THE ORIGIN OF THE BINARY-SEARCH PARADIGM

Zohar Manna
Computer Science Department
Stanford University

Richard Waldinger
Artificial Intelligence Center
SRI International

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.

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 e less than $\sqrt{r}$.

$$
\begin{array}{l}
z \leftarrow 0 \\
v \leftarrow max(r, 1) \\
\textbf{while } \epsilon \leq v \textbf{ do } v \leftarrow v/2 \\
\qquad\qquad \textit{if } [z+v]^2 \leq r \textit{ then } z \leftarrow z+v \\
\textbf{return}(z)
\end{array}
$$

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 the given tolerance e. 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], [85b]) 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 the derivation of a numerical program up to the point at which the binary-search concept emerges and discuss what it indicates about the prospects for automatic program synthesis.

We assume familiarity with the deductive-tableau approach to program synthesis; readers who would like an introduction are referred to the full version of this paper (Manna and Waldinger [85a]). In the full paper, we also show several analogous binary-search programs that have been developed by the same method, including quotient programs for the nonnegative reals and integers.

THE DERIVATION

In the theory of real numbers, the specification for the real-number square-root program is

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

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 *e* is positive. >

This research was supported in part by the National Science Foundation under grants MCS-82-14523 and MCS-81-05565, by the 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.

We begin accordingly with the tableau

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

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

	$\left[\begin{array}{l} true \\ \quad and \\ not\ [((\hat{z}+\epsilon)+\epsilon)^2 \le r] \end{array}\right]$ *and* $\hat{z}^2 \le r$ *and not false*	*if* $(\hat{z}+\epsilon)^2 \le r$ *then* $\hat{z}+\epsilon$ *else* $\hat{z}$

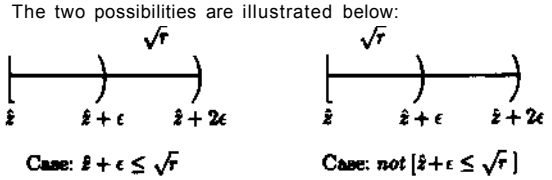
By application of transformation rules, including the rule

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

this goal can be reduced to

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

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



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 + e).* Our rewritten goal 2' expresses that it is equally acceptable to find a real number 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 z to be z + e, 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 $(x, v) \prec_w (r, \epsilon)$ *then if* $0 \le x$ *and* $0 < v$ *then* $\boxed{[sqrt(x, v)]^2 \le x}$ *and* *not* $([sqrt(x, v)+v]^2 \le x)$		

In other words, we assume inductively that the output *8qrt(x, v)* of the program will satisfy the input-output condition for any inputs *x* and *v* such that $(x, v) \prec_w (r, \epsilon)$. The boxed subsentences of goal 3 and the induction hypothesis are unifiable; a most-general unifier is

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

We obtain (after *true-false* transformation)

	4. $(r, 2\epsilon) \prec_w (r, \epsilon)$ *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 2E of these recursive calls will satisfy the input condition for the program, that r is nonnegative and 2c is positive. The condition $(r, 2\epsilon) \prec_w (r, \epsilon)$ ensures that the newly introduced recursive calls cannot lead to a nonterminating computation. The well-founded relation $<_w$ 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

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

A few words on this program are in order.

Discussion of the Program

The program first checks whether the error tolerance e is reasonably small. If ϵ is very big, that is, if *max(r,* 1) < ϵ, then

the output can safely be taken to be 0. For, because $0 \leq r$, we have

$$0^2 \leq r.$$

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

$$not\ [(0 + \epsilon)^2 \leq r].$$

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

If e 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 *e* 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 $\prec_\omega$ selected in the proof is one such that

$$\langle x, 2y \rangle \prec_\omega \langle x, y \rangle,$$

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.

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 formalise 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 in a few easy formal steps. We may hope that truly original ideas will arise from the fortunate application of simple mechanisms.

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 [85a]
Z. Manna and R. Waldinger, The origin of the binary search paradigm, Technical Report, Computer Science Department, Stanford University, Stanford, CA and Artificial Intelligence Center, SRI International, Menlo Park, CA (April 1985).

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

Manna and Waldinger [85c]
Z. Manna and R. Waldinger, The Logical Basis for Computer Programming, Volume 1: Dedutive Reasoning, Addison-Wesley, Reading, Mass., 1985.

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, unpublished 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.

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