. A matching procedure for ω-order logic. Scientific Report No. 4, A F C R L (1976) 66 – 781.

[13] Hannan, J. and Miller, D. Uses of higher-order unification for implementing program transformers. Proceedings of the Fifth International Conference and Symposium on Logic Programming, 1988, MIT Press, 942 – 959.

[14] Huet, G. P. The undecidability of unification in third order logic. *Information and Control* 22, 3 (1973) 257 – 267.

[15] Huet, G. P. A unification algorithm for typed λ-calculus. *Theoretical Computer Science* 1 (1975) 27 – 57.

[16] Huet, G. and Lang, B. Proving and Applying Program Transformations Expressed with Second-Order Logic. Acta Informatica 11 (1978) 31–55.

[17] Lucchesi, C. L. The undecidability of the unification problem for third order languages. Report C S R R 2059, Dept. of Applied Analysis and Computer Science, University of Waterloo, 1972.

[18] Miller, D. Hereditary Harrop formulas and logic programming. Proceedings of the VIII International Congress of Logic, Methodology, and Philosophy of Science, Moscow, August 1987.

[19] Miller, D. A logical analysis of modules in logic programming. *Journal of Logic Programming* 6 (1989) 79 – 108.

[20] Miller, D., and Nadathur, G. Some uses of higher-order logic in computational linguistics. Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 1986, 247 – 255.

[21] Miller, D. and Nadathur, G. Higher-order logic programming. Proceedings of the Third International Logic Programming Conference, London, 1986, *LNCS 225*, Springer Verlag, 448 – 462.

[22] Miller, D., and Nadathur, G. A logic programming approach to manipulating formulas and programs. IEEE Symposium on Logic Programming, 1987, 379 – 388.

[23] Miller, D., Nadathur, G., Pfenning, F. and Scedrov, A. Uniform proofs as a foundation for logic programming. To appear in the *Annals of Pure and Applied Logic.*

[24] Milner, R. A theory of type polymorphism in programming. *Journal of Computer and System Sciences* 17 (1978) 348 – 375.

[25] Nadathur, G. A higher-order logic as the basis for logic programming. Ph.D. Dissertation, University of Pennsylvania, May 1987.

[26] Nadathur, G. and Jayaraman, B. Towards a WAM model for λProlog. To appear in the Proceedings of the North American Conference on Logic Programming, MIT Press, 1989.

[27] Nadathur, G. and Miller, D. An overview of λProlog. Proceedings of the Fifth International Conference on Logic Programming, 1988, MIT Press, 810 – 827.

[28] Paulson, L. C. Natural deduction as higher-order resolution. *The Journal of Logic Programming* 3, 3 (1986) 237 – 258.

[29] Pfenning, F. Partial polymorphic type inference and higher-order unification. Proceedings of the 1988 ACM Conference on Lisp and Functional Programming, 1988, 153 – 163.

[30] Pfenning, F. and Elliot, C. Higher-order abstract syntax. Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation, 1988, 199 – 208.

[31] Scherlis, W. and Scott, D. First steps towards inferential programming. *Information Processing 83*, North-Holland, Amsterdam, 1983.

[32] Sterling, L., and Shapiro, E. *The Art of Prolog: Advanced Programming Techniques.* MIT Press, Cambridge MA, 1986.

[33] van Emden, M. H., and Kowalski, R. A. The semantics of predicate logic as a programming language. *J. ACM* 23, 4 (1976) 733 – 742.

[34] Warren, D. H. D. Higher-order extensions to Prolog: Are they needed? *Machine Intelligence 10*, Hayes, J. E., Michie, D. and Pao, Y-H, eds., Halsted Press, 1982, 441 – 454.