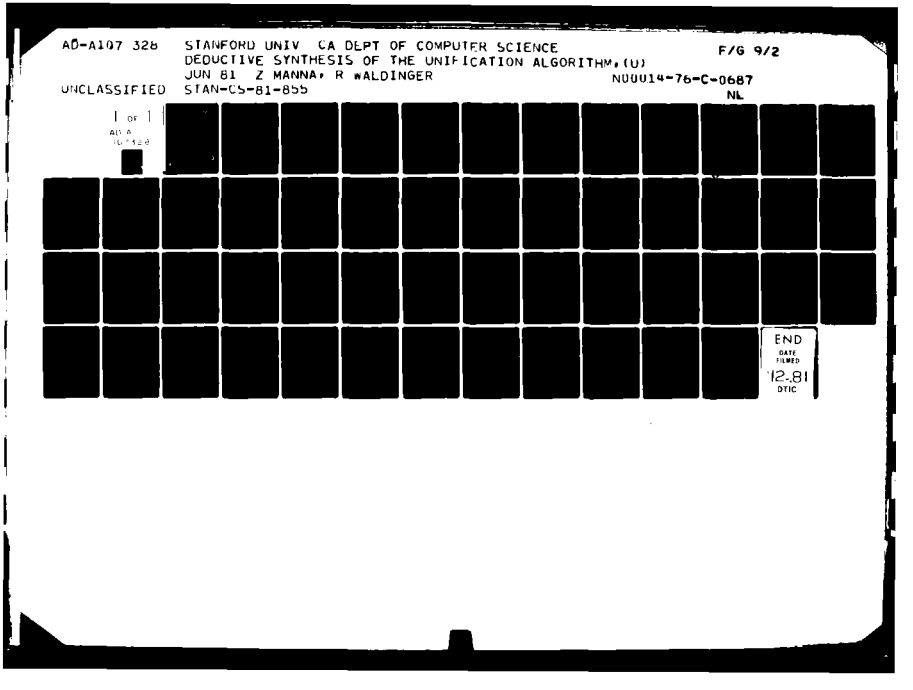
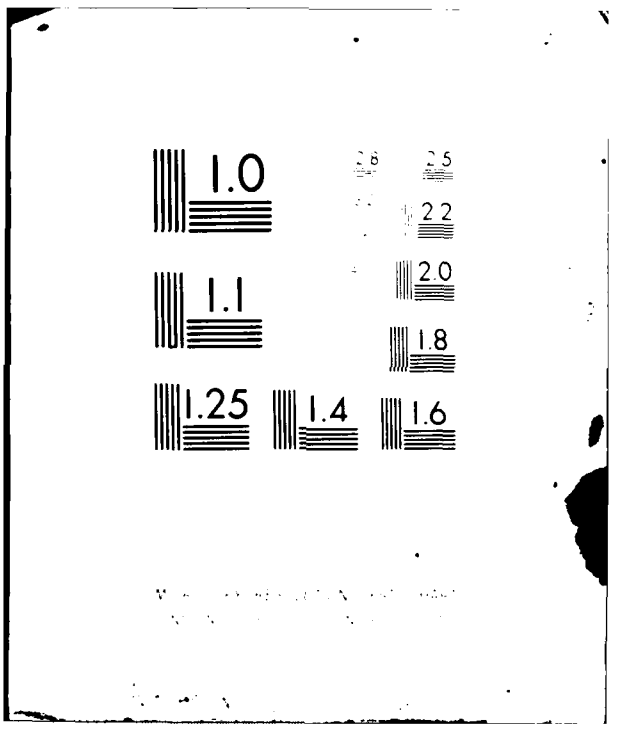

AD-A107 328
STANFORD UNIV CA DEPT OF COMPUTER SCIENCE
F/G 9/2
DEDUCTIVE SYNTHESIS OF THE UNIFICATION ALGORITHM,(U)
JUN 81 Z MANNA, R WALDINGER
N00014-76-C-0687
UNCLASSIFIED
STAN-CS-81-855
NL
1 OF 1
AD A
107328
END
DATE FILMED
12-81
DTIC


1.0
2.8
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

June 1981

LEVEL II

Report. No. STAN-CS-81-855

7

Deductive Synthesis of the Unification Algorithm

by

Zohar Manna
Richard Waldinger

Research sponsored in part by

Office of Naval Research
National Science Foundation
Air Force Office of Scientific Research

Department of Computer Science

Stanford University
Stanford, CA 94305

DEDUCTIVE SYNTHESIS OF THE UNIFICATION ALGORITHM

Zohar Manna
Computer Science Department
Stanford University
and
Applied Mathematics Department
Weizmann Institute

Richard Waldinger
Artificial Intelligence Center
SRI International

ABSTRACT

The *deductive approach* is a formal program-construction method in which the derivation of a program from a given specification is regarded as a theorem-proving task. To construct a program whose output satisfies the conditions of the specification, we prove a theorem stating the existence of such an output. The proof is restricted to be sufficiently constructive so that a program computing the desired output can be extracted directly from the proof. The program we obtain is applicative and may consist of several mutually recursive procedures. The proof constitutes a demonstration of the correctness of this program.

To exhibit the full power of the deductive approach, we apply it to a nontrivial example — the synthesis of a *unification algorithm*. Unification is the process of finding a common instance of two expressions. Algorithms to perform unification have been central to many theorem-proving systems and some programming-language processors.

The task of deriving a unification algorithm automatically is beyond the power of existing program-synthesis systems. In this paper, we use the deductive approach to derive an algorithm from a simple, high-level specification of the unification task. We will identify some of the capabilities required of a theorem-proving system to perform this derivation automatically.

This paper will appear in *Automatic Program Construction* (G. Guiho, ed.), NATO Scientific Series, D. Reidel Pub. Co., Dordrecht, Holland, 1981.

The research was supported in part by the National Science Foundation under Grants MCS-78-02591 and MCS-79-09495, in part by the Office of Naval Research under Contracts N00014-75-C-0816 and N00014-76-C-0687, and in part by the Air Force Office of Scientific Research under Contract AFOSR-81-0014.

INTRODUCTION

In an earlier paper (Manna and Waldinger [1980]) we describe a deductive approach to program synthesis. In this approach, program synthesis is regarded as a theorem-proving task: Given a high-level specification of the purpose of the program, we prove a theorem that establishes the existence of an output satisfying this specification. The proof is restricted to be sufficiently constructive so that the desired program can be extracted directly. This approach is the direct descendant of the technique applied, *e.g.*, by Green [1969] and by Waldinger and Lee [1969].

In the earlier paper, we only applied the technique to very simple examples. In this paper, we consider a somewhat more difficult task: the synthesis of a unification algorithm.

Unification is the process of finding a common instance of two expressions. If such an instance exists, the algorithm is to produce a substitution that will yield that instance when applied to either of the expressions. If no common instance exists, the algorithm is to produce a special symbol indicating this situation. The first unification algorithm appeared in Herbrand's [1930] thesis, but the procedure did not come to widespread attention until it was rediscovered by Prawitz [1960] and employed by Robinson [1965] in his resolution principle for automatic theorem proving. Since then, the algorithm has been used not only for resolution theorem proving but also in many nonresolution theorem provers (see Bledsoe [1977]) and some programming-language processors (*e.g.*, PLANNER, see Hewitt [1971] or PROLOG, see Warren *et al.* [1977] or Colmerauer *et al.* [1979]).

Because of its importance in theorem proving and other applications, some effort has gone into the design of efficient unification algorithms (e.g., Martelli and Montanari [1976]; Paterson and Wegman [1978]) and the extension of the algorithm to more complex logical theories (*e.g.*, higher-order logic, Huet [1975]; associative and commutative theories, Stickel [1975], Livesay *et al.* [1979]).

The unification algorithm was the subject of partial verification efforts (*e.g.*, Waldinger and Levitt [1974]) and an example of automatic debugging (von Henke and Luckham [1974]). An early attempt to synthesize such a program appeared in Manna and Waldinger [1975]. Nevertheless, no complete automatic synthesis, or even verification, of the algorithm has been completed by any system.

The derivation presented in this paper depends on the formulation of a theory of expressions and substitutions. Intuitive observations about these objects can then be expressed and proved within the theory. In this paper, we set down, without proof, those results necessary for the derivation. A full presentation of the theory of expressions and substitutions is included in our forthcoming book (Manna and Waldinger [1982]).

The proof on which the derivation is based is presented in full. A summary of those aspects of the deductive approach necessary to understand the derivation is included. Although this proof can be expressed in the deductive tableau formalism of our [1980] paper, it is given here informally. We do not attempt to describe strategies under which the proof could be generated automatically.

However, afterwards we consider what capabilities would be required of a theorem-proving system to discover such a proof.

THE DEDUCTIVE APPROACH

The specification of a program allows us to express the purpose of the desired program without indicating an algorithm by which that purpose is to be achieved. In general, we are considering the synthesis of programs whose specifications have the form

$$f(a) \Leftarrow \textit{find } z \textit{ such that } R(a,z) \quad \textit{where } P(a).$$

Here, a denotes the input of the desired program and z denotes its output. The *input condition* $P(a)$ expresses the class of legal inputs to which the program is expected to apply. The *output condition* $R(a,z)$ describes the relation the output z is intended to satisfy.

For example, to specify a program to compute the integer square-root of a nonnegative integer n, we would write

$$sqrt(n) \Leftarrow \textit{find } z \textit{ such that } integer(z) \textit{ and } z^2 \leq n < (z+1)^2 \quad \textit{where } integer(n) \textit{ and } 0 \leq n.$$

A specification of the above form describes an applicative program, one that yields an output but produces no side effects. To derive a program from such a specification, we attempt to prove a theorem of the form

$$(\forall a)(\exists z)[\textit{if } P(a) \textit{ then } R(a,z)].$$

This theorem states that, for every input a, there exists an output z satisfying the output condition, provided that the input satisfies the input condition. The proof of this theorem must be constructive, in the sense that, in proving the existence of a satisfactory output z, it must tell us how to find such an output. From this proof, a program to compute z can be extracted.

WELL-FOUNDED INDUCTION

The formation of repetitive program constructs in the deductive approach depends on the application of the principle of mathematical induction. The induction principle we use is the principle of "well-founded induction," which applies to a wide variety of mathematical structures and results in the formation of a recursive procedure in the program being constructed. Before we can present the induction principle, we must introduce the notion of a "well-founded ordering."

Definition: If $\succ$ is a relation over a set S, we will say that $\succ$ satisfies the *decreasing sequence* condition if there are no infinite decreasing sequences $x_1, x_2, x_3, \ldots$ of elements of S; *i.e.*,

there are no sequences such that

$$x_1 \succ x_2 \succ x_3 \succ \ldots .$$

For example, the ordinary "greater-than" relation $\succ$ over the nonnegative integers satisfies the decreasing sequence condition. The same relation over all the integers does not.

Definition: If $\succ$ is a relation over a set S, we will say that $\succ$ is a *well-founded ordering* and S is a well-founded set under this ordering if

- $\succ$ is transitive and
- $\succ$ satisfies the decreasing sequence condition.

We will regard $y \prec x$ as synonymous with $x \succ y$.

For example, the following are all well-founded orderings:

- The $>$ relation over the nonnegative integers
- The subset relation over the finite sets
- The subtree relation over the finite trees.

The *principle of well-founded induction* may be expressed as follows:

> Let $\succ$ be a well-founded ordering over a set S. Then
> to prove that a statement $Q(a)$ is true for all elements a of S,
> consider an arbitrary element a of S,
> assume the induction hypothesis
> $(\forall x)[if\ \ x \prec a\ \ then\ \ Q(x)]$
> and prove that the conclusion
> $Q(a)$
> then follows.

When we are applying the well-founded induction principle in a program-synthesis context, we may use the following special form:

Let $\succ$ be a well-founded ordering over a set S. To construct a program f satisfying the specification

$f(a) \Leftarrow$ find z such that $R(a, z)$
where $P(a)$,

where the inputs a belong to a well-founded set, consider an arbitrary input a, assume the induction hypothesis

$$(\forall x)\left[\begin{array}{l} if\ x \prec a \\ then\ if\ P(x) \\ \qquad then\ R(x, f(x)) \end{array}\right]$$

and then prove the conclusion

$$(\exists z)R(a, z).$$

In other words, we consider an arbitrary input a, and find an output z satisfying the given specification, under the following induction hypothesis: the program $f(x)$ we are trying to construct will satisfy the specification for all inputs x that are less than a in the well-founded ordering.

Application of the induction hypothesis during a proof will cause a recursive call $f(x)$ to appear in the program being constructed. The condition $P(x)$ will ensure that the input x of the recursive call will be a legal input; *i.e.*, it will satisfy the given input condition. The condition $x \prec a$ will ensure that the new recursive call cannot result in an infinite computation.

ℓ-EXPRESSIONS

In this section, we define a class of ℓ-expressions that will contain not only the expressions but also nested lists of expressions formed from a given alphabet.

- *the alphabet*

Suppose that S is an alphabet of symbols, consisting of three disjoint sets:

C: the *constants*

X: the *variables*, and

F: the *function symbols*.

Together, the constants and variables will be referred to as the *atoms* of S. With each function symbol of F is associated a unique positive integer, called its *arity*, indicating how many arguments the function takes.

- *generation rules*

The *expressions* of S are constructed by repeated application of the following *generation rules*:

Any constant of C is an expression;

Any variable of X is an expression;

If f is a function symbol (of arity n)
  and ℓ is a list of expressions (of length n)
then the result of applying f to the expressions in ℓ,
  denoted by $f \bullet \ell$, is an expression.

Note that if $\ell = [\ell_1, \ell_2, \ldots, \ell_n]$ then $f \bullet \ell$ is the expression informally denoted by $f(\ell_1, \ell_2, \ldots, \ell_n)$.

The ℓ-expressions of S are constructed by repeated application of the following generation rules:

The empty list [ ] is an ℓ-expression;

Any expression is an ℓ-expression;

If s is an ℓ-expression
  and m is a list of ℓ-expressions
then the result of inserting s before the first element of m,
  denoted by $s \circ m$, is an ℓ-expression.

Note that if $m = [m_1, m_2, \ldots, m_n]$ then $s \circ m$ is the list of ℓ-expressions informally denoted by $[s, m_1, m_2, \ldots, m_n]$.

• *uniqueness properties*

We assume that each ℓ-expression can only be produced in a unique way from the above rules. This assumption is expressed by the following properties:

$c \neq x$

$c \neq f \bullet \ell$

$x \neq f \bullet \ell$

if $f \bullet \ell = f' \bullet \ell'$
then $f = f'$ *and* $\ell = \ell'$

$c \neq [\,]$

$c \neq s \circ m$

$x \neq [\,]$

$x \neq s \circ m$

$f \bullet \ell \neq [\,]$

$f \bullet \ell \neq s \circ m$

$$[\,] \neq s \circ m$$

$$\textit{if } s \circ m = s' \circ m'$$
$$\textit{then } s = s' \textit{ and } m = m'$$

for all constants c, variables x, function symbols f and f' (of arity n and n', respectively), lists of expressions ℓ and ℓ' (of length n and n', respectively), ℓ-expressions s and s', and lists of ℓ-expressions m and m'.

• *the occurs-in relation*

We will say that an ℓ-expression s *occurs in* an ℓ-expression s', denoted by

$$s \preceq s' \quad \text{or} \quad s' \succeq s,$$

if s is a subexpression of s'; we will say that s *occurs properly in* s', denoted by

$$s \prec s' \quad \text{or} \quad s' \succ s$$

if s occurs in s' but is distinct from s'. We will abbreviate

$$\textit{not } s \prec s' \quad \text{as} \quad s \not\prec s',$$

$$\textit{not } s \preceq s' \quad \text{as} \quad s \not\preceq s',$$

and so forth. Formally, these relations are defined by the following properties:

$$s \preceq s' \quad \textit{if and only if} \quad s \prec s' \textit{ or } s = s' \qquad (\textit{partiality})$$

$$s \not\prec a \qquad (\textit{atom})$$

$$s \not\prec [\,] \qquad (\textit{empty})$$

$$s \prec (f \bullet \ell) \quad \textit{if and only if} \quad s \preceq \ell \qquad (\textit{application})$$

$$s \prec (s' \circ m) \quad \textit{if and only if} \quad s \preceq s' \textit{ or } s \preceq m \qquad (\textit{insertion})$$

for all atoms a, function symbols f (of arity n), lists of expressions ℓ (of length n), ℓ-expressions s and s', and lists of ℓ-expressions m.

We assume as part of its definition that the occurs-in relation is well-founded. This is a way of expressing formally that all the ℓ-expressions are finite. It follows that the relation is irreflexive, *i.e.*,

$$s \not\prec s \qquad (\textit{irreflexivity})$$

for all ℓ-expressions s.

The definition implies the following *component* properties of $\prec$:

$$\ell \prec f \bullet \ell$$

$$s \prec s \circ m$$

$$m \prec s \circ m$$

for every function symbol f (of arity n), list of expressions ℓ (of length n), ℓ-expression s, and list of ℓ-expressions m.

• *the vars function*

The value of $vars(s)$ is the set of variables that occur in the ℓ-expression s. Formally, we define

$$vars(c) = \{\} \qquad (constant)$$

$$vars(x) = \{x\} \qquad (variable)$$

$$vars([\,]) = \{\} \qquad (empty)$$

$$vars(f \bullet \ell) = vars(\ell) \qquad (application)$$

$$vars(s \circ m) = vars(s) \cup vars(m) \qquad (insertion)$$

for all constants c, variables x, function symbols f (of arity n), lists of expressions ℓ (of length n), ℓ-expressions s, and lists of ℓ-expressions m.

Now let us state a proposition relating the occurs-in relation with the *vars* function.

Proposition (variables): For every variable y and ℓ-expression s,

$$y \in vars(s) \quad \textit{if and only if} \quad y \preceq s.$$

In other words, the elements of $vars(s)$ are indeed those variables that occur in s. The proof is by well-founded induction over the occurs-in relation itself.

SUBSTITUTIONS

Substitution is the operation of replacing certain variables of an ℓ-expression by other expressions. We begin by giving an informal exposition of substitutions; subsequently, we give a formal treatment of the same notion.

Informally, we will represent a substitution θ as a set of *replacements*

$$\theta = \{x_1 \leftarrow e_1,\ x_2 \leftarrow e_2,\ \ldots,\ x_n \leftarrow e_n\},$$

where $x_1, x_2, \ldots, x_n$ are distinct variables in X and $e_1, e_2, \ldots, e_n$ are expressions such that $e_i \neq x_i$. Thus, replacements of the form $x \leftarrow x$ are excluded from substitutions, and substitutions of the form $\{x \leftarrow e,\ x \leftarrow e'\}$ are not allowed.

If $x_i \leftarrow e_i$ is a replacement in the substitution θ, we will refer to x_i as the *variable* and e_i as the *expression* of the replacement. We will denote by

$dom(\theta)$: the set of variables $\{x_1, x_2, \ldots, x_n\}$ affected by the substitution θ,

called the *domain* of θ, and

$range(\theta)$: the set of variables that occur in e_1, e_2, ..., or e_n,

called the *range* of θ.

The result $s \blacktriangleleft \theta$ of applying such a substitution θ to an ℓ-expression s is obtained by simultaneously replacing every instance of the variables x_1, x_2, ..., and x_n by the corresponding expressions e_1, e_2, ..., and e_n.

For example, if θ is the substitution

$$\{x \leftarrow f(y),\ y \leftarrow g(a,z)\},$$

then

$$dom(\theta) = \{x, y\},$$

$$range(\theta) = \{y, z\};$$

furthermore, if s is the ℓ-expression

$$[\, x,\ [\, g(a,x),\ y\,]\,],$$

then the result $s \blacktriangleleft \theta$ of applying θ to s is

$$[\, f(y),\ [\, g(a, f(y)),\ g(a,z)\,]\,].$$

Note that the replacements are performed simultaneously: thus, the variable y in $f(y)$ above was *not* replaced by $g(a,z)$ even though θ contains a replacement $y \leftarrow g(a,z)$.

Let us be more precise: suppose that S is an alphabet consisting of the constants C, the variables X, and the function symbols F, as before.

• *generation rules*

The *substitutions* are constructed by repeated application of the following *generation rules*. We define the domain and range sets for each substitution at the same time.

- The empty substitution $\{\,\}$ is a substitution,

- $dom(\{\}) = \{\}$,
- $range(\{\}) = \{\}$;

If θ is a substitution,
  x is a variable not in $dom(\theta)$,
  and e is an expression distinct from x,
then
- the result of adding the replacement $x \leftarrow e$ to the substitution θ, denoted by $(x \leftarrow e) \circ \theta$, is also a substitution,
- $dom((x \leftarrow e) \circ \theta) = \{x\} \cup dom(\theta)$,
- $range((x \leftarrow e) \circ \theta) = vars(e) \cup range(\theta)$.

Note that if $\theta = \{x_1 \leftarrow e_1, x_2 \leftarrow e_2, \ldots, x_n \leftarrow e_n\}$ then $(x \leftarrow e) \circ \theta$ is the substitution informally denoted by $\{x \leftarrow e, x_1 \leftarrow e_1, \ldots, x_n \leftarrow e_n\}$. Furthermore,

$$dom((x \leftarrow e) \circ \theta) = \{x, x_1, \ldots, x_n\}$$

$$range((x \leftarrow e) \circ \theta) = vars(e) \cup vars(e_1) \cup \ldots \cup vars(e_n).$$

We call $\circ$ the *addition* function for substitutions.

Substitutions do not have uniqueness properties: we may regard two substitutions as equal even though they have been constructed in different ways. We will say that two substitutions are equal if they have the same effect when applied to an arbitrary ℓ-expression. But first let us define more precisely what we mean by applying a substitution to an ℓ-expression.

- *the apply function*

If θ is a substitution, and s is an ℓ-expression, then the *apply* function $s \blacktriangleleft \theta$ is defined to satisfy the following properties:

$$s \blacktriangleleft \{\} = s \qquad (\textit{empty substitution})$$

$$c \blacktriangleleft \theta = c \qquad (\textit{constant})$$

$$x \blacktriangleleft ((x \leftarrow e) \circ \theta) = e \qquad (\textit{same variable})$$

$$y \blacktriangleleft ((x \leftarrow e) \circ \theta) = y \blacktriangleleft \theta \quad \text{if } x \neq y \qquad (\textit{distinct variable})$$

$$[\,] \blacktriangleleft \theta = [\,] \qquad (\textit{empty list})$$

$$(f \bullet \ell) \blacktriangleleft \theta = f \bullet (\ell \blacktriangleleft \theta) \qquad (\textit{application})$$

$$(s \circ m) \blacktriangleleft \theta = (s \blacktriangleleft \theta) \circ (m \blacktriangleleft \theta) \qquad (\textit{insertion})$$

for all substitutions θ, constants c, variables x and y, expressions e, function symbols f (of arity n), lists of expressions ℓ (of length n), ℓ-expressions s, and lists of ℓ-expressions m.

Note that, in the *same variable* and *distinct variable* properties, we do not require that x and e be distinct or that $x \notin dom(\theta)$ even though these are conditions on the generation rules for substitutions. This means that $(x \leftarrow e) \circ \theta$ is defined in these cases.

Let us now introduce a simple property relating ℓ-expressions and substitutions.

Proposition (monotonicity): For all ℓ-expressions s and s' and substitutions θ,

(a) if $s \prec s'$ then $s \blacktriangleleft \theta \prec s' \blacktriangleleft \theta$,

(b) if $s \preceq s'$ then $s \blacktriangleleft \theta \preceq s' \blacktriangleleft \theta$.

In other words, the subexpression and proper subexpression relations are maintained after the application of a substitution.

• *agreement and equality*

We will say that two substitutions θ_1 and θ_2 *agree* on an ℓ-expression s if

$$s \blacktriangleleft \theta_1 \;=\; s \blacktriangleleft \theta_2.$$

Now we can define the notion of equality for substitutions. We will say that two substitutions θ_1 and θ_2 are *equal*, denoted by $\theta_1 = \theta_2$, if they agree on all ℓ-expressions; *i.e.*,

$\theta_1 = \theta_2$

if and only if

$s \blacktriangleleft \theta_1 = s \blacktriangleleft \theta_2$ for all ℓ-expressions s.

We assume that the only substitutions are those that have been constructed by a finite number of applications of the generation rules; it follows that any substitution θ is a finite set of replacements and that the sets $dom(\theta)$ and $range(\theta)$ are finite sets of variables. This finiteness can be expressed formally by an appropriate induction principle over the substitutions.

• *characterization of domain and range*

Let us state two propositions that characterize the domain and range of a substitution.

Proposition (domain): For every substitution θ and variable x,

$$x \in dom(\theta) \quad \text{if and only if} \quad x \blacktriangleleft \theta \neq x.$$

That is, the domain is the set of all variables affected by the substitution.

Proposition (range): For every substitution θ and variable y,

$$y \in range(\theta) \quad \text{if and only if} \quad \left[\begin{array}{l}\text{there exists a variable } x \text{ such that} \\ x \in dom(\theta) \text{ and } y \in vars(x \blacktriangleleft \theta)\end{array}\right].$$

That is, the range is the set of all variables that may be introduced by a substitution.

Let us now introduce the notion of the composition of two substitutions.

COMPOSITION OF SUBSTITUTION

We define the composition $\theta \diamond \theta'$ of two substitutions θ and θ' to be the substitution satisfying the following property:

$$s \blacktriangleleft (\theta \diamond \theta') = (s \blacktriangleleft \theta) \blacktriangleleft \theta'$$

for all ℓ-expressions s. In other words, applying the composition $\theta \diamond \theta'$ to an ℓ-expression is the same as applying θ first, and then applying θ' to the result.

For example, if

$$\theta = \{x \leftarrow f(y)\}$$

$$\theta' = \{y \leftarrow g(a, x),\ x \leftarrow b\}$$

and

$$s = h(x, y, z),$$

then

$$s \blacktriangleleft \theta = h(f(y),\ y,\ z)$$

and

$$s \blacktriangleleft (\theta \diamond \theta') = (s \blacktriangleleft \theta) \blacktriangleleft \theta' = h(f(g(a, x)),\ g(a, x),\ z).$$

It follows from the definition that composition has the following properties:

$$\theta \diamond \{\} = \theta \qquad (\textit{right empty})$$

$$\{\} \diamond \theta = \theta \qquad (\textit{left empty})$$

$$(\theta_1 \diamond \theta_2) \diamond \theta_3 = \theta_1 \diamond (\theta_2 \diamond \theta_3) \qquad (\textit{associativity})$$

for any substitutions θ, θ_1, θ_2, and θ_3. Because $\diamond$ is associative, we may write expressions of form $(\theta_1 \diamond \theta_2) \diamond \theta_3$ and $\theta_1 \diamond (\theta_2 \diamond \theta_3)$ as $\theta_1 \diamond \theta_2 \diamond \theta_3$ without fear of ambiguity.

SYNTACTIC CATEGORIES

We will regard constants, variables, expressions, and lists as being of distinct "syntactic categories."

Definition: Let S be an arbitrary alphabet. Then the *syntactic categories* of the ℓ-expressions of S are the following five sets:

- the set C of constants,
- the set X of variables,
- the set consisting of the empty list [ ],
- the set of functional expressions of form $f \bullet \ell$,
  where f is a function symbol (of arity n)
  and ℓ is a list of expressions (of length n).
- the set of nonempty lists of form $t \circ m$,
  where t is an ℓ-expression and m is a list of ℓ-expressions.

By the uniqueness properties of ℓ-expressions, every ℓ-expression belongs to precisely one syntactic category. The power of a substitution to change the syntactic category of an ℓ-expression is severely limited by the following observation:

Proposition (syntactic category): For any substitution θ and ℓ-expression s,

(a) if s is not a variable
then s and $s \blacktriangleleft \theta$ are in the same syntactic category.

(b) if e is an expression
then $e \blacktriangleleft \theta$ is an expression.

Note that it is not necessarily true that, if s is a variable, then $s \blacktriangleleft \theta$ is also a variable; it may be a constant or a functional expression. However, the converse is true:

Corollary: For any substitution θ and ℓ-expression s,

if $s \blacktriangleleft \theta$ is a variable
then s is also a variable.

THE AGREEMENT PROPOSITION AND ITS CONSEQUENCES

The following proposition has many useful consequences:

Proposition (agreement): For all substitutions θ and θ' and ℓ-expressions s, we have

$$s \blacktriangleleft \theta = s \blacktriangleleft \theta'.$$

if and only if

for every x, if $x \in vars(s)$ then $x \blacktriangleleft \theta = x \blacktriangleleft \theta'$.

In other words, two substitutions agree on an ℓ-expression precisely when they agree on all the variables of the ℓ-expression.

An immediate consequence of this proposition tells what happens when all the variables of an ℓ-expression are unchanged by a substitution:

Corollary (invariance): For every substitution θ and ℓ-expression s,

$$s \blacktriangleleft \theta = s$$

if and only if

$$vars(s) \cap dom(\theta) = \{\}.$$

In other words, a substitution has no effect on an ℓ-expression precisely when no variable in the domain of the substitution actually occurs in the ℓ-expression. The proof depends on taking θ' to be the empty substitution $\{\}$ in the Agreement Proposition.

Corollary (replacement invariance): For every ℓ-expression s, expression e, and variable x,

if $x \notin vars(s)$
then $s \blacktriangleleft \{x \leftarrow e\} = s$.

In other words, applying a single replacement to an ℓ-expression has no effect if the variable of the replacement does not occur in the ℓ-expression.

Another consequence of the Agreement Proposition gives a useful characterization of the equality between substitutions. We have defined two substitutions to be equal if they agree on all ℓ-expressions. In fact it suffices to show that they agree on all variables:

Proposition (equality): For all substitutions θ and θ', we have

$$\theta = \theta'$$

if and only if

for every variable x, $x \blacktriangleleft \theta = x \blacktriangleleft \theta'$.

According to this proposition, to prove equality between substitutions it suffices to show that they agree on all variables. In fact, it suffices to consider only variables in their domains:

Corollary (equality): For all substitutions θ and θ', we have

$$\left[\begin{array}{l}\text{for every variable } x, \\ \quad \text{if } x \in dom(\theta) \cup dom(\theta') \\ \quad \text{then } x \blacktriangleleft \theta = x \blacktriangleleft \theta'\end{array}\right] \quad \text{if and only if} \quad \theta = \theta'.$$

The following proposition relates the addition and composition functions:

Proposition (addition-composition): For every substitution θ, variable x, and expression e,

$$(x \leftarrow e \blacktriangleleft \theta) \circ \theta = (x \leftarrow e) \diamond \theta.$$

The proof relies on the Equality Proposition.

• *the subtraction function*

We denote by $\theta - x$ the substitution that has no effect on the variable x, but that agrees with θ on all other variables. Formally, we define $\theta - x$ by the following properties:

$$x \blacktriangleleft (\theta - x) = x$$

$$y \blacktriangleleft (\theta - x) = y \blacktriangleleft \theta \quad \text{if} \quad x \neq y$$

for all substitutions θ and variables x and y. It follows (by the Domain Proposition) that

$$x \notin dom(\theta - x)$$

for all substitutions θ and variables x and y.

Proposition (subtraction): For any variable x, expression e, and substitution θ, where $x \not\in e$, *i.e.*, $x \notin vars(e)$, we have

$$e \blacktriangleleft (\theta - x) = e \blacktriangleleft \theta.$$

The following proposition enables us to break down a substitution into its component replacements:

Proposition (decomposition): For every substitution θ and variable x,

$$\theta = (x \leftarrow x \blacktriangleleft \theta) \circ (\theta - x).$$

SET THEORETIC PROPERTIES OF SUBSTITUTIONS

In this section, we give some properties that relate the domain and range of a substitution with the variables of the ℓ-expressions.

Proposition (variable elimination): For any ℓ-expression s, substitution θ, and variable y,

if $y \in dom(\theta)$ and $y \notin range(\theta)$
then $y \notin vars(s \blacktriangleleft \theta)$.

Thus, if a variable occurs in the domain of a substitution but not in its range, then the substitution will remove that variable from any ℓ-expression in which it occurs.

Proposition (variable introduction): For any ℓ-expression s, substitution θ, and variable y,

if $y \in vars(s \blacktriangleleft \theta)$
then $y \in range(\theta)$ or $y \in vars(s)$,

i.e.,

$$vars(s \blacktriangleleft \theta) \subseteq range(\theta) \cup vars(s).$$

In other words, if a variable occurs in an ℓ-expression after a substitution, then it was introduced by the substitution or it occurred in the ℓ-expression originally.

UNIFIERS

Suppose that s and s' are ℓ-expressions and θ is a substitution of some alphabet S of constants, variables, and function symbols. We will say that θ is a *unifier* of s and s' if

$$s \blacktriangleleft \theta = s' \blacktriangleleft \theta.$$

Example: If

$$s = g(x, z)$$

and

$$s' = g(y, f(y))$$

then the substitution

$$\theta = \{x \leftarrow y,\ z \leftarrow f(y)\}$$

is a unifier, because

$$s \blacktriangleleft \theta = s' \blacktriangleleft \theta = g(y, f(y)).$$

Note that θ is not the only unifier of s and s'; *e.g.*,

$$\lambda = \{x \leftarrow b,\ y \leftarrow b,\ z \leftarrow f(b)\}$$

is a unifier, because

$$s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda = g(b, f(b));$$

also,

$$\rho = \{y \leftarrow x,\ z \leftarrow f(x)\}$$

is a unifier, because,

$$s \blacktriangleleft \rho = s' \blacktriangleleft \rho = g(x, f(x)). \quad \blacksquare$$

Not all pairs of ℓ-expressions have a unifier. For example, there is no unifier for

$$s = g(a, b)$$

and

$$s' = g(x, x).$$

For, the result of applying any substitution θ to $g(a, b)$ will be the expression

$$g(a, b)$$

itself, in which the arguments are distinct. On the other hand, the result of applying θ to $g(x, x)$ will be an expression of form

$$q(e, e),$$

in which the arguments are identical. These two results can never be the same.

Two ℓ-expressions will be said to be *unifiable* if they have (at least one) unifier. Thus, $g(x, z)$ and $g(y, f(y))$ are unifiable, but $g(a, b)$ and $g(x, x)$ are not.

The following propositions characterize the unifiers for different categories of ℓ-expressions.

Proposition (application unifier): For any function symbol f (of arity n) and lists ℓ and ℓ' (of length n), we have

λ is a unifier of $f \bullet \ell$ and $f \bullet \ell'$

if and only if

λ is a unifier of ℓ and ℓ'.

Proposition (insertion unifier): For any ℓ-expressions t and t' and lists of expressions m and m',

λ is a unifier of $t \circ m$ and $t' \circ m'$

if and only if

λ is a unifier of t and t'
and
λ is a unifier of m and m'.

GENERALITY

We will say that a substitution θ is *more general than* a substitution λ, denoted by $\theta \succeq_{gen} \lambda$ or $\lambda \preceq_{gen} \theta$, if there exists some substitution ψ such that

$$\lambda = \theta \diamond \psi.$$

In this case, we will also say that λ is an *instance* of θ.

Example:

The substitution

$$\theta = \{y \leftarrow b\}$$

is more general than the substitution

$$\lambda = \{x \leftarrow a,\ y \leftarrow b\},$$

i.e., $\theta \succeq_{gen} \lambda$, because (taking ψ to be $\{x \leftarrow a\}$ in the definition)

$$\lambda = \theta \diamond \{x \leftarrow a\}.$$

In other words, λ is an instance of θ. ∎

Note that, according to our terminology, a substitution is always more general than itself, *i.e.*,

$$\theta \succeq_{gen} \theta \qquad \textit{(reflexivity)}$$

for every substitution θ. Also, the empty substitution is more general than any substitution, *i.e.*,

$$\{\} \succeq_{gen} \theta \qquad \textit{(empty)}$$

for every substitution θ.

MOST-GENERAL UNIFIERS

We have observed that there may be several distinct unifiers for a given pair of ℓ-expressions. In fact, any instance of a unifier is also a unifier.

Proposition (instance of a unifier): For all substitutions θ and λ and ℓ-expressions s and s',

if θ is a unifier of s and s'
and $\theta \succeq_{gen} \lambda$
then λ is a unifier of s and s'.

Definition: A substitution θ is a *most-general unifier* of two ℓ-expressions s and s' if

- θ is a unifier of s and s'; *i.e.*,
  $s \blacktriangleleft \theta = s' \blacktriangleleft \theta$,

and

- θ is more general than any other unifier of s and s'; *i.e.*,
  if $s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda$
  then $\theta \succeq_{gen} \lambda$
  for any substitution λ.

Combining the above proposition and definition, we see that the unifiers of two ℓ-expressions are precisely the instances of a most-general unifier.

Corollary (most-general unifier): For all substitutions θ and λ and ℓ-expressions s and s',

θ is a most general unifier of s and s'

if and only if

[λ *is a unifier of* s *and* s'
if and only if
$\theta \succeq_{gen} \lambda$]

Example:

We have seen that the two ℓ-expressions

$$s = g(x, z)$$

and

$$s' = g(y, f(y))$$

have many unifiers; *e.g.*,

$$\theta = \{x \leftarrow y,\ z \leftarrow f(y)\},$$

$$\lambda = \{x \leftarrow b,\ y \leftarrow b,\ z \leftarrow f(b)\}, \text{ and}$$

$$\rho = \{y \leftarrow x,\ z \leftarrow f(x)\}.$$

It turns out that θ is a most-general unifier of s and s'. In particular, θ is more general than λ and ρ, because

$$\lambda = \theta \diamond \{y \leftarrow b\}$$

and

$$\rho = \theta \diamond \{y \leftarrow x\}.$$

The unifier λ is not most-general. In particular, λ is not more general than θ; *i.e.*,

$$\theta \neq \lambda \diamond \psi$$

for any substitution ψ. For instance,

$$x \blacktriangleleft \theta = y$$

but (by the definition of composition and the *constant* property of the apply function)

$$x \blacktriangleleft (\lambda \diamond \psi) = (x \blacktriangleleft \lambda) \blacktriangleleft \psi = b \blacktriangleleft \psi = b \neq y.$$

Most-general unifiers are not unique; for example, the above substitution ρ is also a most-general unifier of s and s'. In particular, ρ is more general than θ and λ, because

$$\theta = \rho \diamond \{x \leftarrow y\}$$

and

$$\lambda = \rho \diamond \{x \leftarrow b\}.$$

The following proposition concerns the unifiers of a variable and an expression:

Proposition (variable unifier): For any variable x and expression e such that $x \not\prec e$, we have

$\{x \leftarrow e\}$ is a most-general unifier of x and e.

We include the proof of this proposition, because it is not straightforward, and because it may be regarded as an integral part of the synthesis of the unification algorithm.

Proof:

It suffices to show (by the Most-General Unifier Corollary) that, for an arbitrary substitution λ,

λ is a unifier of x and e
if and only if
$\{x \leftarrow e\} \succeq_{gen} \lambda$,

i.e. (by the definition of the generality relation $\succeq_{gen}$),

$x \blacktriangleleft \lambda = e \blacktriangleleft \lambda$
if and only if
$\lambda = \{x \leftarrow e\} \diamond \lambda^*$ for some substitution λ^*.

On the one hand, if

$$\lambda = \{x \leftarrow e\} \diamond \lambda^*$$

for some substitution λ^*, then

$$\begin{aligned} x \blacktriangleleft \lambda &= x \blacktriangleleft (\{x \leftarrow e\} \diamond \lambda^*) \\ &= (x \blacktriangleleft \{x \leftarrow e\}) \blacktriangleleft \lambda^* \\ &\qquad \text{by the definition of composition} \\ &= e \blacktriangleleft \lambda^* \\ &\qquad \text{by the } \textit{same-variable} \text{ property of the apply function,} \end{aligned}$$

and

$$\begin{aligned} e \blacktriangleleft \lambda &= e \blacktriangleleft (\{x \leftarrow e\} \blacktriangleleft \lambda^*) \\ &= (e \blacktriangleleft \{x \leftarrow e\}) \blacktriangleleft \lambda^* \\ &\qquad \text{by the definition of composition} \\ &= e \blacktriangleleft \lambda^* \end{aligned}$$

by the Replacement Invariance Corollary, because $x \notin vars(e)$.

In short,

$$x \blacktriangleleft \lambda = e \blacktriangleleft \lambda.$$

On the other hand, suppose

$$x \blacktriangleleft \lambda = e \blacktriangleleft \lambda.$$

Then

$$\begin{aligned}
\lambda &= (x \leftarrow x \blacktriangleleft \lambda) \circ (\lambda - x) \\
&\qquad \text{by the Decomposition Proposition} \\
&= (x \leftarrow e \blacktriangleleft \lambda) \circ (\lambda - x) \\
&\qquad \text{by our supposition that } x \blacktriangleleft \lambda = e \blacktriangleleft \lambda \\
&= (x \leftarrow e \blacktriangleleft (\lambda - x)) \circ (\lambda - x) \\
&\qquad \text{by the Subtraction Proposition, because } x \not\prec e \\
&= \{x \leftarrow e\} \diamond (\lambda - x) \\
&\qquad \text{by the Addition-Composition Proposition.}
\end{aligned}$$

Therefore, if λ^* is taken to be $\lambda - x$, we have

$$\lambda = \{x \leftarrow e\} \diamond \lambda^*$$

as we had intended.

This concludes the proof. ∎

IDEMPOTENT SUBSTITUTIONS

We will say that a substitution θ is *idempotent* if it has the special property that

$$\theta = \theta \diamond \theta.$$

Example:

The substitution

$$\theta = \{x \leftarrow f(y)\}$$

is idempotent, because

$$\begin{aligned}
\theta \diamond \theta &= \{x \leftarrow f(y)\} \diamond \{x \leftarrow f(y)\} \\
&= \{x \leftarrow f(y)\}
\end{aligned}$$

$$= \theta.$$

On the other hand,

$$\phi = \{x \leftarrow f(x)\}$$

is not idempotent, because

$$\begin{aligned} \phi \diamond \phi &= \{x \leftarrow f(x)\} \diamond \{x \leftarrow f(x)\} \\ &= \{x \leftarrow f(f(x))\} \\ &\neq \phi. \quad \blacksquare \end{aligned}$$

The property of idempotence is characterized by the following proposition.

Proposition (idempotence): A substitution is idempotent if and only if its domain and range are disjoint; *i.e.*,

$$\theta = \theta \diamond \theta \quad \text{if and only if} \quad dom(\theta) \cap range(\theta) = \{\},$$

for all substitutions θ.

MOST-GENERAL IDEMPOTENT UNIFIERS

Let us return to the example from the beginning of this section.

Example:

We have seen that the two ℓ-expressions

$$s = g(x, z)$$

and

$$s' = g(y, f(y))$$

have among their most-general unifiers the substitutions

$$\theta = \{x \leftarrow y,\ z \leftarrow f(y)\}$$

and

$$\rho = \{y \leftarrow x,\ z \leftarrow f(x)\}.$$

Both of these substitutions happen to be idempotent, *i.e.*,

$$\theta = \theta \diamond \theta \quad \text{and} \quad \rho = \rho \diamond \rho.$$

However, not all most-general unifiers are idempotent.

For instance, the substitution

$$\phi = \{x \leftarrow z,\ z \leftarrow f(z),\ y \leftarrow z\}$$

also turns out to be a most-general unifier of s and s'. It is a unifier, because

$$s \blacktriangleleft \phi \ = \ g(z, f(z)) \ = \ s' \blacktriangleleft \phi.$$

It is more general than the most-general unifier θ, because

$$\theta = \phi \diamond \{z \leftarrow y\}.$$

But ϕ is not idempotent, because

$$z \ \in \ \{x, y, z\} = dom(\phi)$$

and

$$z \ \in \ \{z\} = range(\phi);$$

therefore

$$dom(\phi) \cap range(\phi) \ = \ \{z\} \ \neq \ \{\},$$

and hence (by the Idempotence Proposition) ϕ is not idempotent. ∎

Most-general, idempotent unifiers have some properties we will find useful.

Proposition (most-general, idempotent unifier): If θ is a unifier of two ℓ-expressions s and s', then

θ is most-general and idempotent

if and only if

for every unifier λ of s and s', $\quad \lambda = \theta \diamond \lambda$.

Proposition (domain and range): If θ is a most-general, idempotent unifier of two ℓ-expressions s and s', then

(a) $dom(\theta) \ \subseteq \ vars(s) \cup vars(s')$

(b) $range(\theta) \ \subseteq \ vars(s) \cup vars(s')$.

In other words, the only variables that may appear in θ are those that occur in s or s'.

THE UNIFICATION ALGORITHM

A *unification algorithm* is a procedure for finding a most-general, idempotent unifier for two ℓ-expressions, if any unifiers exist at all. Otherwise, it produces a special symbol *nil*, which is assumd to be distinct from any substitution.

The specification for the unification algorithm may be expressed as follows:

> $unify(s, s') \Leftarrow$ find θ such that
> θ is a most-general, idempotent unifier of s and s' and
> $\theta \neq nil$
> or
> s and s' are not unifiable and
> $\theta = nil$
> for all ℓ-expressions s and s'.

According to the deductive approach, then, we must prove the existence of an output θ satisfying this specification; *i.e.*, we prove the following theorem:

$$(\forall s)(\forall s')(\exists \theta)\left[\begin{array}{l}\theta \text{ is a most-general, idempotent unifier of } s \text{ and } s' \ \textit{and}\\ \theta \neq nil\\ \quad \textit{or}\\ s \text{ and } s' \text{ are not unifiable} \ \textit{and}\\ \theta = nil\end{array}\right]$$

In other words,

$$(\forall s)(\forall s')(\exists \theta)\left[\begin{array}{ll}s \blacktriangleleft \theta = s' \blacktriangleleft \theta \ \textit{and} & (\theta \text{ is a unifier})\\ (\forall \lambda)[\textit{if } s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda \ \textit{then } \theta \succeq_{gen} \lambda] \ \textit{and} & (\theta \text{ is most-general})\\ \theta = \theta \diamond \theta \ \textit{and} & (\theta \text{ is idempotent})\\ \theta \neq nil & \\ \quad \textit{or} & \\ (\forall \lambda)[s \blacktriangleleft \lambda \neq s' \blacktriangleleft \lambda] \ \textit{and} & (s \text{ and } s' \text{ are not unifiable})\\ \theta = nil & \end{array}\right]$$

This theorem will also establish that if two ℓ-expressions are unifiable, they have a most-general, idempotent unifier.

Before we give the proof of the theorem, let us look ahead at the program we will ultimately extract from the proof. For clarity, we present the program as a set of properties of the *unify* function; actually the deductive approach will produce the corresponding LISP-like applicative program. Of course, these properties will not be available to us during the proof.

- **general**

For all ℓ-expressions s and s':

$unify(s, s') = \{\}$ if $s = s'$ (*same*)

$unify(s, s') = nil$ (*distinct*)
if s and s' are nonvariables in distinct syntactic categories.

- **constant**

For all constants c and c':

$unify(c, c') = nil$ if $c \neq c'$ (*distinct*)

- **variable**

For all variables x, expressions e, and lists of ℓ-expressions m:

$unify(x, e) = \{x \leftarrow e\}$ if $x \not\preceq e$ (*not-in, left*)

$unify(e, x) = \{x \leftarrow e\}$ (*not-in, right*)
if $x \not\preceq e$ and e is not a variable

$unify(x, e) = unify(e, x) = nil$ if $x \prec e$ (*in*)

$unify(x, m) = unify(m, x) = nil$ (*list*)

- **function**

For all function symbols f and f' (of arities n and n', respectively) and all lists of expressions ℓ and ℓ' (of lengths n and n', respectively):

$unify(f \bullet \ell,\ f' \bullet \ell') = unify(\ell, \ell')$ if $f = f'$ (*same*)

$unify(f \bullet \ell,\ f' \bullet \ell') = nil$ if $f \neq f'$ (*distinct*)

• **list**

For all ℓ-expressions t and t' and all lists of ℓ-expressions m and m':

let $\theta_{hd} = unify(t, t')$
if $\theta_{hd} = nil$
then $unify(t \circ m,\ t' \circ m') = nil$ (no)
else let $\theta_{tl} = unify(m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd})$
  if $\theta_{tl} = nil$
  then $unify(t \circ m,\ t' \circ m') = nil$ $(yes\text{-}no)$
  else $unify(t \circ m,\ t' \circ m') = \theta_{hd} \diamond \theta_{tl}$ $(yes\text{-}yes)$

In expressing the list properties, we have used the notation

let $x = a$
$P(x)$

as an abbreviation for

$P(a)$.

The virtue of this notation is that if $P(x)$ has many instances of x and if a is a lengthy expression, we would be required to rewrite a many times in writing $P(a)$. Thus, without this abbreviation, the final equality above would read

$$unify(t \circ m, t' \circ m') \;=\; unify(t,\ t') \diamond unify(m \blacktriangleleft unify(t, t'),\ m' \blacktriangleleft unify(t, t')).$$

In the *list* properties, we have given separate names for the three equalities, for easy reference. The *no* property corresponds to the case that $\theta_{hd} = nil$, the *yes-no* property to the case that $\theta_{hd} \neq nil$ and $\theta_{tl} = nil$, and the *yes-yes* property to the case that $\theta_{hd} \neq nil$ and $\theta_{tl} \neq nil$.

Now let us examine a proof of the theorem, to see how the above program can be constructed.

The proof is by well-founded induction over an ordering $\prec_{un}$ between pairs $(s,\ s')$ of ℓ-expressions. Rather than choose this ordering in advance, we will proceed with the proof, under the assumption that a satisfactory well-founded ordering can be defined. Afterwards, the proof will motivate the definition of an appropriate ordering $\prec_{un}$.

For two arbitrary ℓ-expressions s and s', we want to find an output θ that will satisfy the specification for s and s'. In other words, we want to prove the conclusion

$$P(s, s'):\quad (\exists\theta)\left[\begin{array}{l} s \blacktriangleleft \theta = s' \blacktriangleleft \theta \ \ and \\ (\forall\lambda)[if\ s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda\ \ then\ \ \theta \succeq_{gen} \lambda]\ \ and \\ \theta = \theta \diamond \theta \ \ and \\ \theta \neq nil \\ \qquad or \\ (\forall\lambda)[s \blacktriangleleft \lambda \neq s' \blacktriangleleft \lambda]\ \ and \\ \theta = nil \end{array}\right]$$

We will be happy if θ satisfies either of the two disjuncts in this desired conclusion.

As our induction hypothesis, we assume that the program $unify(r, r')$ we are trying to construct will satisfy its specification for all inputs r and r' such that the pair (r, r') is strictly less than (s, s') in the selected ordering $\prec_{un}$. In other words, we assume

$$\textit{if } (r,\ r') \prec_{un} (s,\ s')$$
$$\textit{then} \begin{bmatrix} r \blacktriangleleft unify(r,r') = r' \blacktriangleleft unify(r,r') \ \textit{and} \\ (\forall\lambda)[\textit{if } r \blacktriangleleft \lambda = r' \blacktriangleleft \lambda \ \textit{ then } \ unify(r,r') \succeq_{gen} \lambda] \ \textit{and} \\ unify(r,r') = unify(r,r') \diamond unify(r,r') \ \textit{and} \\ unify(r,\ r') \neq nil \\ \qquad \textit{or} \\ (\forall\lambda)[r \blacktriangleleft \lambda \neq r' \blacktriangleleft \lambda] \ \textit{and} \\ unify(r,r') = nil \end{bmatrix}$$

for all ℓ-expressions r and r'.

The proof distinguishes between several cases, corresponding to the properties of the apply function $\blacktriangleleft$. At the end of each case we will give the property of the *unify* function provided by the proof. Together, these properties constitute the final program.

GENERAL CASES

Case (same): $s = s'$.

Then any substitution θ is a unifier, *i.e.*,

$$s \blacktriangleleft \theta = s' \blacktriangleleft \theta.$$

- To find a unifier that is most-general, *i.e.*, such that

$$(\forall\lambda)[\textit{if } s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda \ \textit{ then } \ \theta \succeq_{gen} \lambda],$$

we note that (in this case) any substitution λ satisfies the antecedent $s \blacktriangleleft \lambda = s' \blacktriangleleft \lambda$. Therefore, θ must have the property that

$$\theta \succeq_{gen} \lambda$$

for any substitution λ. The *empty* property of the most-general relation $\succeq_{gen}$, *i.e.*,

$$\{\} \succeq_{gen} \lambda$$

for any substitution λ, suggests that, in this case, to satisfy the first disjunct of the desired conclusion $P(s, s')$, θ can be taken to be the empty substitution $\{\}$.

- The empty substitution {} is idempotent, *i.e.*,

$$\{\} = \{\} \diamond \{\},$$

by either of the *empty* properties of composition.

- We ignore the final requirement, that $\{\} \neq nil$, because any substitution is distinct from *nil*.

Therefore, the empty substitution {} is a most-general, idempotent unifier of s and s', and satisfies the first disjunct of the desired conclusion $P(s, s')$ in this case.

$unify(s, s') = \{\}$ if $s = s'$

Case (distinct): s and s' are nonvariables of distinct syntactic categories.

Then we can show that s and s' are not unifiable, *i.e.*

$$s \blacktriangleleft \lambda \neq s' \blacktriangleleft \lambda,$$

for any substitution λ. For, let λ be an arbitrary substitution. Recall that, because s and s' are not variables, we have (by the Syntactic Categories Proposition) that

$s \blacktriangleleft \lambda$ is in the same syntactic category as s,

$s' \blacktriangleleft \lambda$ is in the same syntactic category as s',

and therefore

$s \blacktriangleleft \lambda$ and $s' \blacktriangleleft \lambda$ are in distinct syntactic categories.

Hence,

$$s \blacktriangleleft \lambda \neq s' \blacktriangleleft \lambda;$$

i.e., s and s' are not unifiable.

It follows that, in this case, we can satisfy the second disjunct of the desired conclusion $P(s, s')$ by taking θ to be *nil*.

$unify(s, s') = nil$ if s and s' are nonvariables of distinct syntactic categories

CONSTANT CASE

Case (distinct): s and s' are distinct constants c and c' respectively.

Then we can show that c and c' are not unifiable, *i.e.*,

$$c \blacktriangleleft \lambda \neq c' \blacktriangleleft \lambda$$

for any substitution λ. For, let λ be an arbitrary substitution. Then (by the *constant* property of the apply function),

$$c \blacktriangleleft \lambda = c$$

and

$$c' \blacktriangleleft \lambda = c'.$$

But, since $c \neq c'$,

$$c \blacktriangleleft \lambda \neq c' \blacktriangleleft \lambda;$$

i.e., c and c' are not unifiable.

It follows that, in this case, we can satisfy the second disjunct of the desired conclusion $P(c, c')$ by taking θ to be *nil*.

$unify(c, c') = nil$ if $c \neq c'$

VARIABLE CASES

Case (not-in, left): s is a variable x, s' is an expression e, and $x \not\preceq e$.

- We want to find a most-general unifier of x and e. However, by the Variable Unifier Proposition, $\{x \leftarrow e\}$ is a most-general unifier of x and e. This suggests taking

  θ to be $\{x \leftarrow e\}$.

- We also want $\theta = \{x \leftarrow e\}$ to be idempotent, *i.e.*, that

  $$\{x \leftarrow e\} \diamond \{x \leftarrow e\} = \{x \leftarrow e\}.$$

We show the equivalent condition (by the Idempotence Proposition) that

$$dom(\{x \leftarrow e\}) \cap range(\{x \leftarrow e\}) = \{\}.$$

We have (from the definition of the domain and range),

$$dom(\{x \leftarrow e\}) = \{x\} \text{ and}$$

$$range(\{x \leftarrow e\}) = vars(e).$$

Because $x \not\prec e$, *i.e.* (by the Variables Proposition) $x \notin vars(e)$, we have

$$dom(\{x \leftarrow e\}) \cap range(\{x \leftarrow e\}) = \{x\} \cap vars(e) = \{\},$$

as we wanted to show.

We have succeeded in showing that $\{x \leftarrow e\}$ is a most-general, idempotent unifier in this case.

$unify(x, e) = \{x \leftarrow e\}$ if $x \not\preceq e$

Case (not-in, right): s is an expression e, s' is a variable x, *not* $x \preceq e$, and e is not a variable.

As in the previous case, we find that $\{x \leftarrow e\}$ is a most-general, idempotent unifier of e and x. The condition that the first argument e is not a variable is not required in the proof; it is included because the possibility that s and s' are both variables is covered by the previous case.

$unify(e, x) = \{x \leftarrow e\}$ if $x \not\preceq e$ and e is not a variable

Case (in): s is a variable x, s' is an expression e, and $x \prec e$.

In this case, we can show that x and e are not unifiable. For, let λ be an arbitrary substitution. Because $x \prec e$, we have (by the Monotonicity Proposition)

$$x \blacktriangleleft \lambda \prec e \blacktriangleleft \lambda$$

and hence (by the *irreflexivity* of $\prec$)

$$x \blacktriangleleft \lambda \neq e \blacktriangleleft \lambda.$$

Therefore, x and e are not unifiable and we can satisfy the second disjunct of the desired conclusion $P(x, e)$ by taking θ to be *nil*.

The symmetric case, in which s is an expression e, s' is a variable x, and $x \prec e$, is treated similarly.

$unify(x, e) = unify(e, x) = nil$ if $x \prec e$

Case (list): s is a variable x and s' is a list of ℓ-expressions m.

In this case, we can show that x and m are not unifiable. For, let λ be an arbitrary substitution. Then (by the Syntactic Categories Proposition), $x \blacktriangleleft \lambda$ is an expression but $m \blacktriangleleft \lambda$ is a list of ℓ-expressions; hence (by the disjointness of the syntactic categories)

$$x \blacktriangleleft \lambda \neq m \blacktriangleleft \lambda;$$

i.e., x and m are not unifiable. Therefore, we can satisfy the second disjunct of the desired conclusion $P(x, m)$ by taking θ to be *nil*.

The symmetric case, in which s is a list of ℓ-expressions m and s' is a variable x, is treated similarly.

$$\boxed{unify(x, m) = unify(m, x) = nil}$$

FUNCTIONAL CASES

Case(same): s and s' are functional expressions $f \bullet \ell$ and $f' \bullet \ell'$, respectively, where $f = f'$.

Recall that (in this case) we are attempting to prove the conclusion

$$P(f \bullet \ell, f \bullet \ell');$$

i.e., we want to find an output θ such that

$$\begin{array}{l}
(f \bullet \ell) \blacktriangleleft \theta = (f \bullet \ell') \blacktriangleleft \theta \ \ and \\
(\forall \lambda)[if\ (f \bullet \ell) \blacktriangleleft \lambda = (f \bullet \ell') \blacktriangleleft \lambda\ \ then\ \ \theta \succeq_{gen} \lambda]\ \ and \\
\theta = \theta \diamond \theta\ \ and \\
\theta \neq nil \\
\quad\ \ or \\
(\forall \lambda)[(f \bullet \ell) \blacktriangleleft \lambda \neq (f \bullet \ell') \blacktriangleleft \lambda]\ \ and \\
\theta = nil.
\end{array}$$

This reduces (by the Application Unifier Proposition), to finding an output θ such that

$$\begin{array}{ll}
(*) & \ell \blacktriangleleft \theta = \ell' \blacktriangleleft \theta\ \ and \\
& (\forall \lambda)[if\ \ell \blacktriangleleft \lambda = \ell' \blacktriangleleft \lambda\ \ then\ \ \theta \succeq_{gen} \lambda]\ \ and \\
& \theta = \theta \diamond \theta\ \ and \\
& \theta \neq nil \\
& \quad\ \ or \\
& (\forall \lambda)[\ell \blacktriangleleft \lambda \neq \ell' \blacktriangleleft \lambda]\ \ and
\end{array}$$

$\theta = nil.$

Recall that we have assumed as our induction hypothesis that (in this case)

$$P(r, r'): \quad if\ (r,\ r') \prec_{un} (f \bullet \ell,\ f \bullet \ell')$$
$$then \begin{bmatrix} r \blacktriangleleft unify(r,r') = r' \blacktriangleleft unify(r,r')\ \ and \\ (\forall\lambda)[if\ r \blacktriangleleft \lambda = r' \blacktriangleleft \lambda\ \ then\ \ unify(r,r') \succeq_{gen} \lambda]\ \ and \\ unify(r,r') = unify(r,r') \diamond unify(r,r')\ \ and \\ unify(r,\ r') \neq nil \\ \qquad or \\ (\forall\lambda)[r \blacktriangleleft \lambda \neq r' \blacktriangleleft \lambda]\ \ and \\ unify(r,r') = nil \end{bmatrix}$$

for all ℓ-expressions r and r'.

The required condition $(*)$ and consequent of the induction hypothesis are identical if we take r to be ℓ, r' to be ℓ', and θ to be $unify(\ell, \ell')$. Therefore, we can satisfy the conclusion if we can establish the appropriate instance of the induction hypothesis's antecedent, *i.e.*,

$$(\ell,\ \ell') \prec_{un} (f \bullet \ell,\ f \bullet \ell). \qquad (application\ ordering)$$

The well-founded ordering $\prec_{un}$ will be chosen subsequently. Assuming it will satisfy this condition, we have found that the desired conclusion $P(f \bullet \ell,\ f \bullet \ell')$ in this case is satisfied if we take θ to be $unify(\ell, \ell')$.

$$\boxed{unify(f \bullet \ell,\ f' \bullet \ell') = unify(\ell, \ell') \qquad \text{if } f = f'}$$

Case (distinct): s and s' are functional expressions $f \bullet \ell$ and $f' \bullet \ell'$, respectively, where $f \neq f'$.

In this case, we can show that $f \bullet \ell$ and $f' \bullet \ell'$ are not unifiable. For any arbitrary substitution λ, we have (by the *application* property of the apply function)

$$(f \bullet \ell) \blacktriangleleft \lambda \ = \ f \bullet (\ell \blacktriangleleft \lambda)$$

and

$$(f' \bullet \ell') \blacktriangleleft \lambda \ = \ f' \bullet (\ell' \blacktriangleleft \lambda).$$

Because f and f' are distinct, we have (by the *uniqueness* properties of ℓ-expressions),

$$f \bullet (\ell \blacktriangleleft \lambda) \ \neq \ f' \bullet (\ell' \blacktriangleleft \lambda),$$

and hence

$$(f \bullet \ell) \blacktriangleleft \lambda \ \neq \ (f' \bullet \ell') \blacktriangleleft \lambda.$$

Thus, $f \bullet \ell$ and $f' \bullet \ell'$ are not unifiable.

Therefore, we can satisfy the second disjunct of the desired conclusion, in this case, if we take θ to be *nil*.

$$\boxed{unify(f \bullet \ell,\ f' \bullet \ell') = nil \qquad \text{if } f \neq f'}$$

LIST CASES

In all of the list cases, s and s' are nonempty lists $t \circ m$ and $t' \circ m'$, respectively, where t and t' are ℓ-expressions and m and m' are lists of ℓ-expressions.

Recall that (in the list cases) we are attempting to prove the conclusion

$$P(t \circ m,\ t' \circ m');$$

i.e., we want to find an output θ such that

$$\begin{array}{l}
(t \circ m) \blacktriangleleft \theta = (t' \circ m') \blacktriangleleft \theta \ \ and \\
(\forall \lambda)[if\ (t \circ m) \blacktriangleleft \lambda = (t' \circ m') \blacktriangleleft \lambda\ \ then\ \ \theta \succeq_{gen} \lambda] \ \ and \\
\theta = \theta \diamond \theta \ \ and \\
\theta \neq nil \\
\qquad or \\
(\forall \lambda)[(t \circ m) \blacktriangleleft \lambda \neq (t' \circ m') \blacktriangleleft \lambda] \ \ and \\
\theta = nil.
\end{array}$$

By the Insertion Unifier Proposition, this decomposes into finding θ such that

$$\begin{array}{lll}
(*) & (1)\ t \blacktriangleleft \theta = t' \blacktriangleleft \theta \ \ and & (head\ unifier) \\
& (2)\ m \blacktriangleleft \theta = m' \blacktriangleleft \theta \ \ and & (tail\ unifier) \\
& (3)\ (\forall \lambda)\left[\begin{array}{l} if\ (t \circ m) \blacktriangleleft \lambda = (t' \circ m') \blacktriangleleft \lambda \\ then\ \ \theta \succeq_{gen} \lambda \end{array}\right] \ \ and & (most\text{-}generality) \\
& (4)\ \theta = \theta \diamond \theta \ \ and & (idempotence) \\
& (5)\ \theta \neq nil & (nonnil) \\
& \qquad or & \\
& (6)\ (\forall \lambda)[t \blacktriangleleft \lambda \neq t' \blacktriangleleft \lambda \ \ or \ \ m \blacktriangleleft \lambda \neq m' \blacktriangleleft \lambda] \ \ and & (ununifiability) \\
& (7)\ \theta = nil. & (nil)
\end{array}$$

The separate conditions of (*) are numbered for future reference. We attempt to establish conditions (1) to (5) or, alternatively, conditions (6) and (7).

Recall that we have assumed as our induction hypothesis (in this case)

$P(r, r')$: *if* $(r, r') \prec_{un} (t \circ m,\ t' \circ m')$

$$then \begin{bmatrix} r \blacktriangleleft unify(r,r') = r' \blacktriangleleft unify(r,r') \ \ and \\ (\forall \lambda)[if \ r \blacktriangleleft \lambda = r' \blacktriangleleft \lambda \ \ then \ \ unify(r,r') \succeq_{gen} \lambda] \ \ and \\ unify(r,r') = unify(r,r') \diamond unify(r,r') \ \ and \\ unify(r,r') \neq nil \\ \quad or \\ (\forall \lambda)[r \blacktriangleleft \lambda \neq r' \blacktriangleleft \lambda] \ \ and \\ unify(r,r') = nil \end{bmatrix}$$

for all ℓ-expressions r and r'.

Let us compare this induction hypothesis with our required conclusion $(*)$. A natural approach would be to observe that one of the first two conditions, say the *tail unifier* condition (2),

$$m \blacktriangleleft \theta = m' \blacktriangleleft \theta,$$

is identical to the condition that

$$r \blacktriangleleft unify(r,r') = r' \blacktriangleleft unify(r,r'),$$

asserted in our induction hypothesis, if we take r to be m, r' to be m', and θ to be $unify(m,m')$. However, we still would have to show the *head unifier* condition (1),

$$t \blacktriangleleft \theta = t' \blacktriangleleft \theta,$$

i.e.,

$$t \blacktriangleleft unify(m,m') = t' \blacktriangleleft unify(m,m').$$

But this condition is not necessarily true: a unifier of m and m' need not be a unifier of t and t'. Therefore, we would fail to prove the condition.

An attempt to do the same for the *head unifier* condition (1),

$$t \blacktriangleleft \theta = t' \blacktriangleleft \theta,$$

would fail for the same reason; these two required conditions are symmetric.

A less straightforward approach is to observe that one of the first two conditions, say the *tail unifier* condition (2),

$$m \blacktriangleleft \theta = m' \blacktriangleleft \theta,$$

is also equivalent to the same condition of the induction hypothesis

$$r \blacktriangleleft unify(r,\ r') = r' \blacktriangleleft unify(r,\ r')$$

under a more complex substitution than we considered earlier.

proof of the tail unifier condition (2):

Recall the definition of the composition of substitutions

$$m^* \blacktriangleleft (\theta_{hd} \diamond \theta_{tl}) = (m^* \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl}$$

for all ℓ-expressions m^* and substitutions θ_{hd} and θ_{tl}. (We have renamed the variables to reflect how we intend to use them.)

Applying this equality to the left-hand side of our required condition

$$m \blacktriangleleft \theta = m' \blacktriangleleft \theta$$

(taking m^* to be m and θ to be $\theta_{hd} \diamond \theta_{tl}$) we obtain the condition

$$(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl} = m' \blacktriangleleft (\theta_{hd} \diamond \theta_{tl}).$$

Applying the same equality to the right-hand side (taking m^* to be m') yields the condition

$$(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl} = (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl}.$$

This condition is identical to the condition in our induction hypothesis

$$r \blacktriangleleft unify(r, r') = r' \blacktriangleleft unify(r, r')$$

if we take r to be $m \blacktriangleleft \theta_{hd}$, r' to be $m' \blacktriangleleft \theta_{hd}$, and θ_{tl} to be $unify(m \blacktriangleleft \theta_{hd}, m' \blacktriangleleft \theta_{hd})$.

Let us retain the abbreviation θ_{tl} for the term $unify(m \blacktriangleleft \theta_{hd}, m' \blacktriangleleft \theta_{hd})$. Thus, the above match has suggested taking

$$\theta \quad \text{to be} \quad \theta_{hd} \diamond \theta_{tl},$$

where

$$\theta_{tl} \quad \text{is} \quad unify(m \blacktriangleleft \theta_{hd}, m' \blacktriangleleft \theta_{hd})$$

and θ_{hd} is any substitution.

Let us rewrite the induction hypothesis for this case, making the substitutions suggested by the above match:

$$\begin{array}{l} \textit{if } (m \blacktriangleleft \theta_{hd}, m' \blacktriangleleft \theta_{hd}) \prec_{un} (t \circ m, t' \circ m') \\ \textit{then} \left[\begin{array}{l} (m \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl} = (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl} \ \textit{and} \\ (\forall \lambda)[\textit{if } (m \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda = (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda \ \textit{then } \theta_{tl} \succeq_{gen} \lambda] \ \textit{and} \\ \theta_{tl} = \theta_{tl} \diamond \theta_{tl} \ \textit{and} \\ \theta_{tl} \neq \textit{nil} \\ \qquad \textit{or} \\ (\forall \lambda)[(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda \neq (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda] \ \textit{and} \\ \theta_{tl} = \textit{nil} \end{array}\right] \end{array}$$

We will refer to this as the *tail induction hypothesis.*

To apply this tail induction hypothesis, we must establish

- the antecedent

$$(8)\quad (m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd}) \prec_{un} (t \circ m,\ t' \circ m'), \qquad (\textit{tail ordering})$$

to ensure that the consequent of the tail induction hypothesis will be true;

- the condition

$$(9)\quad \theta_{tl} \neq \textit{nil}, \qquad (\textit{tail nonnil})$$

to ensure that the second disjunct of the consequent of the tail induction hypothesis will be false, so that the first disjunct must be true.

Regardless of the choice of θ_{hd}, this will establish the *tail unifier* condition (2) of the required conclusion (*). We must also establish the *head unifier* condition (1), the *most-generality* condition (3), and the idempotence condition (4), where θ is taken to be $\theta_{hd} \diamond \theta_{tl}$. We know $\theta_{hd} \diamond \theta_{tl}$ will satisfy the *nonnil* condition (5), since any substitution is distinct from *nil*. In cases where we fail to establish one or more of these conditions, we can alternatively attempt to establish the *ununifiability* condition (6) and the *nil* condition (7) of the required conclusion (*), taking θ to be *nil*.

We consider these conditions one by one, but not in the given order.

proof of the head unifier condition (1):

We must find a substitution θ_{hd} such that $\theta_{hd} \diamond \theta_{tl}$ is a unifier of t and t'; *i.e.*, that

$$t \blacktriangleleft (\theta_{hd} \diamond \theta_{tl}) = t' \blacktriangleleft (\theta_{hd} \diamond \theta_{tl}),$$

or, equivalently (by the definition of composition), that

$$(t \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl} = (t' \blacktriangleleft \theta_{hd}) \blacktriangleleft \theta_{tl}.$$

It thus suffices to find θ_{hd} such that

$$t \blacktriangleleft \theta_{hd} = t' \blacktriangleleft \theta_{hd}.$$

We observe that the above condition is identical to the condition that

$$r \blacktriangleleft \textit{unify}(r,\ r') = r' \blacktriangleleft \textit{unify}(r,\ r'),$$

asserted in our induction hypothesis, if we take r to be t, r' to be t' and θ_{hd} to be $\textit{unify}(t,\ t')$. We retain the abbreviation

$$\theta_{hd} \quad \text{is} \quad \textit{unify}(t,\ t').$$

Let us rewrite the induction hypothesis for this case, making the substitutions suggested by the above match:

$$\begin{array}{l} \textit{if } \langle t,\, t'\rangle \prec_{un} \langle t \circ m,\, t' \circ m'\rangle \\ \textit{then } \left[\begin{array}{l} t \blacktriangleleft \theta_{hd} = t' \blacktriangleleft \theta_{hd} \textit{ and} \\ (\forall \lambda)[\textit{if } t \blacktriangleleft \lambda = t' \blacktriangleleft \lambda \textit{ then } \theta_{hd} \succeq_{gen} \lambda] \textit{ and} \\ \theta_{hd} = \theta_{hd} \diamond \theta_{hd} \textit{ and} \\ \theta_{hd} \neq \textit{nil} \\ \quad \textit{or} \\ (\forall \lambda)[t \blacktriangleleft \lambda \neq t' \blacktriangleleft \lambda] \textit{ and} \\ \theta_{hd} = \textit{nil} \end{array}\right] \end{array}$$

We will refer to this as the *head induction hypothesis.*

To apply this head induction hypothesis, we must establish

- the antecedent

$$\langle t,\, t'\rangle \prec_{un} \langle t \circ m,\, t' \circ m'\rangle, \qquad (\textit{head ordering})$$

to ensure that the consequent of the head induction hypothesis will be true.

- the condition

$$\theta_{hd} \neq \textit{nil} \qquad (\textit{head nonnil})$$

to ensure that the second disjunct of the consequent of the head induction hypothesis will be false, so that the first disjunct must be true.

As usual, we defer discussion of the *head ordering* condition, that

$$\langle t,\, t'\rangle \prec_{un} \langle t \circ m,\, t' \circ m'\rangle,$$

until we have accumulated all such conditions, so that we can define an ordering $\prec_{un}$ to satisfy them all at once.

The *head nonnil* condition $\theta_{hd} \neq \textit{nil}$ is not necessarily true: t and t' need not be unifiable. Let us now consider the alternate possibility.

Case(no): $\theta_{hd} = \textit{nil}$.

Then, by our head induction hypothesis, t and t' are not unifiable; *i.e.*,

$$t \blacktriangleleft \lambda \neq t' \blacktriangleleft \lambda$$

for all substitutions λ. Therefore, we can satisfy the *ununifiability* condition (6) and the *nil* condition (7) of the required conclusion (*), in this case, by taking θ to be *nil*.

$$unify(t \circ m,\ t' \circ m') = nil \qquad \text{if } \theta_{hd} = nil$$

Case: $\theta_{hd} \neq nil$.

That is, $unify(t,\ t') \neq nil$. In this case, our head induction hypothesis establishes that θ_{hd} is indeed a most-general, idempotent unifier of t and t', and therefore the *head unifier* condition (1) is satisfied. It remains to show the *tail ordering* condition (8) and the *tail nonnil* condition (9); these conditions ensure that we can apply the tail induction hypothesis to establish the *tail unifier* condition (2) of the required conclusion (*). It also remains to show the original *most generality* condition (3) and *idempotence* condition (4). As usual, we assume that we can establish the *tail ordering* condition (8), and defer its proof.

proof of the tail nonnil condition (9):

The condition that $\theta_{tl} \neq nil$, *i.e.*, $unify(m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd}) \neq nil$, is not necessarily true: $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$ need not be unifiable. Let us consider the alternate possibility.

Subcase (yes-no): $\theta_{tl} = nil$.

That is, $unify(m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd}) = nil$. Then, by our tail induction hypothesis (where r and r' were taken to be $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$, respectively), $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$ are not unifiable, *i.e.*,

$$(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda \neq (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda,$$

for all substitutions λ. We establish, in this case, the *ununifiability* condition (6), that $(\forall \lambda)[t \blacktriangleleft \lambda \neq t' \blacktriangleleft \lambda \ \textit{or}\ m \blacktriangleleft \lambda \neq m' \blacktriangleleft \lambda]$. For suppose, to the contrary, that, for some substitution λ,

$$t \blacktriangleleft \lambda = t' \blacktriangleleft \lambda,$$

i.e., λ is a unifier of t and t', and

$$m \blacktriangleleft \lambda = m' \blacktriangleleft \lambda,$$

i.e., λ is a unifier of m and m'.

Because λ is a unifier of t and t', and because θ_{hd} is a most-general, idempotent unifier of t and t', we have (by the Most-General, Idempotent Unifier Proposition)

$$\lambda = \theta_{hd} \diamond \lambda.$$

Hence, because λ is a unifier of m and m', *i.e.*, $m \blacktriangleleft \lambda = m' \blacktriangleleft \lambda$, we have

$$m \blacktriangleleft (\theta_{hd} \diamond \lambda) \;=\; m' \blacktriangleleft (\theta_{hd} \diamond \lambda),$$

or equivalently (by the definition of composition)

$$(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda \;=\; (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda.$$

i.e., λ is a unifier of $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$. But this contradicts our earlier finding, that $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$ are not unifiable. Hence, the *ununifiability* condition (6) is established and, in this case, we can establish the required conclusion (*) by taking θ to be *nil*.

$unify(t \circ m,\ t' \circ m') = nil$ if $\theta_{hd} \neq nil$ and $\theta_{tl} = nil$

Subcase (yes-yes): $\theta_{tl} \neq nil$.

That is, the *tail nonnil* condition (9), that $unify(m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd}) \neq nil$, is true. Let us again retrace our steps, to see what we have established.

Since we have assumed the *tail ordering* condition (8), we know that the consequent of the tail induction hypothesis is true (where r and r' were taken to be $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$, respectively.) Because (in this case), the *tail nonnil* condition (9) is true, we know that the second disjunct of the consequent is false, and therefore that the first disjunct must be true. This implies that θ_{tl} is a most-general, idempotent unifier of $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$, and hence (by applying the definition of composition twice) that $\theta_{hd} \diamond \theta_{tl}$ is a unifier of m and m'. This establishes the *tail unifier* condition (2), which was our reason for applying the tail induction hypothesis in the first place.

proof of the most-generality and idempotence conditions (3 *and* 4):

It still remains to show the *most-generality* condition (3),

$$(\forall \lambda)[if\ \ (t \circ m) \blacktriangleleft \lambda = (t' \circ m') \blacktriangleleft \lambda\ \ then\ \ (\theta_{hd} \diamond \theta_{tl}) \succeq_{gen} \lambda]$$

and the *idempotence* condition (4), that

$$\theta_{hd} \diamond \theta_{tl} = (\theta_{hd} \diamond \theta_{tl}) \diamond (\theta_{hd} \diamond \theta_{tl}).$$

For this purpose, it suffices (by the Most-General, Idempotent Unifier Proposition) to show the single condition that

$$(\forall \lambda)\begin{bmatrix} if\ (t \circ m) \blacktriangleleft \lambda = (t' \circ m') \blacktriangleleft \lambda \\ then\ \ \lambda = (\theta_{hd} \diamond \theta_{tl}) \diamond \lambda \end{bmatrix}$$

i.e. (by the Insertion Unifier Proposition),

$$(\forall\lambda)\begin{bmatrix} \textit{if } t \blacktriangleleft \lambda = t' \blacktriangleleft \lambda \textit{ and } m \blacktriangleleft \lambda = m' \blacktriangleleft \lambda \\ \textit{then } \lambda = (\theta_{hd} \diamond \theta_{tl}) \diamond \lambda \end{bmatrix}$$

Suppose that λ is an arbitrary substitution such that

$$t \blacktriangleleft \lambda = t' \blacktriangleleft \lambda$$

and

$$m \blacktriangleleft \lambda = m' \blacktriangleleft \lambda.$$

We would like to show that then

$$\lambda = (\theta_{hd} \diamond \theta_{tl}) \diamond \lambda.$$

Because λ is a unifier of t and t', and because θ_{hd} is a most-general, idempotent unifier of t and t', we have (by the Most-General, Idempotent Unifier Proposition, again) that

$$\lambda = \theta_{hd} \diamond \lambda.$$

Therefore, because $m \blacktriangleleft \lambda = m' \blacktriangleleft \lambda$ we have

$$m \blacktriangleleft (\theta_{hd} \diamond \lambda) = m' \blacktriangleleft (\theta_{hd} \diamond \lambda),$$

i.e. (by the definition of composition),

$$(m \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda = (m' \blacktriangleleft \theta_{hd}) \blacktriangleleft \lambda.$$

In other words, λ is a unifier of $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$.

But then, because θ_{tl} is a most-general, idempotent unifier of $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$, we have (by the Most-General, Idempotent Unifier Proposition, yet again) that

$$\lambda = \theta_{tl} \diamond \lambda.$$

Therefore,

$$\lambda = \theta_{hd} \diamond \lambda = \theta_{hd} \diamond (\theta_{tl} \diamond \lambda) = (\theta_{hd} \diamond \theta_{tl}) \diamond \lambda.$$

In short, we obtain the condition,

$$\lambda = (\theta_{hd} \diamond \theta_{tl}) \diamond \lambda,$$

that we wanted to show.

We conclude that $\theta_{hd} \diamond \theta_{tl}$ satisfies the *most-generality* condition (3) and the *idempotence* condition (4) of the required conclusion (*), and thus in this case we are justified in taking θ to be $\theta_{hd} \diamond \theta_{tl}$.

$$unify(t \circ m,\ t' \circ m') = \theta_{hd} \diamond \theta_{tl} \qquad \text{if } \theta_{hd} \neq nil \text{ and } \theta_{tl} \neq nil$$

This concludes the final case.

THE ORDERING

We have deferred the choice of an ordering $\prec_{un}$ to satisfy the ordering conditions we have accumulated during the proof. The choice of this ordering is not so well-motivated formally as the other steps of this derivation. The ordering conditions to be satisfied by $\prec_{un}$ are as follows:

the *application ordering* condition

$$\langle \ell,\ \ell' \rangle \prec_{un} \langle f \bullet \ell,\ f \bullet \ell' \rangle,$$

the *head ordering* condition

$$\langle t,\ t' \rangle \prec_{un} \langle t \circ m,\ t' \circ m' \rangle,$$

and the *tail ordering* condition

$$\langle m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd} \rangle \prec_{un} \langle t \circ m,\ t' \circ m' \rangle$$

for all function symbols f, lists of expressions ℓ and ℓ', ℓ-expressions t and t', and lists of ℓ-expressions m and m', where $\theta_{hd} \neq nil$, *i.e.*, $unify(t,\ t') \neq nil$.

It would be natural to attempt to use as the definition of $\prec_{un}$ the subexpression ordering on one of the two arguments. However, if we take $\prec_{un}$ to be, say, the ordering $\prec_1$ on the first argument, defined by

$$\langle r,\ r' \rangle \prec_1 \langle s,\ s' \rangle$$

if and only if

$$r \prec s,$$

it will satisfy the first two of these conditions, and will satisfy the third condition in the case that $m \blacktriangleleft \theta_{hd} = m$. However, this ordering may fail to satisfy the third condition if $m \blacktriangleleft \theta_{hd} \neq m$, because $m \blacktriangleleft \theta_{hd}$ may no longer be a subexpression of $t \circ m$, and may in fact be much larger. For example, if

$$t \text{ is } x$$

t' is $g(a, y, b)$

and

m is $[f(x, x, x)]$,

m' is $[z]$

then

$t \circ m$ is $[x,\ f(x, x, x)]$

$t' \circ m'$ is $[g(a, y, b),\ z]$.

In this case

θ_{hd} is $\{x \leftarrow g(a, y, b)\}$

and

$m \blacktriangleleft \theta_{hd}$ is $[f(g(a, y, b),\ g(a, y, b),\ g(a, y, b))]$

$m' \blacktriangleleft \theta_{hd}$ is $[z]$.

Thus, $m \blacktriangleleft \theta_{hd}$ is not a subexpression of $t \circ m$.

In the case that $m \blacktriangleleft \theta_{hd} \neq m$, however, it can be shown that the variables of $m \blacktriangleleft \theta_{hd}$ and $m' \blacktriangleleft \theta_{hd}$ are a proper subset of the variables of $t \circ m$ and $t' \circ m'$; *i.e.*,

$$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) \subset vars(t \circ m) \cup vars(t' \circ m').$$

In other words, θ_{hd} removes variables from m and m' without introducing any that are not in $t \circ m$ or $t' \circ m'$. Thus, the ordering $\prec_{vars}$, defined by

$(r, r') \prec_{vars} (s, s')$

if and only if

$vars(r) \cup vars(r') \subset vars(s) \cup vars(s')$,

will satisfy the *tail ordering* condition in this case. Thus, in the above example,

$$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) = \{y, z\}$$

and

$$vars(t \circ m) \cup vars(t' \circ m') = \{x, y, z\},$$

and hence

$$(m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd}) \ \prec_{vars}\ (t \circ m,\ t' \circ m').$$

However, this ordering will fail to satisfy the first two conditions of $\prec_{un}$, and will also fail to satisfy the third condition in the case that $m \blacktriangleleft \theta_{hd} = m$ and $m' \blacktriangleleft \theta_{hd} = m'$. For example (by the *application* property of *vars*),

$$vars(\ell) = vars(f \bullet \ell)$$

$$vars(\ell') = vars(f \bullet \ell')$$

and hence

$$vars(\ell) \cup vars(\ell') \ = \ vars(f \bullet \ell) \cup vars(f \bullet \ell'),$$

i.e.,

$$not\ [(\ell, \ell') \prec_{vars} (f \bullet \ell,\ f \bullet \ell')].$$

In other words, the first condition, $(\ell,\ \ell') \prec_{un} (f \bullet \ell,\ f \bullet \ell')$, is never satisfied under the $\prec_{vars}$ ordering.

The successful ordering $\prec_{un}$ is a lexicographic combination of these two orderings $\prec_{vars}$ and $\prec_1$, defined by the property

$(r,\ r') \prec_{un} (s,\ s')$

if and only if

$vars(r) \cup vars(r') \ \subset \ vars(s) \cup vars(s')$
or
$vars(r) \cup vars(r') \ = \ vars(s) \cup vars(s') \quad and \quad r \prec s.$

- To see that, under this definition, the *application ordering* condition,

$$(\ell,\ \ell') \prec_{un} (f \bullet \ell,\ f \bullet \ell'),$$

is satisfied, note that (as we mentioned above)

$$vars(\ell) \cup vars(\ell') \ = \ vars(f \bullet \ell) \cup vars(f \bullet \ell')$$

and (by a *component* property of $\prec$)

$$\ell \prec f \bullet \ell.$$

Hence, by the definition of the ordering $\prec_{un}$, the *application ordering* condition is indeed satisfied.

- To see that, under this definition, the *head ordering* condition,

$$\langle t,\ t' \rangle \prec_{un} \langle t \circ m,\ t' \circ m' \rangle,$$

is satisfied, note that (by the *insertion* property of *vars*)

$$vars(t) \subseteq vars(t) \cup vars(m) = vars(t \circ m)$$
$$vars(t') \subseteq vars(t') \cup vars(m') = vars(t' \circ m'),$$

and hence

$$vars(t) \cup vars(t') \subseteq vars(t \circ m) \cup vars(t' \circ m').$$

In case the inclusion is proper, *i.e.*,

$$vars(t) \cup vars(t') \subset vars(t \circ m) \cup vars(t' \circ m'),$$

we have

$$\langle t,\ t' \rangle \prec_{un} \langle t \circ m,\ t' \circ m' \rangle$$

immediately. On the other hand, if the two sets are equal, *i.e.*,

$$vars(t) \cup vars(t') = vars(t \circ m) \cup vars(t' \circ m'),$$

we note that (by a *component* property of $\prec$)

$$t \prec t \circ m.$$

Hence, by the definition of the ordering $\prec_{un}$, the *head ordering* condition is also satisfied in this case.

- Finally, we must show that the *tail ordering* condition,

$$\langle m \blacktriangleleft \theta_{hd},\ m' \blacktriangleleft \theta_{hd} \rangle \prec_{un} \langle t \circ m,\ t' \circ m' \rangle,$$

is satisfied, where $\theta_{hd} = unify(t,\ t') \neq nil$. First, we have

$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd})$

$\subseteq$ $vars(m) \cup range(\theta_{hd}) \cup vars(m') \cup range(\theta_{hd})$
by the Variable Introduction Proposition

$=$ $vars(m) \cup vars(m') \cup range(\theta_{hd})$

$\subseteq$ $vars(m) \cup vars(m') \cup vars(t) \cup vars(t')$

by the Domain and Range Proposition,
because θ_{hd} is a most-general, idempotent unifier for t and t'

$$= \quad vars(t \circ m) \cup vars(t' \circ m')$$

by the *insertion* property of *vars*

In short,

$$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) \quad \subseteq \quad vars(t \circ m) \cup vars(t' \circ m').$$

By the definition of the ordering $\prec_{un}$, we must either show that this inclusion is proper, *i.e.*,

$$(*) \qquad vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) \quad \subset \quad vars(t \circ m) \cup vars(t' \circ m'),$$

or show that

$$(**) \qquad m \blacktriangleleft \theta_{hd} \prec t \circ m.$$

For this purpose, we distinguish between two subcases.

Subcase: $m \blacktriangleleft \theta_{hd} = m$.

Then (by a *component* property of $\prec$)

$$m \blacktriangleleft \theta_{hd} \quad = \quad m \quad \prec t \circ m.$$

Subcase: $m \blacktriangleleft \theta_{hd} \neq m$.

In this case, we will show $(*)$, that the inclusion is proper, *i.e.*,

$$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) \quad \subset \quad vars(t \circ m) \cup vars(t' \circ m').$$

We have already shown the $\subseteq$ inclusion; therefore, it suffices to show the existence of a variable z such that

$$(\ddagger) \qquad z \in vars(t \circ m) \cup vars(t' \circ m')$$

but

$$(\dagger\dagger) \qquad z \notin vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}).$$

First, because $m \blacktriangleleft \theta_{hd} \neq m$, we know (by the Invariance Corollary) that

$$vars(m) \cap dom(\theta_{hd}) \neq \{\},$$

i.e., there is a variable z such that

$$z \in vars(m)$$

and

$$z \in dom(\theta_{hd}).$$

We know (by the *insertion* property of *vars*) that

$$vars(m) \subseteq vars(t) \cup vars(m) = vars(t \circ m).$$

Then, because $z \in vars(m)$, we have the desired property (†),

$$z \in vars(t \circ m) \cup vars(t' \circ m').$$

Next, because θ_{hd} is idempotent, we have (by Idempotence Proposition)

$$dom(\theta_{hd}) \cap range(\theta_{hd}) = \{\},$$

and, thus, because

$$z \in dom(\theta_{hd}),$$

we have

$$z \notin range(\theta_{hd}).$$

It follows (by the Variable Elimination Proposition) that

$$z \notin vars(m \blacktriangleleft \theta_{hd})$$

$$z \notin vars(m' \blacktriangleleft \theta_{hd}),$$

and hence

$$z \notin vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}).$$

This is the desired property (††). We have thus established the proper inclusion

$$vars(m \blacktriangleleft \theta_{hd}) \cup vars(m' \blacktriangleleft \theta_{hd}) \subset vars(t \circ m) \cup vars(t' \circ m').$$

In both subcases, we can conclude that

$$(m \blacktriangleleft \theta_{hd}, m' \blacktriangleleft \theta_{hd}) \prec_{un} (t \circ m, t' \circ m').$$

Thus, the *tail ordering* condition for $\prec_{un}$ is satisfied.

This concludes the entire derivation proof. ∎

ALTERNATE DERIVATIONS

We have followed only one proof of the specification of the desired theorem. Had we followed other proofs, different programs would have resulted. For example, in the *list* cases of the above derivation, we first matched the *tail unifier* condition $m \blacktriangleleft \theta = m' \blacktriangleleft \theta$ against the induction hypothesis (after applying the definition of composition); had we instead matched the symmetric *head unifier* condition $t \blacktriangleleft \theta = t' \blacktriangleleft \theta$, another proof would have been obtained, and the *list* cases of the resulting program would have been as follows:

let $\theta^*_{tl} = unify(m,\ m')$
if $\theta^*_{tl} = nil$
then $unify(t \circ m,\ t' \circ m') = nil$
else let $\theta^*_{hd} = unify(t \blacktriangleleft \theta^*_{tl},\ t' \blacktriangleleft \theta^*_{tl})$
  if $\theta^*_{hd} = nil$
  then $unify(t \circ m,\ t' \circ m') = nil$
  else $unify(t \circ m,\ t' \circ m') = \theta^*_{tl} \diamond \theta^*_{hd}$.

This program will also satisfy the same specification as the original program but, because it examines the list from right to left rather than left to right, it may produce a different most-general, idempotent unifier.

In general, by exploring different branches of the proof tree, we may obtain families of different unification algorithms analogous to the families of different sort programs obtained from a single specification by Clark and Darlington [1980]. The particular derivation we obtained did not take the efficiency of the final program into account. Other branches of the derivation tree lead to more efficient unification algorithms.

AUTOMATION OF THE PROOF

Our primary objective in examining the above derivation in such detail is to consider the computational prerequisites for discovering the proof automatically. Let us review the proof from this point of view.

The first requirement of a theorem-proving system for program synthesis is that it be able to prove theorems that contain existential quantifiers and that require mathematical induction. Existential quantifiers are necessary to transform the specification into a theorem, and induction is necessary to introduce repetitive constructs into the target program. Although resolution theorem provers, say, can prove theorems with existential quantifiers, and several theorem provers (*e.g.*,

Boyer and Moore [1975], Huet and Hullot [1980]) can do proofs by induction, it is rare to see these abilities combined.

The amount of knowledge about ℓ-expressions and substitutions necessary to produce the above proof is formidable. If such knowledge were built into the system, the system would then be specially tailored to this subject domain, and would loose generality. On the other hand, if the knowledge were provided to the system as a set of axioms, the system would also need to know how to use the knowledge efficiently.

Much of the derivation proof is fairly mechanical. At each stage, one must decide which property to apply next, from a finite collection of legal next steps. However, certain steps are not straightforward, and are motivated only by their ultimate success. For example, in the list cases, the straightforward use of the induction hypothesis failed, but the application of the definition of composition allowed us to use the induction hypothesis in a more general way, and resulted in the introduction of the composition $\theta_{hd} \diamond \theta_{tl}$ in the final program.

In finding the well-founded ordering $\prec_{un}$, the use of the sets $vars(s)$ and $vars(s')$ of variables in the ℓ-expressions s and s' was not suggested by the specification, which makes no reference to this notion.

The idempotence condition $\theta = \theta \diamond \theta$ was included in the initial specification. This condition played a vital part in the proof; however, the unification algorithm would be equally useful without this property. Had the idempotence condition not been required initially, it or an equivalent condition would have had to be invented and added to the specification in the middle of the proof.

Even with the idempotence condition provided, the proof seems somewhat more difficult than current theorem-provers can produce. Our hope is that studying hand derivations of this sort will enable us to improve the power of automatic systems.

INTERACTIVE SYNTHESIS

Although the above proof may be beyond the power of current automatic systems, a partially interactive system could be used to produce it with known techniques. This approach requires more human effort, but it still would convey many of the benefits of automatic synthesis:

- The person would provide those steps that require cleverness but the system would take care of the routine details.
- Whatever mistakes the person might make, the system would not permit him to produce a program that did not meet its specification.
- The program would be accompanied by a full proof of its correctness.
- The derivation could be retained, so that if the program needed modification, the appropriate portions of the program could be updated without endangering its correctness.

- The assumptions on which the correct operation of the program depends would be made explicit.

Of course, for an interactive system to be successful, it would have to communicate in terms the person would be able to understand.

ACKNOWLEDGMENTS

We would like to thank Yoni Malachai, Pierre Wolper, and Frank Yellin, for their careful reading of the manuscript and their many suggestions; and Evelyn Eldridge-Diaz for her TEXing of the paper.

REFERENCES

Bledsoe, W. W. [Aug. 1977], "Non-resolution theorem proving," Artificial Intelligence Journal, Vol. 9, No. 1, pp. 1-35.

Boyer, R. S. and J S. Moore [Jan. 1975], "Proving theorems about LISP functions," JACM, Vol. 22, No. 1, pp. 129-144.

Clark, K. and J. Darlington [Feb. 1980], "Algorithm classification through synthesis," Computer Journal, Vol. 23, No. 1, pp. 61-65.

Colmerauer, A., H. Kanoui and M. van Caneghem [1979], "Etude et realisation d'un systeme Prolog," Internal Report, Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Université d' Aix-Marseille II.

Green, C. C. [May 1969], "Application of theorem proving to problem solving," Proceedings of the International Joint Conference on Artificial Intelligence, Washington DC, pp. 219-239.

Herbrand, J. [1930], "Researches in the theory of demonstration," in *From Frege to Gödel: a source book in mathematical logic*, 1879-1931 (J. Van Heijenoort, ed.), Harvard University Press, Cambridge, MA, 1967, pp. 525-581.

Hewitt, C. [Apr. 1971], "Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot," Ph.D. thesis, MIT, Cambridge, MA.

Huet, G. P. [June 1975], "A unification algorithm for typed λ-calculus," Theoretical Computer Science, Vol. 1, No. 1, pp. 27-57.

Huet, G. P. and J.-M. Hullot [Oct. 1980], "Proofs by induction in equational theories with constructors," Proceedings of Symposium on Foundations of Computer Science, Syracuse, NY, pp. 96-107.

Livesay, M., J. Siekmann, P. Szabó and E. Unvericht [Feb. 1979], "Unification problems for combinations of associativity, commutativity, distributivity and idempotence axioms," Proceedings of the Fourth Workshop on Automated Deduction, Austin, Texas, pp. 175-184.

Manna, Z. and R. Waldinger [Summer 1975], "Knowledge and reasoning in program synthesis," Artificial Intelligence Journal, Vol. 6, No. 2, pp. 175-208.

Manna, Z. and R. Waldinger [Jan. 1980], "A deductive approach to program synthesis," ACM Transactions on Programming Languages and Systems, Vol. 2, No. 1, pp. 92-121.

Manna, Z. and R. Waldinger [1982], *Deductive Basis for Computer Programming*, forthcoming.

Martelli, A. and U. Montanari [July 1976], "Unification in linear time and space: a structured presentation,' Internal Report, IEI, Pisa.

Paterson, M. S. and M. N. Wegman [April 1978], "Linear unification," Journal of Computer and System Sciences, Vol. 16, No. 2, pp. 158-167.

Prawitz, D. [1960], "An improved proof procedure," Theoria, Vol. 26, pp. 102-139.

Robinson, J. A. [Jan. 1965], "A machine-oriented logic based on the resolution principle", JACM, Vol. 12, No. 1, pp. 23-41.

Stickel, M. E. [Sept. 1975], "A complete unification algorithm for associative-commutative functions," Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, pp. 71-76.

Von Henke, F. W. and D. C. Luckham [April 1974], "A methodology for verifying programs," Proceedings of the International Conference on Reliable Software, Los Angeles, CA, pp. 156-164.

Waldinger, R. J. and R. C. T. Lee [May 1969], "PROW: a step toward automatic program writing," Proceedings of the International Joint Conference on Artificial Intelligence, Washington, DC, pp. 241-252.

Waldinger, R. J. and K. N. Levitt [1974], "Reasoning about programs," *Artificial Intelligence*, Vol. 5, pp. 235-316.

Warren, D., L. M. Pereira and F. Pereira [Aug 1977], "PROLOG – the language and its implementation compared with LISP," Proceedings of Symposium on Artificial Intelligence and Programming Languages, SIGPLAN Notices (ACM), Vol. 12, No. 8, pp. 109-115.