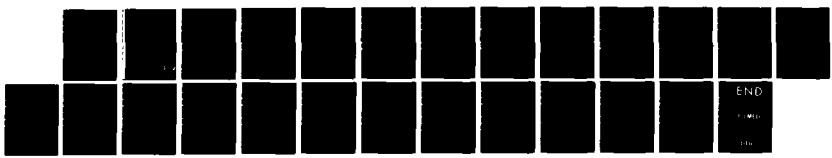
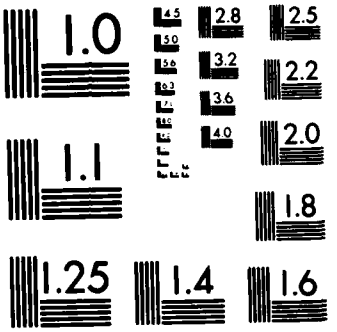
AD-A154 367 THE ORIGIN OF THE BINARY-SEARCH PARADIGM(U) STANFORD UNIV CA DEPT OF COMPUTER SCIENCE Z MANNA ET AL. MAR 85 STAN-CS-85-1044 1/1

UNCLASSIFIED F/G 12/1 NL





March 1985

Report No. STAN-CS-85-1044

The Origin of the Binary-Search Paradigm

by

Zohar Manna

Richard Waldinger

N00014-84-C-0706

Department of Computer Science

Stanford University
Stanford, CA 94305

THE ORIGIN OF THE BINARY-SEARCH PARADIGM

ZOHAR MANNA
Computer Science Department
Stanford University
Stanford, CA 94305

RICHARD WALDINGER
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025

ABSTRACT

In a binary-search algorithm for the computation of a numerical function, the interval in which the desired output is sought is divided in half at each iteration. The paper considers how such algorithms might be derived from their specifications by an automatic program-synthesis system. The derivation of the binary-search concept has been found to be surprisingly straightforward. The programs obtained, though reasonably simple and efficient, are quite different from those that would have been constructed by informal means.

Key Words: program synthesis, theorem proving, binary search, real square root

INTRODUCTION

Some of the most efficient algorithms for the computation of numerical functions rely on the technique of *binary search*; according to this technique, the interval in which the desired output is sought is divided in half at each iteration until it is smaller than a given tolerance.

For example, let us consider the following program for finding a real-number approximation to the square root of a nonnegative real number r. The program sets z to be within a given positive tolerance ϵ less than $\sqrt{r}$.

$z \leftarrow 0$
$v \leftarrow max(r, 1)$
while $\epsilon \leq v$ *do* $v \leftarrow v/2$
    if $[z + v]^2 \leq r$ *then* $z \leftarrow z + v$
return(z)

This is a classical square-root program based on one that appeared in Wensley [59]. The program establishes and maintains the loop invariant that z is within v less than $\sqrt{r}$, i.e., that $\sqrt{r}$ belongs to the half-open interval $[z, z + v)$. At each iteration, the program divides this interval in half and tests whether $\sqrt{r}$ is in the right or left half, adjusting z and v accordingly, until v is smaller than

This research was supported in part by the National Science Foundation under grants MCS-82-14523 and MCS-81-05565, by Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-81-0014, by the Office of Naval Research under Contract N00014-84-C-0706, and by a contract from the International Business Machines Corporation.

the given tolerance ϵ. The program is reasonably efficient; it terminates after $\lceil log_2(max(r,1)/\epsilon) \rceil$ iterations.

Analogous programs provide an efficient means of computing a variety of numerical functions. It is not immediately obvious how such programs can be developed by automatic program-synthesis systems, which derive programs to meet given specifications. Some researchers (e.g., Dershowitz and Manna [77], Smith [85]) have suggested that synthesis systems be provided with several general program schemata, which could be specialized as required to fit particular applications. Binary search would be one of these schemata. The system would be required to discover which schema, if any, is applicable to a new problem.

It may indeed be valuable to provide a synthesis system with general schemata, but this approach leaves open the question of how such schemata are discovered in the first place. To our surprise, we have found that the concept of binary search emerges quite naturally and easily in the derivations of some numerical programs and does not need to be built in. The programs we have obtained in this way are reasonably simple and efficient, but bizarre in appearance and quite different from those we would have constructed by informal means.

The programs have been derived in a deductive framework (Manna and Waldinger [80], [85]) in which the process of constructing a program is regarded as a task of proving a mathematical theorem. According to this approach, the program's specification is phrased as a theorem, the theorem is proved, and a program guaranteed to meet the specification is extracted from the proof. If the specification reflects our intentions correctly, no further verification or testing is required.

In this paper we outline our deductive framework and show the derivation of a numerical program up to the point at which the binary-search concept emerges. We then show several analogous binary-search programs that have been developed by this method. Finally we discuss what these findings indicate about the prospects for automatic program synthesis.

DEDUCTIVE PROGRAM SYNTHESIS

In this section we describe our framework for deductive program synthesis, emphasizing those aspects that are essential for the derivation fragment that appears in this paper. Readers who would like a fuller introduction to this approach are referred to Manna and Waldinger ([80], [85]).

We begin with an outline of the logical concepts we shall need.

LOGICAL PREREQUISITES

The system deals with

- *terms* composed [in the usual way] of constants $a, b, c, \ldots$, variables $u, v, w, \ldots$, function symbols, and the conditional (*if-then-else*) term constructor.
- *atoms* composed of terms, relation (predicate) symbols, including the equality symbol $=$, and the truth symbols *true* and *false*;

- *sentences* composed of atoms and logical connectives.

Sentences are quantifier-free. We sometimes use infix notation for function and relation symbols (for example, $x + a$ or $0 \leq y$). An *expression* is a term or a sentence. An expression is said to be *ground* if it contains no variables. Certain of the symbols are declared to be *primitive*; these are the computable symbols of our programming language.

Let e, s, and t be expressions, where s and t are either both sentences or both terms. If we write e as $e[s]$, then $e[t]$ denotes the result of replacing every occurrence of s in $e[s]$ with t.

We loosely follow the terminology of Robinson [79]. We denote a substitution θ by $\{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \ldots, x_n \leftarrow t_n\}$. For any expression e, the expression $e\theta$ is the result of *applying* θ *to* e, obtained by simultaneously replacing every occurrence of the variable x_i in e with the corresponding term t_i. We shall also say that $e\theta$ is an *instance* of e.

Variables in sentences are given an implicit universal quantification; a sentence is true under a given interpretation if and only if every instance of the sentence is true, and if and only if every ground instance of the sentence (i.e., an instance that contains no variables) is true.

Let e, s, and t be expressions, where s and t are either both sentences or both terms, and let θ be a substitution. If we write e as $e[s]$, then $e\theta[t]$ denotes the result of replacing every occurrence of $s\theta$ in $e\theta$ with t.

We now describe the basic notions of deductive program synthesis.

SPECIFICATIONS AND PROGRAMS

A specification is a statement of the purpose of the desired program, which need give no indication of the method by which that purpose is to be achieved. In this paper we consider only *applicative* (or *functional*) programs, which yield an output but alter no data structures and produce no other side effects. The specifications for these programs have the form

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

In other words, the program f we want to construct is to yield, for a given *input* a, an *output* z satisfying the *output condition* $R[a, z]$, provided that the input a satisfies the *input condition* $P[a]$. In other words, z is to satisfy the *input-output* condition

$$\textit{if } P[a] \quad \textit{then } R[a, z].$$

For example, suppose we want to specify the program *sqrt* to yield a real number z that is within a given tolerance ϵ less than $\sqrt{r}$, the exact square root of a given nonnegative real number r. Then we might write

$$sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that } z^2 \leq r \textit{ and not } [(z + \epsilon)^2 \leq r] \quad \text{where } 0 \leq r \textit{ and } 0 < \epsilon.$$

In other words, we want to find an output z satisfying the output condition

$$z^2 \leq r \ \textit{and not} \ [(z+\epsilon)^2 \leq r],$$

provided that the inputs r and ϵ satisfy the input condition

$$0 \leq r \ \textit{and} \ 0 < \epsilon.$$

The above square-root specification is not a program and does not indicate a particular method for computing the square root; it describes the input-output behavior of many programs, employing different algorithms and perhaps producing different outputs.

The programs we consider are sets of expressions of the form

$$f_i(a) \Leftarrow t_i,$$

where t_i is a *primitive* term, i.e., one expressed entirely in the vocabulary of our programming language. These programs can be mutually recursive; i.e., we regard the function symbols f_i as primitive. In the usual way, such a program indicates a method for computing an output. For the most part, in this paper we shall consider programs consisting of only a single expression $f(a) \Leftarrow t$, which may be recursive.

In a given theory, a program f is said to *satisfy* a specification of the above form if, for any input a satisfying the input condition $P[a]$, the program $f(a)$ terminates and produces an output t satisfying the output condition $R[a, t]$.

DEDUCTIVE TABLEAUS

The fundamental structure of our system, the deductive tableau, is a set of *rows*, each of which must contain a sentence, either an *assertion* or a *goal*; any of these rows may contain an expression, the *output entry*. An example of a tableau follows:

assertions	goals	outputs $f(a)$
$P[a]$		
	$R[a, z]$	z
if $q(u)$ *then* $R[u, 0]$		
	$q(a)$	0

Here u and z are variables and a and 0 are constants.

Under a given interpretation, a tableau is true whenever the following condition holds:

> If all instances of each of the assertions are true,
> then some instance of at least one of the goals is true.

Equivalently, the tableau is true if some instance of at least one of the assertions is false or some instance of at least one of the goals is true. Thus, the above tableau is true if $P[a]$ is false, if

if $q(b)$
then $\mathcal{R}[b, 0]$

is false, if $\mathcal{R}[a, c]$ is true, or if $q(a)$ is true (among other possibilities).

In a given theory, a tableau is said to be *valid* if it is true under any model for the theory.

Under a given interpretation and for a given specification

$f(a) \Leftarrow$ find z such that $\mathcal{R}[a, z]$
where $P[a]$,

a goal is said to have a *suitable* output entry if, whenever an instance of the goal is true, the corresponding instance t' of the output entry will satisfy the *input-output* condition

if $P[a]$
then $\mathcal{R}[a, t']$.

(If the goal has no explicit output entry, then it is said to have a suitable output entry if, whenever an instance of the goal is true, any term t' satisfies the input-output condition.) An assertion is said to have a suitable output entry if, whenever an instance of the assertion is false, the corresponding instance t' of the output entry will satisfy the input-output condition.

Example

In the theory of the real numbers, consider the square-root specification

$sqrt(r, \epsilon) \Leftarrow$ find z such that
$z^2 \leq r$ *and not* $[(z + \epsilon)^2 \leq r]$
where $0 \leq r$ *and* $0 < \epsilon$

and the following tableau:

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $z^2 \leq r$ *and* *not* $[(z + \epsilon)^2 \leq r]$	z
	3. *not* $[\epsilon^2 \leq r]$	0

This tableau is valid in the theory of real numbers, because, under any model of the theory, either the assertion (which has no variables) is false or some instance of one of the two goals is true. (In particular, the instance of goal 2 obtained by taking z to be $\sqrt{r}$ itself is true.)

Under any model for the theory, the output entries of the above tableau are suitable for the square-root specification. In particular, if some instance of goal 2, obtained by replacing z with s, is true, then s will satisfy the input-output condition. That is,

$$\textit{if } 0 \leq r \textit{ and } 0 < \epsilon$$
$$\textit{then } s^2 \leq r \textit{ and not}\left[(s+\epsilon)^2 \leq r\right]$$

is true. Also, if assertion 1, which has no output entry, is false, then any term s satisfies the above condition. ∎

Under a given interpretation I and for a given specification, two tableaus $\mathcal{T}_1$ and $\mathcal{T}_2$ have the same *meaning* if

$\mathcal{T}_1$ is true under I
if and only if
$\mathcal{T}_2$ is true under I

and

the output entries of $\mathcal{T}_1$ are suitable
if and only if
the output entries of $\mathcal{T}_2$ are suitable.

In a given theory and for a given specification, two tableaus are *equivalent* if, under any model I for the theory, the meaning of the two tableaus is the same.

PROPERTIES OF A TABLEAU

Let us consider a particular theory and a particular specification, which will both remain fixed throughout this discussion. We shall use the following properties of a tableau:

- ***Duality Property***

Any tableau is equivalent to the one obtained by removing an assertion and adding its negation as a new goal, with the same output entry. Similarly, any tableau is equivalent to the one obtained by removing a goal and adding its negation as a new assertion. Thus, we could manage with a system that has no goals or a system that has no assertions, but the distinction between assertions and goals does have some intuitive significance.

- ***Renaming Property***

Any tableau is equivalent to the one obtained by systematically renaming the variables of any row. More precisely, we may replace any of the variables of the row with new variables, making sure that all occurrences of the same variable in the row (including those in the output entry) are replaced by the same variable and that distinct variables in the row are replaced by distinct variables. In other words, the variables of a row are dummies that may be renamed freely.

• *Instance Property*

Any tableau is equivalent to the one obtained by introducing as a new row any instance of an existing row. The new row is obtained by replacing all occurrences of certain variables in the existing row (including those in the output entry) with terms. Note that the existing row is not replaced; the new one is simply added.

THE DEDUCTIVE PROCESS

Consider a particular theory and the specification

$f(a) \Leftarrow$ find z such that $\mathcal{R}[a, z]$
where $P[a]$.

We form the initial tableau

assertions	goals	outputs $f(a)$
$P[a]$		
	$\mathcal{R}[a, z]$	z

We may also include in the initial tableau (as an assertion) any valid sentence of the theory.

Note that the output entries of this tableau are suitable: Under any model for the theory, if the initial assertion $P[a]$ is false, then any output satisfies the input-output condition vacuously; and if some instance $\mathcal{R}[a, t]$ of the initial goal is true, the corresponding instance t of the associated output entry satisfies the input-output condition. Furthermore, the valid sentences included as initial assertions cannot be false.

We attempt to show that the above tableau is valid. We proceed by applying deduction rules that add new rows without changing the tableau's meaning in any model for the theory. In other words, under a given model, the tableau is true before application of the rule if and only if it is true afterwards, and the output entries are suitable before if and only if they are suitable afterwards. We describe the deduction rules in the next section.

The deductive process continues until we obtain either of the two rows

	true	t

or

false		t

where the output entry t is primitive, i.e., expressed entirely in the vocabulary of our programming language. (We regard the input constant a and the function symbol f as primitive.) At this point, we derive the program

$f(a) \Leftarrow t$.

We claim that t satisfies the given specification. For, in applying the deduction rules, we have guaranteed that the new output entries are suitable if the earlier output entries are suitable. We have seen that the initial output entries are all suitable; therefore, the final output entry t is also suitable. This means that, under any model, if the final goal *true* is true or the final assertion *false* is false, the corresponding output entry t will satisfy the input-output condition

if $P[a]$
then $\mathcal{R}[a, t]$.

But under any model the truth symbols *true* and *false* are true and false, respectively, and hence t will satisfy the input-output condition. Therefore, the program $f(a) \Leftarrow t$ does satisfy the specification.

THE DEDUCTION RULES

We now introduce the deduction rules of our system, emphasizing those that play a role in the portions of the square-root derivation we present. We begin with the simplest of the rules.

THE TRANSFORMATION RULES

The transformation rules replace subexpressions of an assertion, goal, or output entry with equal or equivalent expressions. For instance, with the transformation rule

$$P \textit{ and } \textit{true} \rightarrow P,$$

we can replace the subsentence $((A \textit{ or } B) \textit{ and } \textit{true})$ with $(A \textit{ or } B)$ in the assertion

$((A \textit{ or } B) \textit{ and } \textit{true}) \textit{ or } D$		0

yielding

$(A \textit{ or } B) \textit{ and } D$		0

With the transformation rule (in the theory of integers or reals)

$$u + u \rightarrow 2u,$$

we can replace a subterm $(a + b) + (a + b)$ with the term $2(a + b)$.

We use an *associative-commutative* matching algorithm (cf. Stickel [81]), so that the associative and commutative properties of operators can be taken into account in applying the transformation rules. Thus, we can use the above rules to replace a subsentence $(\textit{true } \textit{and } B)$ with the sentence B and the subterm $(a + b) + b$ with the term $a + 2b$.

We include a complete set of *true-false* transformation rules, such as

not false → *true*
if P *then false* → *not* P.

Repeated application of these rules can eliminate from a tableau row any occurrence of the truth symbols *true* and *false* as a proper subsentence.

The soundness of the transformation rules is evident, since each produces an expression equivalent or equal (in the theory) to the one to which it is applied.

THE RESOLUTION RULE: GROUND VERSION

The resolution rule corresponds to case analysis in informal reasoning. We first present the *ground version* of the rule, which applies to ground goals. We express it in the following notation:

assertions	goals	outputs $f(a)$
	$\mathcal{F}[P]$	s
	$\mathcal{G}[P]$	t
	$\mathcal{F}[true]$ *and* $\mathcal{G}[false]$	*if* P *then* s *else* t

In other words, suppose our tableau contains two ground goals, $\mathcal{F}$ and $\mathcal{G}$, whose output entries are s and t, respectively. Suppose further that $\mathcal{F}$ and $\mathcal{G}$ have a common subsentence P. Then we may derive and add to our tableau the new goal obtained by replacing all occurrences of P in $\mathcal{F}$ with *true*, replacing all occurrences of P in $\mathcal{G}$ with *false*, and forming the conjunction of the results. The output entry associated with the derived goal is the conditional expression whose test is the common subexpression P and whose *then*-clause and *else*-clause are the output entries s and t for $\mathcal{F}$ and $\mathcal{G}$, respectively. Because the resolution rule always introduces occurrences of the truth symbols *true* and *false* as proper subsentences, we can immediately apply *true-false* transformation rules to the derived goal.

For example, suppose our tableau contains the rows

assertions	goals	outputs $f(a, b)$
	$\boxed{p(a,b)}$ *and* $q(a)$	a
	not (*if* $r(b)$ *then* $\boxed{p(a,b)}$)	b

These goals have a common subsentence $p(a, b)$, indicated by boxes. Therefore we may derive and add to our tableau the new goal

	true and $q(a)$ *and* *not* (*if* $r(b)$ *then false*)	*if* $p(a, b)$ *then* a *else* b

By repeated application of transformation rules, this goal reduces to

	$q(a)$ *and* $r(b)$	*if* $p(a, b)$ *then* a *else* b

If one of the given goals has no output entry, the derived output entry is not a conditional expression; it is simply the output entry of the other given goal. If neither given goal has an output entry, the derived goal has no output entry either. We do not require that the two given goals be distinct; we may apply the rule to a goal and itself.

We have presented the resolution rule as it applies to two goals. According to the duality property of tableaus, however, we may transform an assertion into a goal simply by negating it. Therefore, we can apply the rule to an assertion and a goal, or to two assertions.

The resolution rule may be restricted by a *polarity strategy* (Murray [82]; see also Manna and Waldinger [80]), according to which we need not apply the rule unless some occurrence of P in $\mathcal{F}$ is "positive" and some occurrence of P in $\mathcal{G}$ is "negative". (Here a subsentence of a tableau is regarded as positive or negative if it is within the scope of a respectively even or odd number of negation connectives. Each assertion is considered to be within the scope of an implicit negation; thus, while goals are positive, assertions are negative. The *if*-clause P of a subsentence (*if* P *then* Q) is considered to be within the scope of an additional implicit negation.) This strategy allows us to disregard many useless applications of the rule.

Let us show that the resolution rule is sound; that is, in a given model of the theory and for a given specification, the meaning of the tableau is the same before and after application of the rule. It actually suffices to show that, if the derived goal is true, then at least one of the given goals is true; and if the given output entries are suitable, so is the derived output entry.

Suppose the derived goal ($\mathcal{F}[$*true*$]$ *and* $\mathcal{G}[$*false*$]$) is true. Then both its conjuncts $\mathcal{F}[$*true*$]$ and $\mathcal{G}[$*false*$]$ are true. We distinguish between two cases, depending on whether or not the common subsentence P is true or false. In the case in which P is true, the [ground] goal $\mathcal{F}[P]$ has the same truth-value as the conjunct $\mathcal{F}[$*true*$]$; that is, $\mathcal{F}[P]$ is true. In the case in which P is false, the goal $\mathcal{G}[P]$ has the same truth-value as the conjunct $\mathcal{G}[$*false*$]$; that is, $\mathcal{G}[P]$ is true. In either case, one of the two given goals, $\mathcal{F}[P]$ and $\mathcal{G}[P]$, is true.

Now assume that the given output entries are suitable. To show that the derived output entry is suitable, we suppose that the derived goal is true and establish that the derived output entry satisfies the input-output condition. We have seen that, in the case in which P is true, the given goal $\mathcal{F}[P]$ is true; because its output entry s is suitable, it satisfies the input-output condition. Similarly, in the case in which P is false, the term t satisfies the input-output condition. In either case, therefore, the conditional expression (*if* P *then* s *else* t) satisfies the input-output condition; but this is the derived output entry.

THE RESOLUTION RULE: GENERAL VERSION

We have described the ground version of the resolution rule, which applies to goals with no variables. We now present the general version, which applies to goals with variables. In this case, we can apply a substitution to the goals, as necessary, to create a common subsentence.

assertions	goals	outputs $f(a)$
	$\mathcal{F}[P]$	s
	$\mathcal{G}[\hat{P}]$	t
	$\mathcal{F}\theta[true]$ *and* $\mathcal{G}\theta[false]$	*if* $P\theta$ *then* $s\theta$ *else* $t\theta$

More precisely, suppose our tableau contains goals $\mathcal{F}$ and $\mathcal{G}$, which have no variables in common. (This can be ensured by renaming the variables of the rows as necessary, according to the *renaming* property.) Suppose further that some of the subsentences of $\mathcal{F}$ and some of the subsentences of $\mathcal{G}$ are unifiable, with a most-general unifier θ; let $P\theta$ be the unified subsentence. Then we may derive and add to our tableau the new goal obtained by replacing all occurrences of $P\theta$ in $\mathcal{F}\theta$ with *true*, replacing all occurrences of $P\theta$ in $\mathcal{G}\theta$ with *false*, and forming the conjunction of the results. The associated output entry is a conditional expression whose test is the unified subsentence $P\theta$ and whose *then*-clause and *else*-clause are the corresponding instances $s\theta$ and $t\theta$, respectively, of the given output entries.

In other words, to apply the general version of the rule to $\mathcal{F}$ and $\mathcal{G}$, we apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$. The soundness of the general version follows from the soundness of the ground version. The polarity strategy applies as before. If we wish to apply the rule to an assertion and a goal or to two assertions, we can regard the assertions as goals by negating them, as in the ground case.

For example, suppose our tableau contains the rows

assertions	goals	outputs $f(a, b)$
	$\boxed{y \leq a \text{ and } \textit{not}\,[y + b \leq a]}$ *and* $p(y)$	$g(y)$
if $q(x, v)$ *then* $\boxed{f(x, v) \leq x \text{ and } \textit{not}\,[f(x, v) + v \leq x]}$		

The boxed subsentences are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow a,\ v \leftarrow b,\ y \leftarrow f(a, b)\}.$$

The subsentences are respectively positive and negative, as indicated by the annotation. We may regard the assertion as a goal by negating it. By application of the general version of the resolution rule, we may derive the new row

	$\begin{bmatrix} true\ and \\ p(f(a,\ b)) \end{bmatrix}$ *and* *not* $\begin{bmatrix} if\ q(a,\ b) \\ then\ false \end{bmatrix}$	$g(f(a,\ b))$

By the application of *true-false* transformation rules, this goal reduces to

	$p(f(a,\ b))$ *and* $q(a,\ b)$	$g(f(a,\ b))$

Note that the unifier θ has been applied to all variables in the given rows, including those in the output entry. Because the given assertion has no output entry, the derived output entry is not a conditional expression. This application of the rule is in accordance with the polarity strategy.

The resolution rule and the *true-false* transformation rules have been shown by Murray [82] to constitute a complete system for first-order logic. The polarity strategy maintains this completeness.

We use an associative-commutative unification algorithm (as in Stickel [81]) so that the associative and commutative properties of such operators as addition and conjunction can be taken into account in finding a unifier; thus, $p\big(f(x) + \big(b + g(a)\big)\big)$ can be unified with $p\big(\big(g(y) + f(b)\big) + x\big)$.

We have introduced two additional rules to give special treatment to equality and other important relations (Manna and Waldinger [85]), but these rules play no part in the portion of the derivation to be discussed.

We shall need the induction rule; this we describe next.

THE MATHEMATICAL INDUCTION RULE

The rules presented so far do not allow us to introduce any repetitive construct into the program being derived. The induction rule accounts for the introduction of recursion in the derived program. We employ a single well-founded induction rule, which applies to a variety of theories.

A well-founded relation $\prec_w$ is one that admits no infinite decreasing sequences, i.e., sequences $x_1, x_2, x_3, \ldots$, such that

$$x_1 \succ_w x_2 \ \textit{and}\ x_2 \succ_w x_3 \ \textit{and} \ldots .$$

For instance, the less-than relation $<$ is well-founded in the theory of nonnegative integers, but not in the theory of real numbers.

The version of the well-founded induction rule we need for the derivation is expressed as follows (the general version is more complex):

Suppose our initial tableau is

assertions	goals	outputs $f(a)$
$P[a]$		
	$\mathcal{R}[a, z]$	z

In other words, we are attempting to construct a program f that, for an arbitrary input a, yields an output z satisfying the input-output condition

if $P[a]$
then $\mathcal{R}[a, z]$.

According to the well-founded induction rule, we may prove this assuming as our induction hypothesis that the program f will yield an output $f(x)$ satisfying the same input-output condition

if $P[x]$
then $\mathcal{R}[x, f(x)]$,

provided that x is less than a with respect to some well-founded relation $\prec_w$, that is, $x \prec_w a$. In other words, we may add to our tableau the new assertion

if $x \prec_w a$ *then if* $P[x]$ *then* $\mathcal{R}[x, f(x)]$		

The well-founded relation $\prec_w$ used in the induction rule is arbitrary and must be selected later in the proof.

For example, consider the initial tableau obtained from the square-root specification:

assertions	goals	outputs $sqrt(r, \epsilon)$
$0 \leq r$ *and* $0 < \epsilon$		
	$z^2 \leq r$ *and* *not* $[(z+\epsilon)^2 \leq r]$	z

By application of the well-founded induction rule, we may introduce as a new assertion the induction hypothesis

if $\langle x, v \rangle \prec_w \langle r, \epsilon \rangle$ *then if* $0 \leq x$ *and* $0 < v$ *then* $\boxed{[sqrt(x, v)]^2 \leq x \text{ and } \textit{not}\ ([sqrt(x, v) + v]^2 \leq x)}$		

In other words, we may assume inductively that the output of the square-root program we construct will satisfy the input-output condition for inputs x and v that are less than the given inputs r and ϵ with respect to some well-founded relation $\prec_w$.

Use of the induction hypothesis in the proof may account for the introduction of a recursive call into the derived program. For example, suppose that in the square-root derivation we manage to develop a goal of form

	$\mathcal{G}\left[\begin{array}{l}\hat{z}^2 \leq s \ \ and \\ not\ ([\hat{z}+\delta]^2 \leq s\end{array}\right]^+$	$t[\hat{z}]$

The boxed subsentences of this goal and the induction hypothesis are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow s,\ v \leftarrow \delta,\ \hat{z} \leftarrow sqrt(s, \delta)\}.$$

Therefore, we can apply the resolution rule to obtain the new goal

	$\mathcal{G}[true]$ *and* $not\left[\begin{array}{l}if\ \langle s, \delta\rangle \prec_w \langle r, \epsilon\rangle \\ then\ if\ 0 \leq s\ \ and\ \ 0 < \delta \\ \qquad then\ false\end{array}\right]$	$t[sqrt(s, \delta)]$

This goal reduces under transformation to

	$\mathcal{G}[true]$ *and* $\langle s, \delta\rangle \prec_w \langle r, \epsilon\rangle$ *and* $0 \leq s$ *and* $0 < \delta$	$t[sqrt(s, \delta)]$

Note that a recursive call $sqrt(s, \delta)$ has been introduced into the output entry as a result of this step. The condition $(0 \leq s \ \ and \ \ 0 < \delta)$ in the goal ensures the legality of the arguments s and δ, i.e., that they satisfy the input condition of the desired program. The condition $\langle s, \delta\rangle \prec_w \langle r, \epsilon\rangle$ ensures that the evaluation of the recursive call cannot lead to a nonterminating computation. (If there were an infinite computation, we could construct a corresponding infinite sequence of pairs of arguments decreasing with respect to $\prec_w$, thus contradicting the definition of a well-founded relation.)

The particular well-founded relation $\prec_w$ referred to in the induction hypothesis is not yet specified; it is selected at a later stage of the proof. If we allow well-founded relations to be objects in our domain, we may regard the sentence $x \prec_w y$ as an abbreviation for $\prec(w, x, y)$; thus, w is a variable that may be instantiated to a particular relation. We assume that the properties of many known well-founded relations (such as $\prec_{tree}$, the proper-subtree relation over trees) and of functions for combining them are among the assertions of our initial tableau.

We have given the simplest version of the induction rule, which is applied only to the initial rows of the tableau; in its general version, we may apply the rule to any of the rows, and we may

strengthen or generalize the rows to which the rule is applied. In this more general version, the rule accounts for the introduction of auxiliary subprograms into the program being constructed. We shall avoid discussion of auxiliary subprograms here.

We are now ready to present the most interesting segment of the derivation of the square-root program.

THE DERIVATION

Recall that, in the theory of real numbers, the specification for the real-number square-root program is

$$sqrt(r,\ \epsilon) \Leftarrow \text{find } z \text{ such that } z^2 \leq r \ \textit{and not}\ [(z+\epsilon)^2 \leq r],$$
$$\text{where } 0 \leq r \ \textit{and}\ 0 < \epsilon.$$

In other words, we want to find an estimate z that is within a tolerance ϵ less than $\sqrt{r}$, the exact square root of r, where we may assume that r is nonnegative and ϵ is positive.

We begin accordingly with the tableau

assertions	goals	outputs $sqrt(r,\ \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $\boxed{z^2 \leq r}^{+}$ *and not* $[(z+\epsilon)^2 \leq r]$	z

The assertion and goal of this tableau are the input and output conditions, respectively, of the given specification; the output entry of the goal is the output variable of the program.

THE DISCOVERY OF BINARY SEARCH

We are about to apply the resolution rule to goal 2 and itself. To make this step easier to understand, let us write another copy of goal 2.

	2′. $\hat{z}^2 \leq r$ *and not* $\boxed{(\hat{z}+\epsilon)^2 \leq r}$	$\hat{z}$

We have renamed the variable of the second copy of the goal, so that the two copies have no variables in common.

The boxed subsentences of the two copies of the goal are unifiable; a most-general unifier is

$$\theta: \quad \{z \leftarrow \hat{z}+\epsilon\}.$$

Therefore, we can apply the resolution rule between the two copies of goal 2 to obtain

	true and not $[((\hat{z}+\epsilon)+\epsilon)^2 \leq r]$ *and* $\hat{z}^2 \leq r$ *and not false*	*if* $(\hat{z}+\epsilon)^2 \leq r$ *then* $\hat{z}+\epsilon$ *else* $\hat{z}$

By application of transformation rules, including the rule

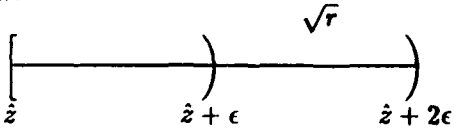
$$u + u \rightarrow 2u,$$

this goal can be reduced to

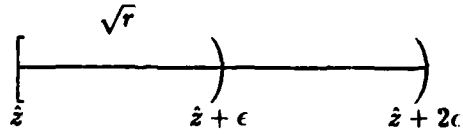
	3.	$\hat{z}^2 \leq r$ *and* *not* $[(\hat{z}+2\epsilon)^2 \leq r]$	*if* $(\hat{z}+\epsilon)^2 \leq r$ *then* $\hat{z}+\epsilon$ *else* $\hat{z}$

(We have reordered the conjuncts for pedagogical reasons only; because we use associative-commutative unification, their actual order is irrelevant.)

According to goal 3, it suffices to find a rougher estimate $\hat{z}$, which is within a tolerance 2ϵ less than $\sqrt{r}$, the exact square root of r. For then either $\hat{z}+\epsilon$ or $\hat{z}$ itself will be within ϵ less than $\sqrt{r}$, depending on whether or not $\hat{z}+\epsilon$ is less than or equal to $\sqrt{r}$. The two possibilities are illustrated below:



Case: $\hat{z}+\epsilon \leq \sqrt{r}$



Case: *not* $[\hat{z}+\epsilon \leq \sqrt{r}\,]$

Goal 3 contains the essential idea of binary search as applied to the square-root problem. Although the idea seems subtle to us, it appears almost immediately in the derivation. The step is nearly inevitable: any brute-force search procedure would discover it.

The derivation of goal 3 is logically straightforward, but the intuition behind it may be a bit mysterious. Let us paraphrase the reasoning in a more geometric way. Our initial goal 2 expresses that it suffices to find a real number z such that $\sqrt{r}$ belongs to the half-open interval $[z,\ z+\epsilon)$. Our rewritten goal $2'$ expresses that it is equally acceptable to find a real number $\hat{z}$ such that $\sqrt{r}$ belongs to the half-open interval $[\hat{z},\ \hat{z}+\epsilon)$. We shall be content to achieve either of these goals; i.e., we shall be happy if $\sqrt{r}$ belongs to either of the two half-open intervals. In taking $\hat{z}$ to be $z+\epsilon$, we are concatenating the two intervals, obtaining a new half-open interval $[z,\ z+2\epsilon)$ twice the length of the original. It suffices to find a real number z such that $\sqrt{r}$ belongs to this new, longer interval, because then $\sqrt{r}$ must belong to one or the other of the two smaller ones.

INTRODUCTION OF THE RECURSIVE CALLS

Let us continue the derivation one more step. By the well-founded induction rule, we may introduce the induction hypothesis

if $\langle x, v\rangle \prec_w \langle r, \epsilon\rangle$ *then if* $0 \le x$ *and* $0 < v$ *then* $\boxed{\begin{array}{l}[sqrt(x, v)]^2 \le x \text{ and} \\ \text{not } ([sqrt(x, v) + v]^2 \le x)\end{array}}$		

In other words, we assume inductively that the output $sqrt(x, v)$ of the program will satisfy the input-output condition for any inputs x and v such that $\langle x, v\rangle \prec_w \langle r, \epsilon\rangle$. The boxed subsentences of goal 3 and the induction hypothesis are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow r,\ v \leftarrow 2\epsilon,\ z \leftarrow sqrt(r, 2\epsilon)\}.$$

We obtain (after *true-false* transformation)

	4. $\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ *and* $0 \le r$ *and* $0 < 2\epsilon$	*if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \le r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

Note that at this point three recursive calls $sqrt(r, 2\epsilon)$ have been introduced into the output entry. The condition $(0 \le r$ *and* $0 < 2\epsilon)$ ensures that the arguments r and 2ϵ of these recursive calls will satisfy the input condition for the program, that r is nonnegative and 2ϵ is positive. The condition $\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ ensures that the newly introduced recursive calls cannot lead to a nonterminating computation. The well-founded relation $\prec_w$ that serves as the basis for the induction is as yet unspecified.

We omit those portions of the derivation that account for the introduction of the base case and the choice of the well-founded relation. The final program we obtain is

$$\begin{array}{lll} sqrt(r, \epsilon) & \Leftarrow & \textit{if } \epsilon \le max(r, 1) \\ & & \textit{then if } [sqrt(r, 2\epsilon) + \epsilon]^2 \le r \\ & & \quad \textit{then } sqrt(r, 2\epsilon) + \epsilon \\ & & \quad \textit{else } sqrt(r, 2\epsilon) \\ & & \textit{else } 0. \end{array}$$

A few words on this program are in order.

DISCUSSION OF THE PROGRAM

The program first checks whether the error tolerance ϵ is reasonably small. If ϵ is very big, that is, if $max(r, 1) < \epsilon$, then the output can safely be taken to be 0. For, because $0 \le r$, we have

$$0^2 \le r.$$

And because $max(r, 1) < \epsilon$, we have $r < \epsilon$ and $1 < \epsilon$, and hence $r < \epsilon^2$ – that is,

$$\textit{not } [(0 + \epsilon)^2 \le r].$$

Thus, 0 satisfies both conjuncts of the output condition in this case.

If ϵ is small, that is, $\epsilon \leq max(r,\ 1)$, the program finds a rougher estimate $sqrt(r,\ 2\epsilon)$, which is within 2ϵ less than $\sqrt{r}$. The program asks whether increasing this estimate by ϵ will leave it less than $\sqrt{r}$. If so, the rough estimate is increased by ϵ; if not, the rough estimate is already close enough.

The termination of the program is a bit problematic, because the argument ϵ is doubled with each recursive call. However, the argument r is unchanged and recursive calls are evaluated only in the case in which $\epsilon \leq max(r,\ 1)$, so there is a uniform upper bound on these increasing arguments. More precisely, the well-founded relation $\prec_w$ selected in the proof is one such that

$$\langle x,\ 2y\rangle \prec_w \langle x,\ y\rangle,$$

provided that $0 < y \leq max(r,\ 1)$.

If the multiple occurrences of the recursive call $sqrt(r,\ 2\epsilon)$ are combined by eliminating common subexpressions, the program we obtain is reasonably efficient; it requires $\lceil log_2(max(r,\ 1)/\epsilon)\rceil$ recursive calls.

Our final program is somewhat different from the iterative program we considered in the beginning. The iterative program divides an interval in half at each iteration; the recursive program doubles an interval with each recursive call. Division of the interval in half occurs implicitly as the recursive program unwinds, i.e., when the recursive calls yield output values.

It is possible to obtain a version of the iterative program by formal derivation within the deductive-tableau system. Although the derivation and the resulting program are more complex (it requires two additional inputs), it was this derivation we discovered first, because we were already familiar with the iterative program.

We first found the recursive program in examining the consequences of purely formal derivation steps, not because we expected them to lead to a program but because we were looking for strategic considerations that would rule them out. When we examined the program initially, we suspected an error in the derivation. We had not seen programs of this form before, and we certainly would not have constructed this one by informal means.

ANALOGOUS ALGORITHMS

Many binary-search algorithms have been derived in an analogous way. Let us first consider some other real-numerical problems.

REAL-NUMBER ALGORITHMS

Suppose a program to perform real-number division is specified as follows:

$$\begin{aligned} div(r,\ s,\ \epsilon) \ \Leftarrow\ & \text{find } z \text{ such that} \\ & z \cdot s \leq r \ \textit{and not}\ \left[(z+\epsilon)\cdot s \leq r\right] \\ \text{where } 0 \leq r \ \textit{and}\ & 0 < s \ \textit{and}\ 0 < \epsilon. \end{aligned}$$

In other words, the program is required to yield a real number z that is within a tolerance ϵ less than r/s, the exact quotient of dividing r by s. We obtain the program

$$\begin{aligned} div(r,\, s,\, \epsilon) \;\Leftarrow\; & \textit{if } c \cdot s \leq r \\ & \textit{then if } \left[div(r,\, s,\, 2c) + \epsilon\right] \cdot s \leq r \\ & \qquad \textit{then } div(r,\, s,\, 2c) + \epsilon \\ & \qquad \textit{else } div(r,\, s,\, 2c) \\ & \textit{else } 0. \end{aligned}$$

The rationale for this program, like its derivation, is analogous to that for the real-number square root. The program first checks whether the error tolerance is reasonably small, that is, if $c \cdot s \leq r$. If ϵ is very big, that is, if $r < c \cdot s$, then the output can be taken safely to be 0. For because $0 \leq r$, we have

$$0 \cdot s \leq r.$$

And because $r < c \cdot s$, we have $r < (0 + c) \cdot s$, that is,

$$\textit{not } \left[(0 + c) \cdot s \leq r\right].$$

Thus, 0 satisfies both conjuncts of the output condition in this case.

On the other hand, if c is small, that is, if $\epsilon \cdot s \leq r$, the program finds a rougher estimate $div(r,\, s,\, 2c)$, which is within $2c$ less than r/s. The program considers whether increasing this estimate by c will leave it less than r/s. If so, the rough estimate may be increased by c; if not, the rough estimate is already close enough.

The termination proof for this program is also analogous to that for the square root. Although the argument c is doubled with each recursive call, the other arguments are unchanged and the calls are evaluated only in the case in which $c \cdot s \leq r$, that is, $c \leq r/s$. Thus, there is a uniform upper bound on the doubled argument.

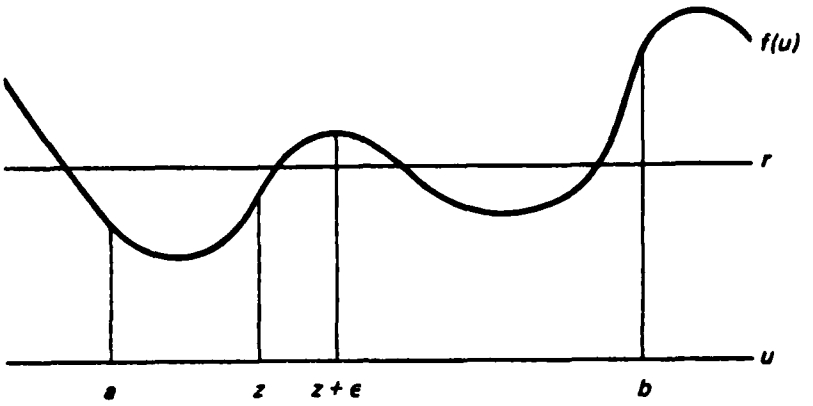
It may be clear from the above discussion that there is little in the derivations for the square-root and division programs that depends on the properties of these functions. More or less the same derivation suffices to find an approximate solution to an arbitrary real-number equation $f(z) = r$.

For a given computable function f, we consider the specification

$$\begin{aligned} solve(r,\, \epsilon) \Leftarrow\; & \text{find } z \text{ such that} \\ & \qquad f(z) \leq r \;\textit{ and }\; \textit{not } \left[f(z + \epsilon) \leq r\right] \\ \text{where } \; & f(a) \leq r \;\textit{ and }\; \begin{bmatrix} \textit{if } b < u \\ \textit{then not } \left(f(u) \leq r\right) \end{bmatrix}. \end{aligned}$$

Here a and b are primitive constants and u is a variable. In other words, we assume that there exist real numbers a and b such that $f(a) \leq r$ and $f(u) > r$ for every real u greater than b. The

specification is illustrated as follows:



Note that we do not need to assume f is increasing or even continuous; if f is not continuous, an exact solution to the equation $f(a) = r$ need not exist, but only an approximate solution is required by the specification.

The program we obtain is

$$\begin{aligned} solve(r,\ \epsilon) \ \Leftarrow\ & \textit{if } a+\epsilon \le b \\ & \textit{then if } f\big(solve(r,\ 2\epsilon)+\epsilon\big) \le r \\ & \qquad \textit{then } solve(r,\ 2\epsilon)+\epsilon \\ & \qquad \textit{else } solve(r,\ 2\epsilon) \\ & \textit{else } a. \end{aligned}$$

In the recursive case, in which $a + \epsilon \le b$, the program is so closely analogous to the previous binary-search programs as to require no further explanation. In the base case, in which $b < a + \epsilon$, the output can safely be taken to be a. For, by our input condition, we have

$$f(a) \le r$$

and (again by our input condition, because $b < a + \epsilon$)

$$not\ \big[f(a+\epsilon) \le r\big].$$

Thus, a satisfies both conjuncts of the output condition in this case.

The above program may be regarded as a schema, because we may take the symbol f to be any primitive function symbol. An even more general binary-search program schema can be derived from the specification

$$\begin{aligned} search(r,\ \epsilon) \ \Leftarrow\ & \text{find } z \text{ such that} \\ & \qquad p(r,\ z)\ \textit{and}\ not\, p(r,\ z+\epsilon) \\ \text{where } p(a)\ \textit{and}\ & \begin{bmatrix} \textit{if } b < u \\ \textit{then } not\, p(r, u) \end{bmatrix}, \end{aligned}$$

where p is a primitive relation symbol and a and b are primitive constants. We obtain the schema

$$\begin{aligned} search(r,\ \epsilon) \ \Leftarrow\ & \textit{if } a+\epsilon \le b \\ & \textit{then if } p\big(r, search(r,\ 2\epsilon)+\epsilon\big) \\ & \qquad \textit{then } search(r,\ 2\epsilon)+\epsilon \\ & \qquad \textit{else } search(r,\ 2\epsilon) \\ & \textit{else } a. \end{aligned}$$

INTEGER ALGORITHMS

The programs we have discussed apply to the nonnegative real numbers; using the same approach, we have derived analogous programs that apply to the nonnegative integers. These derivations require a generalization step in applying the induction rule. We have avoided presenting generalization and the concomitant introduction of auxiliary programs in this paper, but we give some results of these derivations here.

Integer square root

The integer square-root program is intended to find the integer part of $\sqrt{n}$, the real square root of a nonnegative integer n. It can be specified in the theory of nonnegative integers as follows:

$$sqrt(n) \Leftarrow \text{find } z \text{ such that } z^2 \leq n \text{ and not}\big[(z+1)^2 \leq n\big].$$

In other words, the program must yield a nonnegative integer z that is within 1 less than $\sqrt{n}$.

In the course of the derivation, we are led to introduce an auxiliary program to meet the more general specification

$$sqrt2(n, i) \Leftarrow \text{find } z \text{ such that } z^2 \leq n \text{ and not } \big[(z+i)^2 \leq n\big] \quad \text{where } 0 < i.$$

In other words, we wish to find a nonnegative integer z that is within i less than $\sqrt{n}$. This auxiliary specification is precisely analogous to the real-number square-root specification, with i playing the role of the error tolerance ϵ.

The programs we obtain to meet these specifications are

$$sqrt(n) \Leftarrow sqrt2(n, 1),$$

where

$$sqrt2(n, i) \Leftarrow \textit{if } i \leq n \textit{ then if } \big[sqrt2(n, 2i) + i\big]^2 \leq n \textit{ then } sqrt2(n, 2i) + i \textit{ else } sqrt2(n, 2i) \textit{ else } 0.$$

Integer quotient

The integer quotient program can be specified similarly:

$$quot(m, n) \Leftarrow \text{find } z \text{ such that } z \cdot n \leq m \text{ and not } \big[(z+1) \cdot n \leq m\big] \quad \text{where } 0 < n.$$

In other words, we wish to find a nonnegative integer z that is within 1 less than m/n, the real-number quotient of m and n.

In the course of the derivation, we are led to introduce an auxiliary program to meet the more general specification

$$\begin{aligned} quot3(m,\ n,\ i) \ \Leftarrow\ & \text{find } z \text{ such that} \\ & \quad z \cdot n \le m \ \textit{and}\ \textit{not}\big[(z+i)\cdot n \le m\big] \\ \text{where } 0 < n \ \textit{and}\ & \ 0 < i. \end{aligned}$$

In other words, we wish to find a nonnegative integer z that is within i less than m/n.

The programs obtained to meet these specifications are

$$quot(m,\ n) \ \Leftarrow\ quot3(m,\ n,\ 1)$$

where

$$\begin{aligned} quot3(m,\ n,\ i) \ \Leftarrow\ & \textit{if}\ i \cdot n \le m \\ & \textit{then}\ \textit{if}\ \big[quot3(m,\ n,\ 2i) + i\big] \cdot n \le m \\ & \qquad \textit{then}\ quot3(m,\ n,\ 2i) + i \\ & \qquad \textit{else}\ quot3(m,\ n,\ 2i) \\ & \textit{else}\ 0. \end{aligned}$$

The derivation is again analogous.

DISCUSSION

The derivations were first discovered manually; the real-number square-root derivation was subsequently reproduced by Yellin in an interactive program-synthesis system. The only automatic implementation of the system (Russell [83]) is unable to construct the derivation for a simple reason: it never attempts to apply the resolution rule to a goal and itself.

The results of this investigation run counter to our usual experience. It is common for a bit of reasoning that seems simple and intuitively straightforward to turn out to be difficult to formalize and more difficult still to duplicate automatically. Here the opposite is true: an idea that requires a substantial leap of human ingenuity to discover is captured mechanically in a few easy formal steps.

ACKNOWLEDGMENTS

We would like to thank Martin Abadi, Yoni Malachi, Eric Muller, Mark Stickel, Jonathan Traugott, and Frank Yellin for discussions and helpful suggestions on the subject of this paper.

REFERENCES

Dershowitz and Manna [77]
N. Dershowitz and Z. Manna, The evolution of programs: Automatic program modification, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 6, November 1977, pp. 377-385.

Manna and Waldinger [80]
Z. Manna and R. Waldinger, A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, January 1980, pp. 90-121.

Manna and Waldinger [85]
Z. Manna and R. Waldinger, Special relations in automated deduction, *Journal of the ACM*, 1985, to appear.

Murray [82]
N. V. Murray, Completely nonclausal theorem proving, *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 67–85.

Robinson [79]
J. A. Robinson, *Logic: Form and Function*, North-Holland, New York, N. Y., 1979.

Russell [83]
S. Russell, PSEUDS: A programming system using deductive synthesis, Technical Report, Computer Science Department, Stanford University, Stanford, Calif., September 1983.

Smith [85]
D. R. Smith, Top-down synthesis of simple divide-and-conquer algorithms, *Artificial Intelligence*, 1985, to appear.

Stickel [81]
M. E. Stickel, A unification algorithm for associative-commutative functions, *Journal of the ACM*, Vol. 28, No. 3, July 1981, pp. 423-434.

Wensley [59]
J. H. Wensley, A class of nonanalytical iterative processes, *Computer Journal*, Vol. 1, January 1959, pp. 163-167.

END

FILMED

7-85

DTIC