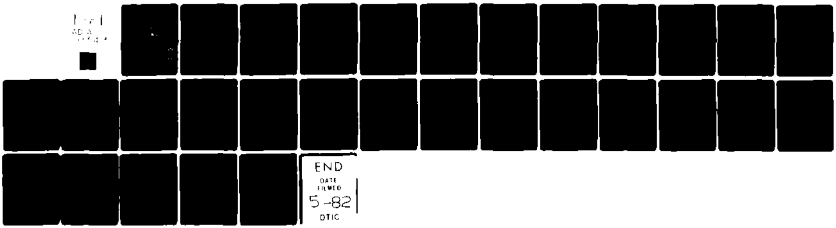
1982-03

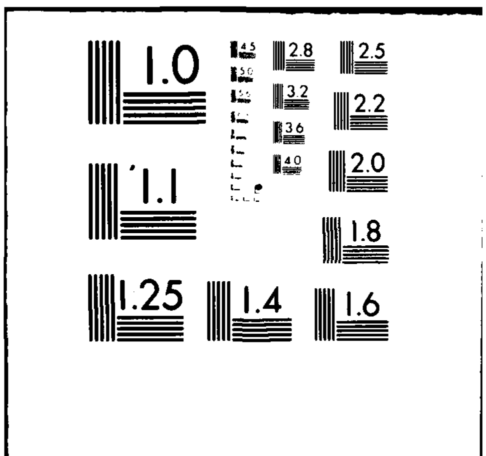
Derived Preconditions and Their Use in Program Synthesis

Smith, Douglas R.

Monterey, California. Naval Postgraduate School

AD-A113 431 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 12/1
DERIVED PRECONDITIONS AND THEIR USE IN PROGRAM SYNTHESIS.(U)
MAR 82 D R SMITH
UNCLASSIFIED NPS-52-82-002 NL





MICROCOPY RESOLUTION TEST CHART

NPS52-82-002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Derived Preconditions
and Their Use in Program Synthesis

Douglas R. Smith

March 1982

Prepared for:
Naval Postgraduate School
Monterey, California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

David A. Schrady
Acting Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgradaute School with funds provided by the Chief of Naval Research.

This report was prepared by:

DOUGLAS R. SMITH
Assistant Professor
of Computer Science

Reviewed by:

GORDON H. BRADLEY, Chairman
Department of Computer Science

Released by:

WILLIAM M. TOLLES
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
NPS52-82-002

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Derived Preconditions
and Their Use in Program Synthesis

5. TYPE OF REPORT & PERIOD COVERED
Technical Report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Douglas R. Smith

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, CA 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
61152N;RR000-01--10
N0001482AF00001

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, CA 93940

12. REPORT DATE
March 1982

13. NUMBER OF PAGES
31

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
To appear in: Sixth Conference on Automated Deduction, Lecture Notes in Computer Science, Springer-Verlag, 1982.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
theorem proving, deduction, program synthesis, problem reduction, divide and conquer.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper we pose and begin to explore a deductive problem more general than that of finding a proof that a given goal formula logically follows from a given set of hypotheses. The problem is most simply stated in the propositional calculus: given a goal A and hypothesis H we wish to find a formula P, called a precondition, such that A logically follows from $P \wedge H$. A precondition provides any additional conditions under which A can be shown to follow from H. A slightly more complex definition of preconditions in a first-order theory is given and used throughout the paper. A formal system based on natural deduc-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

tion is presented in which preconditions can be derived. A number of examples are then given which show how derived preconditions are used in a program synthesis method we are developing. These uses include theorem proving, formula simplification, simple code generation, the completion of partial specifications for a subalgorithm, and other tasks of a deductive nature.

Derived Preconditions and Their Use in Program Synthesis*

Douglas R. Smith
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940, USA

30 November 1981; Revised 9 March 1982

Abstract

In this paper we pose and begin to explore a deductive problem more general than that of finding a proof that a given goal formula logically follows from a given set of hypotheses. The problem is most simply stated in the propositional calculus: given a goal A and hypothesis H we wish to find a formula P, called a precondition, such that A logically follows from $P \wedge H$. A precondition provides any additional conditions under which A can be shown to follow from H. A slightly more complex definition of preconditions in a first-order theory is given and used throughout the paper. A formal system based on natural deduction is presented in which preconditions can be derived. A number of examples are then given which show how derived preconditions are used in a program synthesis method we are developing. These uses include theorem proving, formula simplification, simple code generation, the completion of partial specifications for a subalgorithm, and other tasks of a deductive nature.

0. Introduction

Traditionally, the subject of automatic theorem proving has dealt with the problem of finding a proof that a given goal formula A logically follows from a

* The work reported herein was supported by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

given hypothesis H. In this paper we pose a more general deductive problem and suggest that systems for solving this more general problem can extend the utility of deductive mechanisms, and provide a framework for overcoming some problematic features of current theorem provers. The problem is most simply stated in the propositional calculus: given a goal A and hypothesis H we wish to find a formula P, called a precondition, such that A logically follows from $P \wedge H$. In other words a precondition provides any additional conditions under which A can be shown to follow from H.

A formal system in which preconditions can be derived is described in section 2. Each rule in this natural deduction-like system has a reduction component which reduces a goal A_0 to subgoals $A_1, A_2, \ldots, A_k$ and a composition component which composes preconditions of subgoals $A_1, A_2, \ldots, A_k$ to form a precondition of A_0.

After presenting basic terminology in section 1 a formal system for deriving preconditions is given in section 2. A number of examples are presented in section 3 which show how derived preconditions are used in a program synthesis method we are developing [9,10]. These uses include theorem proving, formula simplification, simple code generation, the completion of partial specifications for a subalgorithm, and other tasks of a deductive nature.

1. Terminology

The examples given below are drawn from a program synthesis system which works within a many-sorted first-order theory TT. The theory includes data types such as $\mathbb{N}$ (natural numbers), LIST($\mathbb{N}$) (linear lists of natural numbers), and BAGS($\mathbb{N}$) (multisets of natural numbers). We will use the (possibly subscripted) symbols i,j,k for variables ranging over $\mathbb{N}$, x,y,z for variables over LIST($\mathbb{N}$), and B as a variable over BAGS($\mathbb{N}$). The theory also includes a number of functions and predicates defined on these types and axiomatic specifications of their interactions. The notions of term, atomic formula, literal, and (well-formed) formula have their usual definitions [5]. Let T and F be propositional constants which have the values true and false respectively in all models of TT. We make use of a distinguished subset of the theorems of TT called known theorems which are assumed to be immediately available to the deductive system. The set of known theorems may change over time but initially includes all axioms of TT. All of the known theorems required by the examples are listed in the appendix.

Let $Q_1x_1\ Q_2x_2\cdots Q_nx_n\ G$ be a closed formula not necessarily in prenex form where Q_i is either $\exists$ or $\forall$ for $i=1,2,\ldots,n$. A $x_1x_2\cdots x_n$-precondition of $Q_1x_1\ Q_2x_2\cdots Q_nx_n\ G$ is a quantifier-free formula P dependent only on variables $x_1,x_2,\ldots,x_n$ such that

$$Q_1x_1Q_2x_2\cdots Q_nx_n[\ P \Rightarrow G\ ]$$

is valid in TT. P is also a weakest $x_1x_2\cdots x_n$-precondition if

$$Q_1x_1Q_2x_2\cdots Q_nx_n[\ P \equiv G\ ]$$

is valid in TT.

Two well-known special cases of these concepts can be given. First, if T can be derived as a $x_1x_2\cdots x_n$-precondition of a goal $Q_1x_1\ Q_2x_2\cdots Q_nx_n\ G$ then the derivation is in fact a proof of the validity of $Q_1x_1\ Q_2x_2\cdots Q_nx_n\ G$ since

$$Q_1x_1Q_2x_2\cdots Q_nx_n\ [T \Rightarrow G] \equiv Q_1x_1Q_2x_2\cdots Q_nx_n\ G$$

Therefore any system for deriving preconditions can also be used for theorem proving. Second, Dijkstra's concept [3] of a "weakest pre-condition" WP(S,R) of a program S with respect to post-condition R may be defined as a weakest q-precondition of

$$\forall q \exists k \exists p[\ \mathrm{TERMINATE}(S,q,k,p) \wedge R(p))\ ]$$

where TERMINATE(S,q,k,p) holds iff program S activated in initial state q terminates within k steps (assuming a suitable definition of a program step) in a final state p. I.e.,

$$\forall q[\ \mathrm{WP}(S,R)[q] \equiv \exists k \exists p\ \mathrm{TERMINATE}(S,q,k,p) \wedge R(p)\ ]$$

Our program synthesis method is not directly related to Dijkstra's approach to algorithm design [3].

In general a given goal may have many preconditions. Characteristics of a useful precondition seem to depend on the application domain. In program synthesis we generally want preconditions which are a) easily computable, b) in as simple a form as possible, and c) as weak as possible. (Criterion (c) prevents the boolean constant F from being an acceptable precondition for all goals.) Clearly there is a tradeoff between these criteria. We are currently investigating the possibility of measuring each criterion by a separate heuristic function, then combining the results to form a net complexity measure on preconditions. For reasons to be discussed later we assume that such a complexity measure ranges over a well-founded set (such as $\mathbb{N}$ under the usual $<$ relation) and that we seek to minimize complexity over all preconditions. In this paper however we are mostly concerned with setting up a formal system within which preconditions can be derived, and showing how to solve some program synthesis problems using it.

2. A Formal System for Deriving Preconditions

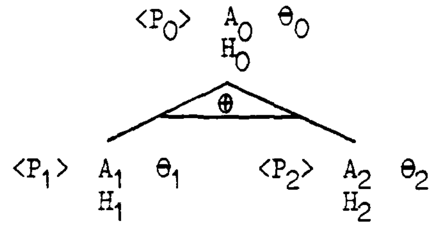
2.1 Goal Preparation

In presenting a set of rules which allow us to derive preconditions we use the notation $\frac{A}{H}$ to denote the statement that well-formed formula A logically follows from the set of hypotheses H in TT, i.e., $h_1 \wedge h_2 \wedge \dots \wedge h_k \Rightarrow A$ is valid in TT where $H = \{h_1, h_2, \dots h_k\}$.

A *goal statement* $\frac{A}{H}$ *and the known theorems of* TT are prepared as follows. First, all occurences of equivalence ($\equiv$) and implication ($\Rightarrow$) signs are eliminated and negation signs are moved in as far as possible. H and the known theorems of TT are then skolemized in the usual way [5], i.e., existentially quantified variables are replaced by skolem functions of the universally quantified variables on which they depend. Quantifiers are then dropped with the understanding that all remaining variables are universally quantified. The goal A is skolemized in a dual manner with universally quantified variables replaced by skolem functions of the existential variables on which they depend. All quantifiers are then dropped with the understanding that all variables in A which remain are existentially quantified. The preparation of A is equivalent (via duality of goals and assertions) to preparing ~A as an hypothesis then taking the negation of the result as our prepared goal.

2.2 Reduction/Composition Rules

Rules which reduce a goal statement to two subgoal statements are expressed in the following form:



where A_0, A_1, and A_2 are goal formulas, H_0, H_1, and H_2 are sets of hypotheses, θ_0, θ_1, and θ_2 are substitutions, P_0, P_1, and P_2 are formulas (the derived preconditions), and $\oplus$ is either $\vee$ or $\wedge$. A rule of this form asserts that if P_i is a (weakest) precondition of $H_i\theta_i \Rightarrow A_i\theta_i$ where i=1,2 then P_0 is a (weakest) precondition of $H_0\theta_0 \Rightarrow A_0\theta_0$. P_0 generally is $P_1 \oplus P_2$. Substitution θ_0 is formed from substitutions θ_1 and θ_2 in ways that depend on $\oplus$.

If $\oplus$ is $\wedge$ then θ_0 is the unifying composition of θ_1 and θ_2, denoted $uc(\theta_1, \theta_2)$ [7]. If $\theta_0 = uc(\theta_1,\theta_2)$ then θ_0 is a most general substitution such that for any literal L

$$(L\theta_1)\theta_0 = (L\theta_0)\theta_1 = L\theta_0 = (L\theta_2)\theta_0 = (L\theta_0)\theta_2.$$

$uc(\theta_1,\theta_2)$ may be computed by finding the most general unifier of

$$(t_1,\ldots,t_n,t_{n+1},\cdots,t_{n+m})$$

$$(v_1,\ldots,v_n,v_{n+1},\cdots,v_{n+m})$$

where

$$\theta_1 = \{t_1/v_1,\ldots,t_n/v_n\}$$

$$\theta_2 = \{t_{n+1}/v_{n+1},\ldots,t_{n+m}/v_{n+m}\}.$$

If these expressions cannot be unified then the result is a special atom NIL. For example,

$$uc(\{a/z\},\{b/z\}) = NIL$$

$$uc(\{\},\{a/z\}) = \{a/z\}$$

$$uc(\{f(x)/z\},\{f(a)/z\}) = \{f(a)/z,a/x\}$$

If $\oplus$ is $\vee$ then θ_0 is formed by the <u>disjunctive composition</u> of P_1, θ_1, P_2 and θ_2, which is denoted $dc(P_1,\theta_1,P_2,\theta_2)$. The disjunctive composition may be computed as follows assuming that the derived preconditions P_1 and P_2 contain no variables. Let $\{S_1, S_2, \ldots, S_m\}$ be the set of skolem function names in P_1 which come from the top level goal in the current deduction. For example if the top level goal is $Q(u, f_1(u)) \Rightarrow R(x, f_2(x), f_3)$ and P_1 is $W(f_1(f_3), g_2(f_3))$ then $\{f_1, f_3\}$ is the set of skolem function names in P_1 which comes from the top level goal. Let $P_1(y_1, \ldots, y_k)$ be the formula resulting from the replacement of each occurence of skolem function S_j by variable y_j in P_1. In the above example $P_1(y_1, y_2)$ denotes $W(y_1, g_2(y_2))$. Function dc is defined as follows.

$$
\begin{aligned}
dc(P_1, \theta_1, P_2, \theta_2) = {} & \text{if } \theta_1 = \text{NIL and } \theta_2 = \text{NIL then NIL} \\
& \text{else if } P_1 = T \text{ or } \theta_2 = \text{NIL then } \theta_1 \\
& \text{else if } P_2 = T \text{ or } \theta_1 = \text{NIL then } \theta_2 \\
& \text{else if } \theta_1 = \{\} \text{ then } \theta_2 \\
& \text{else } \{h_x(S_1, S_2, \ldots, S_m)/x \mid t/x \in \theta_1 \text{ or } t/x \in \theta_2\}
\end{aligned}
$$

where

$$h_x(y_1, \ldots, y_m) = \text{if } P_1(y_1, \ldots, y_m) \text{ then } x\theta_1 \text{ else } x\theta_2.$$

Loosely speaking, the disjunctive composition of P_1, θ_1, P_2, and θ_2 behaves like θ_1 when P_1 holds and behaves like θ_2 otherwise. Some examples:

$$dc(a_0 > 3,\ \{f_1(a_0)/x\},\ T,\ \{a_0/x\}) = \{a_0/x\}$$

$$dc(f_1 > f_2(f_1),\ \{f_1/z, f_2(f_3)/x\},\ f_1 < f_2(f_3),\ \{f_2(f_1)/z, f_3/x\})$$
$$= \{h_z(f_1, f_2, f_3)/z,\ h_x(f_1, f_2, f_3)/x\}$$

where

$$h_z(y_1, y_2, y_3) = \text{if } y_1 > y_2 \text{ then } y_1 \text{ else } y_2$$

$$h_x(y_1, y_2, y_3) = \text{if } y_1 > y_2 \text{ then } y_2 \text{ else } y_3$$

A complete deduction involving a disjunctive composition is given in section 2.5.

Rules which reduce a goal statement to one subgoal are notated

$$
\begin{array}{lll}
\langle P_0 \rangle & \begin{matrix} A_0 \\ H_0 \end{matrix} & \theta_0 \\
 & \Big| & \\
\langle P_1 \rangle & \begin{matrix} A_1 \\ H_1 \end{matrix} & \theta_1
\end{array}
$$

Occasionally, as in the application of known theorems which are implications, the relation between goal and subgoals is not one of equivalence but implication. Rules of this kind are notated

$$\begin{array}{ccc} \langle P_0 \rangle & \underset{H_0}{A_0} & \theta_0 \\ & \uparrow & \\ \langle P_1 \rangle & \underset{H_1}{A_1} & \theta_1 \end{array}$$

which asserts that if P_1 is a precondition of $H_1\theta_1 \Rightarrow A_1\theta_1$ then P_0 is a precondition of $H_0\theta_0 \Rightarrow A_0\theta$. For rules of this kind we cannot assert that P_0 is a weakest precondition of $H_0\theta_0 \Rightarrow A_0\theta_0$ even if P_1 is known to be a weakest precondition of $H_1\theta_1 \Rightarrow A_1\theta_1$.

The following rules are for the most part extensions of typical goal reduction rules [2,5,8].

R1. Reduction of Conjunctive Goals

$$\begin{array}{c} \langle P_1 \wedge P_2 \rangle \quad \underset{H}{A \wedge B} \quad uc(\theta_1,\theta_2) \\ \wedge \\ \langle P_1 \rangle \quad \underset{H}{A} \quad \theta_1 \qquad\qquad \langle P_2 \rangle \quad \underset{H}{B} \quad \theta_2 \end{array}$$

R2. Reduction of Disjunctive Goals

$$\begin{array}{c} \langle P_1 \vee P_2 \rangle \quad \underset{H}{A \vee B} \quad dc(P_1,\theta_1,P_2,\theta_2) \\ \vee \\ \langle P_1 \rangle \quad \underset{H}{A} \quad \theta_1 \qquad\qquad \langle P_2 \rangle \quad \underset{H}{B} \quad \theta_2 \end{array}$$

R3. Reduction of Conjunctive Hypotheses

$$\begin{array}{c} \langle P \rangle \quad A \quad \theta \\ \{B \wedge C\} \cup H \\ | \\ \langle P \rangle \quad A \quad \theta \\ \{B, C\} \cup H \end{array} \qquad\qquad \begin{array}{ccc} & \langle P_1 \vee P_2 \rangle \quad A \quad dc(P_1, \theta_1, P_2, \theta_2) & \\ & \{B \wedge C\} \cup H & \\ & \vee & \\ \langle P_1 \rangle \quad A \quad \theta_1 & & \langle P_2 \rangle \quad A \quad \theta_2 \\ \{B\} \cup H & & \{C\} \cup H \end{array}$$

R4. Reduction of Disjunctive Hypotheses

$$\begin{array}{ccc} & \langle P_1 \wedge P_2 \rangle \quad A \quad uc(\theta_1, \theta_2) & \\ & \{B \vee C\} \cup H & \\ & \wedge & \\ \langle P_1 \rangle \quad A \quad \theta_1 & & \langle P_2 \rangle \quad A \quad \theta_2 \\ \{B\} \cup H & & \{C\} \cup H \end{array}$$

R5. Application of an Equivalence Formula

$$\begin{array}{c} \langle P \rangle \quad A \quad \theta\theta_1 \\ H \\ | \\ \langle P \rangle \quad B\theta \quad \theta_1 \\ H \end{array} \qquad \begin{array}{r} C \equiv B \text{ is a known theorem of TT} \\ \text{or an hypothesis in H and } \theta \text{ unifies } \{A, B\} \end{array}$$

R6. Application of an Implicational Formula

$$\begin{array}{c} \langle P \rangle \quad A \quad \theta\theta_1 \\ H \\ \uparrow \\ \langle P \rangle \quad D \quad \theta_1 \\ H\theta \end{array} \qquad \begin{array}{r} \text{if } C \Rightarrow B \text{ is a known theorem of TT or hypothesis in H,} \\ \text{and D is } C\theta \text{ where } \theta \text{ unifies } \{A, B\} \\ \text{or D is } {\sim}B\theta \text{ where } \theta \text{ unifies } \{A, {\sim}C\} \text{ or } \{{\sim}A, C\} \end{array}$$

R7. Forward Inference from an Hypothesis

$$\begin{array}{l} \langle P\rangle\ \underset{\{B\}\cup H}{A}\ \theta \\ \quad\Big| \quad \text{if } D \Rightarrow E \text{ or } D \equiv E \text{ is a known theorem of TT} \\ \quad\Big| \quad\quad \text{or hypothesis in H and } \theta_1 \text{ unifies } \{B,D\} \\ \langle P\rangle\ \underset{\{B,E\theta_1\}\cup H}{A}\ \theta \end{array}$$

R8. Goal/Hypothesis Duality rules

R8a

$$\begin{array}{c} \langle P\rangle\quad \underset{H}{\sim B \vee A}\quad \theta \\ \Big| \\ \langle P\rangle\ \underset{\{B\}\cup H}{A}\ \theta \end{array}$$

R8b

$$\begin{array}{c} \langle P\rangle\ \underset{\{B\}\cup H}{A}\ \theta \\ \Big| \\ \langle P\rangle\quad \underset{H}{\sim B \vee A}\quad \theta \end{array}$$

R9. Substitution of Equal Terms

$$\begin{array}{l} \langle P\rangle\quad \underset{H}{A(r)}\quad \theta \\ \quad\quad\Big| \quad\quad \text{if } r=s \text{ is an hypothesis in H} \\ \quad\quad\Big| \quad\quad\quad \text{or a known theorem of TT} \\ \langle P\rangle\quad \underset{H}{A(s)}\quad \theta \end{array}$$

R10. Conditional Equality Substitution

$$\begin{array}{c} \langle P_1 \wedge P_2\rangle\quad \underset{H}{A(r)}\quad uc(\theta_1,\theta_2) \\ \wedge \\ \langle P_1\rangle\quad \underset{H\theta_0}{A(s_2)\theta_0}\quad \theta_1 \qquad\qquad \langle P_2\rangle\quad \underset{H\theta_0}{B\theta_0}\quad \theta_2 \end{array}$$

if $B \Rightarrow s_1 = s_2$ is an hypothesis or a known theorem and θ_0 unifies $\{r,s_1\}$

2.3 Primitive Goals

There are several types of primitive goal statements in our system. Each are described by notations of the form $\langle P\rangle\ \underset{H}{A}\ \theta$ which assert that P is a precondition of $H\theta \Rightarrow A\theta$ if the associated condition holds.

P1. $\langle T \rangle \; \frac{A}{H} \; \theta$ if θ unifies $\{A,B\}$ where B is a known theorem of TT or $B \in H$

P2. $\langle F \rangle \; \frac{A}{H} \; \mathrm{NIL}$ if θ unifies $\{A,\sim B\}$ or $\{\sim A,B\}$, where B is a known theorem of TT

In addition to P1 and P2 any goal with a null hypothesis may be taken as primitive:

P3. $\langle A' \rangle \; \frac{A}{\{\}} \; \{\}$ if A has the form $\bigvee_{i=1}^{k} A_i$ and A' has the form $\bigvee_{j=1}^{m} A_{i_j}$ where $\{A_{i_j}\}_{j=1,m} \subseteq \{A_i\}_{i=1,k}$ and for each j, $1 \le j \le m$, A_{i_j} depends on the variables $x_1,x_2,\ldots,x_n$ only when we seek a $x_1,x_2,\ldots,x_n$-precondition.

Primitive goals of type P1 and P2 yield weakest preconditions but in general primitive goals of type P3 do not. Note that any goal statement can be converted to an equivalent goal with a null hypothesis by repeated applications of rule R8b.

2.4 The Deduction Process

The derivation of a precondition of goal statement $\frac{A}{H}$ can be described by a two stage process. In the first phase rules are repeatedly applied to goals reducing them to subgoals and generating a goal tree. Rules are not applied to a goal satisfying the primitive goal tests P1 and P2 or if the goal has been specially converted to satisfy P3. If for some reason, such as limits on computational resource, it is desired to terminate the reduction process before all subgoals have been reduced to primitive goals of type P1 or P2, then any subgoals waiting for rule application can be converted to a primitive goal of type P3. The result of this reduction process is a goal tree with primitive goals as leaf nodes.

The second phase involves the bottom-up composition of preconditions and substitutions. Initially each primitive goal yields a precondition and a substitution. Subsequently whenever a precondition or substitution has been found for each subgoal of a goal $\frac{A}{H}$ then a precondition and substitution is composed for $\frac{A}{H}$ according to the reduction/composition rule employed. Each newly composed precondition is then run through a simplification process to be described later.

Usually several rules can be applied to a given goal and each rule will generate a precondition. In an computer implementation of this system we would make use of a complexity measuring function and select that precondition of least complexity among the alternatives.

2.5 An Example

As an example of the use of this system suppose that we wish to show that

$$\forall i_0 \forall i_1 \exists i_2 [(i_0 < i_1 \wedge i_2 = 0) \vee (i_0 \geq i_1 \wedge i_2 = 1)] \quad (1)$$

is valid in TT where i_0, i_1, i_2 are variables over $\mathbb{N}$ (natural numbers). We do so by trying to derive T as a $i_0 i_1 i_2$-precondition of (1). The goal after preparation is:

$$(r_0 < r_1 \wedge i_2 = 0) \vee (r_0 \geq r_1 \wedge i_2 = 1)$$

where r_0 and r_1 are skolem constants of type $\mathbb{N}$. The derivation is depicted below in figure 1. Initially (1) is reduced via rule R2 to two subgoals then each of these subgoals are reduced via rule R1 to two other subgoals. Subgoals $i_2 = 0$ and $i_2 = 1$ match axiom $i = i$ (theorem n0 in the Appendix) with substitution $\{0/i_2\}$ and $\{1/i_2\}$ respectively and thus are primitive goals of type P1. Suppose that goals $r_0 < r_1$ and $r_0 \geq r_1$ are taken as primitive goals of type P3. The composition phase now begins. Subgoals $r_0 < r_1 \wedge i_2 = 0$ and $r_0 \geq r_1 \wedge i_2 = 1$ yield preconditions $(T \wedge r_0 < r_1)$ and $(T \wedge r_0 \geq r_1)$ respectively. A simplification process reduces these preconditions to $r_0 < r_1$ and $r_0 \geq r_1$ respectively. The composed substitutions for the immediate subgoals of (1) are just the unifying compositions $uc(\{0/i_2\},\{\ \}) = \{0/i_2\}$, and $uc(\{1/i_2\},\{\ \}) = \{1/i_2\}$ respectively. The derived precondition of goal (1) is $(r_0 < r_1 \vee r_0 \geq r_1)$ which simplifies (via theorem n4) to T. The composed substitution is the disjunctive composition $\{f_{i_2}(r_0, r_1)/i_2\}$ where

$$f_{i_2}(j_1, j_2) = \text{if } j_1 < j_2 \text{ then } 0 \text{ else } 1.$$

The derivation shows that T is a precondition of

$$(r_0<r_1 \wedge f_{i_2}(r_0,r_1)=0) \vee (r_0\geq r_1 \wedge f_{i_2}(r_0,r_1)=1)$$

i.e., that our original goal is valid. Furthermore we have obtained a substitution term for the one existentially quantified variable in (1). After requantifying we obtain the valid formula:

$$\forall i_0 \forall i_1 [(i_0<i_1 \wedge f_{i_2}(i_0,i_1)=0) \vee (i_0\geq i_1 \wedge f_{i_2}(i_0,i_1)=1)].$$

In this example and all that follow we annotate the arcs with the name of the rule and theorem used and note the primitive goal type of each leaf node. Also in this example we write the simplified form of the composed precondition P immediately under P. Hereafter in examples we will simply omit the composed precondition in favor of its simplified form. Also we omit substitutions when they are inessential to an understanding of a derivation.

2.6 Formula Simplification

Any deductive mechanism needs a means to simplify formulas which are generated during the deductive process. Simplification can be usefully viewed as the task of finding a weakest precondition (in all variables) of formula A. The search for a simple weakest precondition is kept short by using only a few of the known theorems of TT. The strategy followed in the examples is to repeat

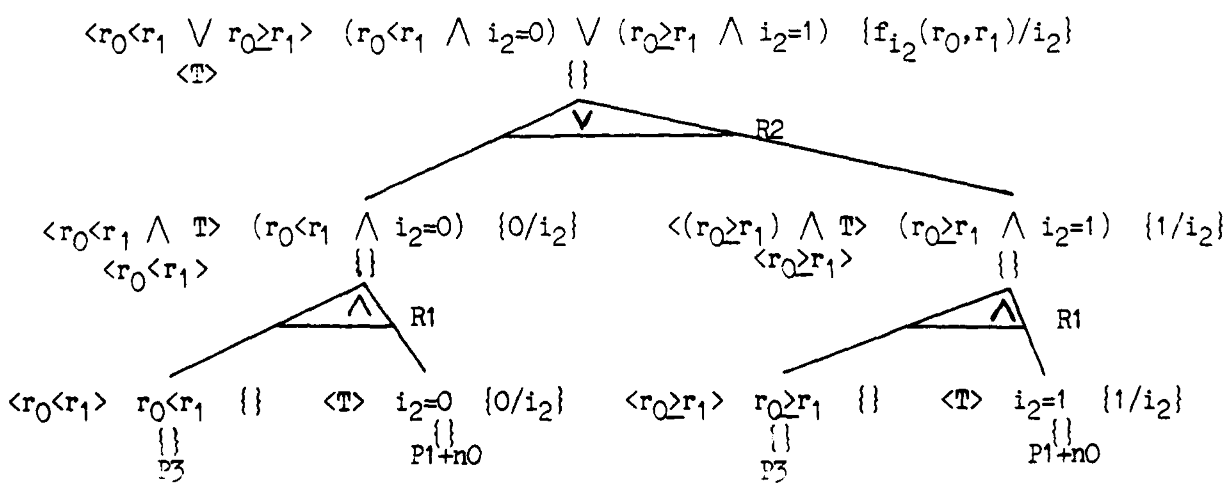


Figure 1.

the following sequence of rule applications until the goal has been reduced to literals:

a) simplify the goal as much as possible using known equivalence theorems of TT,

b) multiply subexpressions out using p9 and p10 (DeMorgan's Laws),

c) break the result of (b) down to subexpressions using R1 or R2.

The multiplication step allows us to mix preconditions which were returned from different branches of the goal tree.

A precondition generating mechanism used for simplification purposes must be carefully controlled in order to avoid infinite regress. One way around this problem is to prohibit simplification of preconditions generated during the simplification process. Instead we check whether the final derived precondition P is simpler than the initial goal formula A. If not then A is returned otherwise we attempt to simplify P. If our complexity measuring function ranges over a well-founded set then this simplification process will terminate.

Suppose that we need to simplify the expression

$$(i>j \vee i=0) \wedge (i<j \vee j=0) \tag{2}$$

where i and j vary over $\mathbb{N}$. The derivation in figure 2a yields

$$(i>0 \wedge j=0) \vee i=0$$

as a weakest precondition (i.e. equivalent form) of (2). The derivation in figure 2b yields

$$(i=0 \vee j=0) \tag{3}$$

as a weakest precondition. The result is that (2) has been simplified to (3).

3. The Use of Derived Preconditions in Program Synthesis

In this section we show how derived preconditions can play a central role in the design of algorithms [9,10]. Many of the key steps in the design process involve finding a precondition of a formula constructed by instantiation of a formula schema with functions, predicates and types from the specification and the partially designed algorithm. The resulting derived precondition is used to either strengthen or complete some aspect of the target algorithm.

Initially a user supplies a complete formal specification of a problem which he desires to solve. The specification consists of a naming of the input

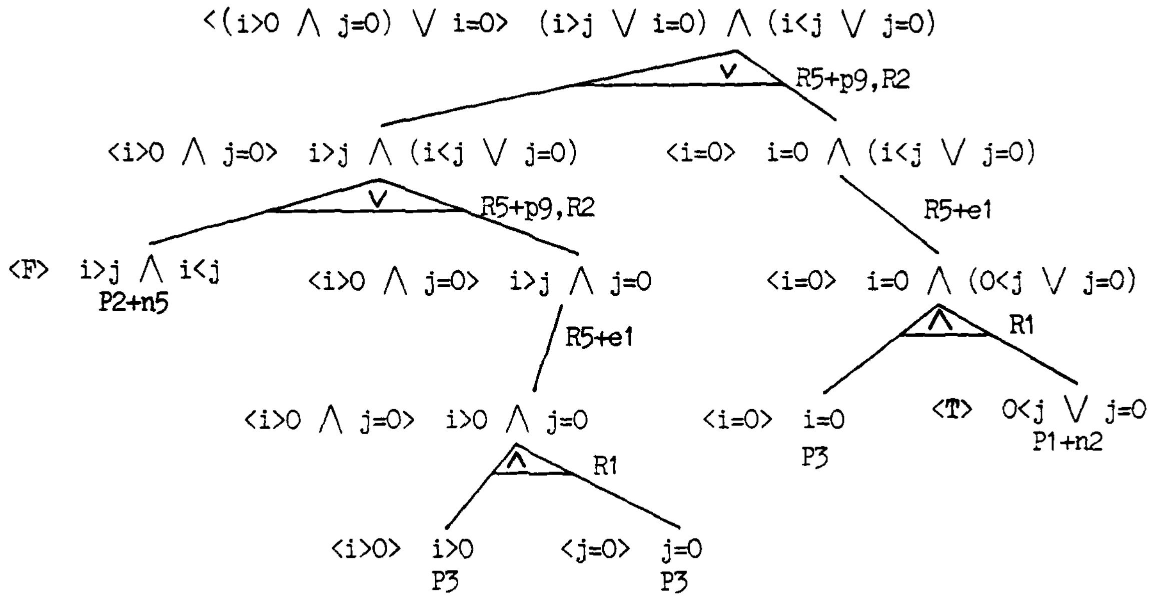


Figure 2a. First pass at simplifying goal formula (2).

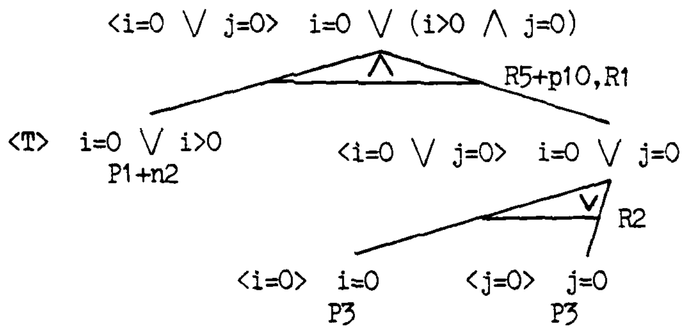


Figure 2b. Second pass: simplifying the result of figure 2a.

and output data types, and two formulas called the input and output conditions. The types, functions and predicates involved in the specification must be part of the language of TT. For example, the problem of sorting a list of natural numbers may be specified as follows:

$$\text{QSORT}(x) = z \text{ such that } \text{ORD}(z) \wedge \text{BAG}(x)=\text{BAG}(z)$$
$$\text{where QSORT: LIST}(\mathbb{N}) \rightarrow \text{LIST}(\mathbb{N}).$$

Here the input and output types are LIST($\mathbb{N}$) (lists of natural numbers). There is no input condition (except the implicit condition of the input type) and the output condition is ORD(z) $\wedge$ BAG(x)=BAG(z) where ORD(z) holds iff the list z is in nondecreasing order, and BAG(x)=BAG(z) holds iff the multiset (bag) of elements in x and z is the same.

We will construct a divide and conquer algorithm (quicksort) of the form:

```
QSORT(x) = if
              PRIM(x)  →  QSORT := f(x) []
             ~PRIM(x)  →  (x1,x2) := DECOMPOSE(x);
                          (z1,z2) := (QSORT(x1),QSORT(x2));
                          QSORT := COMPOSE(z1,z2)
           fi
```

where PRIM is a predicate which determines when to terminate recursion, f is a function which provides a solution for primitive inputs, DECOMPOSE and COMPOSE are decomposition and composition functions respectively. In this program schema PRIM, f, DECOMPOSE, and COMPOSE are uninterpreted functions whose value we have to determine. The if-fi construct is Dijkstra's nondeterministic conditional statement [3]. Associated with the algorithm schema is a correctness schema which will be introduced later.

The first step in the synthesis process involves the representation of the users problem by a problem reduction model [10]. This format extends the specification of a problem and restricts the type of algorithms which can be used to solve the problem to one of a small number of algorithms which work by problem reduction. For present purposes the relevant parts of the representation for QSORT are:

a) a relation IDR, called the input decomposition relation, which constrains the way in which input x_0 can be decomposed into objects x_1 and x_2 and serves as a partial output condition on subalgorithm DECOMPOSE in the divide and conquer schema:

$$IDR(x_0,x_1,x_2) \equiv BAG(x_0)=BAG(x_1)\,\mathbf{U}\,BAG(x_2)$$

where $B_1 \mathbf{U} B_2$ denotes the bag-union of bags B_1 and B_2.

b) a relation OCR, called the output composition relation, which constrains the way in which output object z_0 can be formed from objects z_1 and z_2 and serves as a partial output condition on the subalgorithm COMPOSE:

$$OCR(z_0,z_1,z_2) \equiv BAG(z_0)=BAG(z_1) \cup BAG(z_2)$$

c) a well-founded ordering relation $\succ$ on LIST($\mathbb{N}$) is used to ensure that the target program terminates on all inputs:

$$x_0 \succ x_1 \equiv LG(x_0)>LG(x_1)$$

where the function LG(x) returns the length of the list x.

3.1 Checking and Enforcing Compatibility in the Representation

The representation of the user's problem by a problem reduction model is constructed by heuristic means. A formula expressing the mutual compatibility of various parts of the model is constructed and an attempt is made to verify it. If the derived precondition P is T then the parts are compatible otherwise we use P to modify the model to ensure compatibility. For example we want the input decomposition relation IDR to be compatible with the well-founded ordering $\succ$, in the sense that

$$\forall x_0 \forall x_1 \forall x_2 \; [IDR(x_0,x_1,x_2) \Rightarrow x_0 \succ x_1 \wedge x_0 \succ x_2]$$

i.e., if x_0 can decompose into lists x_1 and x_2 then x_1 and x_2 must both be smaller than x_0 under the $\succ$ relation. After substituting in the form of IDR and the well-founded ordering for the QSORT example, and preparing the formula we obtain the goal:

$$BAG(a_0)=BAG(a_1) \cup BAG(a_2) \Rightarrow LG(a_0)>LG(a_1) \wedge LG(a_0)>LG(a_2) \qquad (4)$$

where a_0, a_1, and a_2 are skolem constants for the (universally quantified) variables x_0, x_1, x_2. The derivation of a $x_0x_1x_2$-precondition of (4) is given in figure 3. The resulting precondition is

$$BAG(x_0)=BAG(x_1) \cup BAG(x_2) \Rightarrow LG(x_1)>0 \wedge LG(x_2)>0$$

which means that IDR is not strong enough to imply the consequent of the original goal. From the definition of preconditions it follows that the conjunction of IDR and the derived precondition will in fact imply the consequent of (4).

$$\langle Q\rangle \quad BAG(x_0)=BAG(x_1)\cup BAG(x_2) \Rightarrow LG(x_0)>LG(x_1) \wedge LG(x_0)>LG(x_2)$$

R8a

$$\langle Q\rangle \quad LG(x_0)>LG(x_1) \wedge LG(x_0)>LG(x_2)$$
$$\{BAG(x_0)=BAG(x_1)\cup BAG(x_2)\}$$

$\wedge$ R1

$$\langle Q_1\rangle \quad LG(x_0)>LG(x_1)$$
$$\{BAG(x_0)=BAG(x_1)\cup BAG(x_2)\}$$

R7+1b2

$$\langle Q_1\rangle \quad LG(x_0)>LG(x_1)$$
$$H$$

R9

$$\langle Q_1\rangle \quad LG(x_1)+LG(x_2)>LG(x_1)$$
$$H$$

R5+n6

$$\langle Q_1\rangle \quad LG(x_2)>0$$
$$H$$

R8b

$$\langle Q_1\rangle \quad Q_1$$
$$\{\}$$
$$P3$$

$$\langle Q_2\rangle \quad LG(x_0)>LG(x_2)$$
$$\{BAG(x_0)=BAG(x_1)\cup BAG(x_2)\}$$

R7+1b2

$$\langle Q_2\rangle \quad LG(x_0)>LG(x_2)$$
$$H$$

R9

$$\langle Q_2\rangle \quad LG(x_1)+LG(x_2)>LG(x_2)$$
$$H$$

R5+n6

$$\langle Q_2\rangle \quad LG(x_1)>0$$
$$H$$

R8b

$$\langle Q_2\rangle \quad Q_2$$
$$\{\}$$
$$P3$$

where

Q_1 is $BAG(x_0)=BAG(x_1)\cup BAG(x_2) \Rightarrow LG(x_2)>0$

Q_2 is $BAG(x_0)=BAG(x_1)\cup BAG(x_1) \Rightarrow LG(x_2)>0$

Q is $BAG(x_0)=BAG(x_1)\cup BAG(x_2) \Rightarrow (LG(x_2)>0 \wedge LG(x_1)>0)$

$H = \{BAG(x_0)=BAG(x_1)\cup BAG(x_2), LG(x_0)=LG(x_1)+LG(x_2)\}$

Figure 3. Checking Compatibility of IDR and $\succ$

Thus we can form a new strengthened input decomposition relation IDR' where

$$IDR'(x_0,x_1,x_2) \equiv IDR(x_0,x_1,x_2) \wedge [BAG(x_0)=BAG(x_1) \cup BAG(x_2) \Rightarrow LG(x_1)>0 \wedge LG(x_2)>0]$$

The derivation in figure 3 guarantees that IDR' is compatible with the well-founded ordering. After simplifying IDR' we have

$$IDR'(x_0,x_1,x_2) \equiv BAG(x_0)=BAG(x_1) \cup BAG(x_2) \wedge LG(x_1)>0 \wedge LG(x_2)>0.$$

3.2 Reducing a Quantified Predicate to a Target Language Expression

The predicate PRIM(x) in the divide and conquer schema is intended to distinguish nondecomposable from decomposable inputs. In the QSORT example it is sufficient for ~PRIM(x_0) to be a x_0-precondition of

$$\forall x_0 \exists x_1 \exists x_2\ IDR'(x_0,x_1,x_2)$$

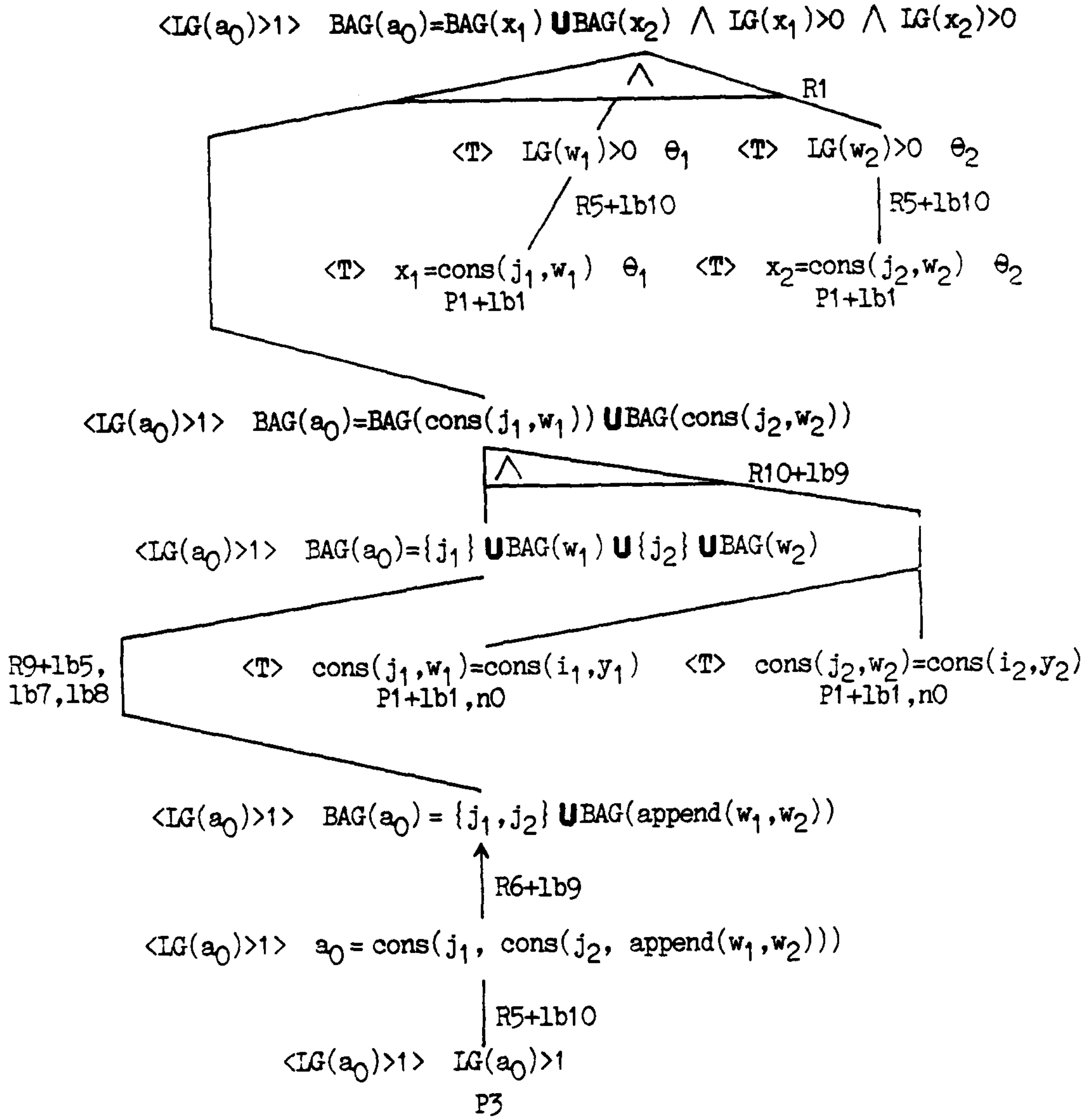
i.e. a list is decomposable only if there are lists into which it can decompose. The deduction in figure 4 yields the precondition $LG(a_0)>1$ and after some simple manipulations $LG(x)\leq 1$ and $LG(x)>1$ can be substituted for PRIM(x) and ~PRIM(x) respectively in QSORT. One additional mechanism is needed to correctly handle this example. The reduction/composition rule R1 treats each subgoal independently and combines the returned substitutions into their unifying composition. This treatment does not work well when the subgoals have common variables. Most theorem proving systems allow substitutions in one subgoal to be applied to the other (since different substitutions may be found independently for the same variable) and we follow this practice here.

3.3 Simple Code Generation through Substitution of a Term for an Output Variable.

With the PRIM predicate in hand the synthesis process can proceed to the task of finding a target language expression to handle primitive inputs in the quicksort algorithm. A correctness formula for the primitive branch of the quicksort algorithm is:

$$\forall x \exists z[LG(x)\leq 1 \Rightarrow ORD(z) \wedge PERM(x,z)].$$

The deduction in figure 5 shows that T is a xz-precondition of this formula thus proving its validity in TT. The substitution gives us a value for z for any x,



where $\theta_1 = \{cons(j_1,w_1)/x_1\}$ and $\theta_2 = \{cons(j_2,w_2)/x_2\}$

Figure 4. Generating a target language expression for ~PRIM

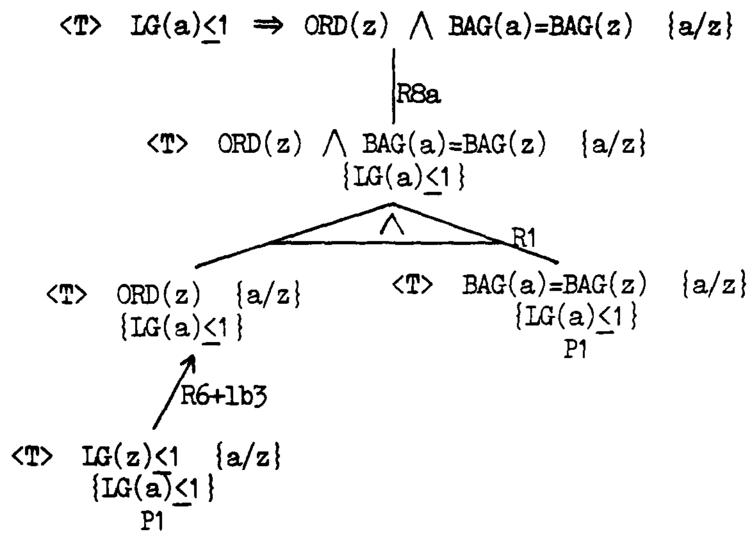


Figure 5. Finding a target language term

namely x itself. Thus the primitive branch of our quicksort is completed since x is the desired output value. The target algorithm now has the form

```
QSORT(x) = if
              LG(x)≤1  →  QSORT := x []
              LG(x)>1  →  . . .
           fi
```

3.4 Completion of the Partial Specification of a Subalgorithm

The next step in the synthesis provides our final example and completes the construction of the top level algorithm for QSORT. The nonprimitive branch of QSORT has two uninterpreted functions COMPOSE and DECOMPOSE which have partial specifications based on OCR and IDR respectively. We look for a known target language function satisfying either partial specification and find that the function APPEND, which appends one list onto the end of another, satisfies the (partial) specification for COMPOSE. The algorithm schema then becomes:

$$
\begin{aligned}
\mathrm{QSORT}(x) = \mathbf{if}\ & \\
& \mathrm{LG}(x) \leq 1 \ \rightarrow\ \mathrm{QSORT} := x\ \square \\
& \mathrm{LG}(x) > 1 \ \rightarrow\ (x_1, x_2) := \mathrm{DECOMPOSE}(x); \\
& \qquad\qquad (z_1, z_2) := (\mathrm{QSORT}(x_1), \mathrm{QSORT}(x_2)); \\
& \qquad\qquad \mathrm{QSORT} := \mathrm{APPEND}(z_1, z_2) \\
\mathbf{fi}\ &
\end{aligned}
$$

where subalgorithm DECOMPOSE remains to be synthesized and has partial specification

$$
\begin{aligned}
\mathrm{DECOMPOSE}(x) = (x_1, x_2) \text{ such that } & [\mathrm{LG}(x) > 1 \Rightarrow (\mathrm{BAG}(x) = \mathrm{BAG}(x_1) \cup \mathrm{BAG}(x_2) \wedge \\
& \mathrm{LG}(x_1) > 0 \wedge \mathrm{LG}(x_2) > 0)] \\
& \text{where DECOMPOSE: } \mathrm{LIST}(\mathbb{N}) \rightarrow \mathrm{LIST}(\mathbb{N})^2.
\end{aligned}
$$

The concern now is to find any additional output conditions needed by DECOMPOSE in order to make QSORT satisfy its formal specifications. A sufficient condition for the total correctness of QSORT [10] is:

$$
\begin{aligned}
\forall x_0 \forall x_1 \forall x_2 \forall z_0 \forall z_1 \forall z_2\ [[\ & \mathrm{BAG}(x_0) = \mathrm{BAG}(x_1) \cup \mathrm{BAG}(x_2) \wedge \\
& \mathrm{LG}(x_1) > 0 \wedge \mathrm{LG}(x_2) > 0 \wedge \\
& \mathrm{BAG}(x_1) = \mathrm{BAG}(z_1) \wedge \mathrm{ORD}(z_1) \wedge \\
& \mathrm{BAG}(x_2) = \mathrm{BAG}(z_2) \wedge \mathrm{ORD}(z_2) \wedge \\
& z_0 = \mathrm{APPEND}(z_1, z_2)] \Rightarrow (\mathrm{BAG}(x_0) = \mathrm{BAG}(z_0) \wedge \mathrm{ORD}(z_0))]
\end{aligned}
\tag{6}
$$

If (6) is not valid it is because the specification of DECOMPOSE is too weak. We seek therefore a $x_0 x_1 x_2$-precondition of (6) and add it to the output specification of DECOMPOSE. Preparing (6) results in the substitution of skolem constants $a_0, b_1, b_2, c_0, c_1, c_2$ for $x_0, x_1, x_2, z_0, z_1, z_2$ respectively. Let H denote the set of conjuncts in the antecedent of the prepared correctness formula and A the consequent. An expression of the form P(ALL(B)) will be used to abbreviate $\forall x \in B\ P(x)$ where B is a bag variable. The derivations given in figures 6a and 6b yield

$$\mathrm{ALL}(\mathrm{BAG}(x_1)) \leq \mathrm{ALL}(\mathrm{BAG}(x_2)).$$

Strengthening DECOMPOSE with this precondition we obtain the complete specification

<T> $\mathrm{BAG}(a_0)=\mathrm{BAG}(c_0)$

$\{c_0=\mathrm{APPEND}(c_1,c_2)\}\cup H$

| R9

<T> $\mathrm{BAG}(a_0)=\mathrm{BAG}(\mathrm{APPEND}(c_1,c_2))$

H

| R9+1b5

<T> $\mathrm{BAG}(a_0)=\mathrm{BAG}(c_1)\cup\mathrm{BAG}(c_2)$

$\{\mathrm{BAG}(b_1)=\mathrm{BAG}(c_1),\mathrm{BAG}(b_2)=\mathrm{BAG}(c_2)\}\cup H$

| R9

<T> $\mathrm{BAG}(a_0)=\mathrm{BAG}(b_1)\cup\mathrm{BAG}(b_2)$

$\{\mathrm{BAG}(a_0)=\mathrm{BAG}(b_1)\cup\mathrm{BAG}(b_2)\}$

P1

Figure 6a. Nonprimitive branch of QSORT

<P> $\mathrm{ORD}(c_0)$

$\{c_0=\mathrm{APPEND}(c_1,c_2)\}\cup H$

| R9

<P> $\mathrm{ORD}(\mathrm{APPEND}(c_1,c_2))$

H

$\wedge$ R5+1b4,R1

<T> $\mathrm{ORD}(c_1)$
$\{\mathrm{ORD}(c_1)\}\cup H$
P1

<T> $\mathrm{ORD}(c_2)$
$\{\mathrm{ORD}(c_2)\}\cup H$
P1

<P> $\mathrm{ALL}(\mathrm{BAG}(c_1))\leq\mathrm{ALL}(\mathrm{BAG}(c_2))$
$\{\mathrm{BAG}(c_1)=\mathrm{BAG}(b_1),\ \mathrm{BAG}(c_2)=\mathrm{BAG}(b_2)\}\cup H$

/ R9

<P> $\mathrm{ALL}(\mathrm{BAG}(b_1))\leq\mathrm{ALL}(\mathrm{BAG}(b_2))$

H

/ R8b

<P> $\sim H \vee P$

P3

where P is $\mathrm{ALL}(\mathrm{BAG}(b_1))\leq\mathrm{ALL}(\mathrm{BAG}(b_2))$

Figure 6b. Completing the specification of DECOMPOSE

$$\text{DECOMPOSE}(x) = (x_1,x_2) \text{ such that } [\text{LG}(x)>1 \Rightarrow (\text{BAG}(x)=\text{BAG}(x_1) \cup \text{BAG}(x_2) \wedge \text{LG}(x_1)>0 \wedge \text{LG}(x_2)>0 \wedge \text{ALL}(\text{BAG}(x_1))\leq\text{ALL}(\text{BAG}(x_2))]$$
$$\text{where DECOMPOSE: LIST}(\mathbb{N}) \rightarrow \text{LIST}(\mathbb{N})^2.$$

The synthesis process is then recursively invoked to design an algorithm meeting these specifications.

The synthesis system from which we've drawn the examples is an attempt to obtain increased synthesis performance by 1) dividing the synthesis task into a number of relatively small deductive tasks, and 2) using large amounts of knowledge about programming. The system makes use of two types of programming knowledge: 1) control strategy knowledge encoded by program schemas (such as the schema for divide and conquer used above) and their associated correctness schemas, and 2) data structure knowledge represented in part by the known theorems of TT. Other recent deductive approaches to program synthesis [1,4,6] also make use of data structure knowledge, but have different approaches to representing control knowledge and tend to construct programs on the basis of a single large deductive task.

4. Conclusion

In this paper we have defined a new deductive problem, that of finding a precondition of a given formula, and presented a formal system within which preconditions can be derived. We have tried to convey a sense of the flexibility and usefulness of such a system through a number of examples drawn from the domain of program synthesis. We are currently implementing a system based on the one described here and hope to report on such issues as formula complexity measures and control, which we have largely ignored here, in a future paper.

APPENDIX

Listed below are the known theorems used in the examples of this paper. It is important that these assertions are expressed in their strongest form (i.e., as equivalences rather than implications) whenever possible, so that it can be determined whether a weakest precondition has been derived or not. Often a theorem is used in one direction only although it may be stated as an equivalence.

Propositional theorems

p1. $A \vee \sim A$

p2. $\sim(A \wedge \sim A)$

p3. $T \wedge A \equiv A$

p4. $T \vee A \equiv T$

p5. $F \wedge A \equiv F$

p6. $F \vee A \equiv A$

p7. $\sim(A \wedge B) \equiv \sim A \vee \sim B$

p8. $\sim(A \vee B) \equiv \sim A \wedge \sim B$

p9. $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

p10. $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

p11. $(A \Rightarrow B) \equiv (\sim A \vee B)$

p12. $A \vee (A \wedge B) \equiv A$

p12. $A \wedge (A \vee B) \equiv A$

Equality theorems

e1. $P(x) \wedge x=y \equiv P(y) \wedge x=y$ where $P(x)$ is a formula depending on term x.

Natural number theorems

Let i,j,k denote variables of type $\mathbb{N}$.

n0. $i=i$

n1. $i \geq 0$

n2. $i=0 \vee i>0$

n3. $i \leq j \vee i \geq j$

n4. $i<j \vee i \geq j$

n5. $\sim(i<j \wedge i>j)$

n6. $i+j>i \equiv j>0$

n7. $\sim(i>k) \equiv i \leq k$

n8. $\sim(i<k) \equiv i \geq k$

n9. $i>k_1 \wedge j>k_2 \Rightarrow i+j>k_1+k_2+1$

List and Bag theorems

Let w_0, w_1, w_2 vary over LIST($\mathbb{N}$), and let B_1, B_2 vary over BAGS($\mathbb{N}$).

lb1. $w_0 = w_0$

lb2. $\mathrm{BAG}(w_0)=\mathrm{BAG}(w_1) \cup \mathrm{BAG}(w_2) \Rightarrow \mathrm{LG}(w_0)=\mathrm{LG}(w_1)+\mathrm{LG}(w_2)$

lb3. $\mathrm{LG}(w_0)\leq 1 \Rightarrow \mathrm{ORD}(w_0)$

lb4. $[\mathrm{ORD}(w_1) \wedge \mathrm{ORD}(w_2) \wedge \mathrm{ALL}(\mathrm{BAG}(w_1))\leq\mathrm{ALL}(\mathrm{BAG}(w_2))] \equiv \mathrm{ORD}(\mathrm{APPEND}(w_1,w_2))$

lb5. $\mathrm{BAG}(\mathrm{APPEND}(w_0,w_1)) = \mathrm{BAG}(w_0) \cup \mathrm{BAG}(w_1)$

lb6. $B_1 = B_1$

lb7. $\{i_1\} \cup \{i_2\} = \{i_1, i_2\}$

lb8. $B_1 \cup B_2 = B_2 \cup B_1$

lb9. $w_1 = \mathrm{cons}(i_0, \mathrm{cons}(i_1, \ldots \mathrm{cons}(i_n, w_2)\ldots)) \Rightarrow \mathrm{BAG}(w_1) = \{i_0, i_1, \ldots, i_n\} \cup \mathrm{BAG}(w_2)$

lb10. $w_0 = \mathrm{cons}(i_0, \mathrm{cons}(i_1, \ldots \mathrm{cons}(i_n, w_1)\ldots)) \equiv \mathrm{LG}(w_0)>n$

REFERENCES

1. Bibel, W. Syntax-directed, Semantics-Supported Program Synthesis, Art. Intell. 14(3), 1980, pp 243-262.

2. Bledsoe, W. Nonresolution theorem proving, Art. Intell. 9(1), 1977, pp 1-35.

3. Dijkstra, E.W. A Discipline of Programming, Prentice-Hall Inc., Englewood Cliffs, NJ, 1976.

4. Guiho, G., and Gresse, C., Program Synthesis from Incomplete Specifications, Proc. Fifth Conf. on Automated Deduction, Lecture Notes in Computer Science, Eds. W. Bibel and R. Kowalski, Springer-Verlag, Berlin, 1980.

5. Loveland, D.W. (1978), Automated Theorem Proving: A Logical Basis, North Holland Pub. Co., New York, 1978.

6. Manna, Z., and Waldinger, R. (1980), A Deductive Approach to Program Synthesis, ACM Trans. on Prog. Lang. 2(1), Jan. 1980, pp 90-121.

7. Nilsson, N. Principles of Artificial Intelligence, Tioga Pub. Co., Palo Alto, CA, 1980.

8. Reiter, R. (1976), A Semantically Guided Deductive System for Automatic Theorem Proving, IEEE Trans. on Computers C-25(4), 1976, pp 328-334.

9. Smith, D.R. A Design for an Automatic Programming System, Proc. of IJCAI-7, Vancouver, B.C, Canada, 1981, pp 1024-1027.

10. Smith, D.R. The Synthesis of Divide and Conquer Algorithms, Forthcoming technical report.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Marvin Denicoff Director, ONR Information Science Program Office of Naval Research Code 437 800 N. Quincy Street Arlington, VA 22217	1
Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, CA 93940	40
Professor Douglas R. Smith, Code 52Sc Department of Computer Science Naval Postgraduate School Monterey, CA 93940	12

DATE

ILME