Final Report

Covering the Period 18 June 1970 to 1 June 1971

RESEARCH IN ADVANCED FORMAL THEOREM-PROVING TECHNIQUES

By: J. F. RULIFSON

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HEADQUARTERS
600 INDEPENDENCE AVENUE, S.W.
ROOM 607
WASHINGTON, D.C. 20546
Attention: MR. JAMES ALBUS

CONTRACT NASW-2086

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

Final Report *June 1971*

Covering the Period 18 June 1970 to 1 June 1971

RESEARCH IN ADVANCED FORMAL THEOREM-PROVING TECHNIQUES

By: J. F. RULIFSON

Prepared for:

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HEADQUARTERS
600 INDEPENDENCE AVENUE, S.W.
ROOM 607
WASHINGTON, D.C. 20546
Attention: MR. JAMES ALBUS

CONTRACT NASW-2086

SRI Project 8721

Approved by:

B. RAPHAEL, *Director*
Artificial Intelligence Center

BONNAR COX, *Executive Director*
Information Science and Engineering Division

Copy No. 5

CONTENTS

ABSTRACT

This report summarizes the present status of a continuing research program aimed at the design and implementation of a language for expressing problem-solving procedures in several areas of artificial intelligence, including program synthesis, robot planning, and theorem proving. Notations, concepts, and procedures common to the representation and solution of many of these problems have been abstracted and incorporated as features into the language.

This report describes briefly the areas of research covered, and then presents abstracts of six papers that contain extensive description and technical detail of the work.

Page Intentionally Left Blank

I INTRODUCTION

For two years, Stanford Research Institute has been engaged in the design of a language, called QA4, for expressing problem-solving procedures.* We have abstracted notations, concepts, and procedures common to the representation and solution of many of these problems, and have incorporated these features into the language. The first version of an interpreter for this language is nearing completion.

This work was begun as part of a general program of research in artificial intelligence supported at SRI by NASA and ARPA under Contracts NAS12-2221, NASW-2164, and NASW-2086. For the past year the work has been supported primarily by NASA under Contract NASW-2086; this is the final report for that project. Early in the project its scope was changed from the development of advanced formal theorem-proving methods, as originally stated, to the development of new programming formalisms and associated problem-solving methodologies. These latter goals are now viewed as prerequisites for the work on theorem-proving methods per se, and also have use in several other problem domains.

As noted, the effort described here is part of a program that has been partially supported by several different projects. Additionally, we anticipate continuing this program at an increased level under new support. The program is at a mid-point in its development. Preliminary language design has been completed, and implementation of an interpreter is nearly completed.

The results from this project are being reported in several technical publications. This final report consists of a summary of the research performed, and abstracts of the papers and reports that contain further details.

II SUMMARY OF RESEARCH

The goals of this project are to

- Explore natural, intuitive methods of theorem-proving and problem-solving
- Formalize a language for expressing these theorem-proving and problem-solving techniques and plans
- Design and start the construction of an interpreter for this language.

---

*C. C. Green and R. A. Yates participated heavily in the original work in this program. R. J. Waldinger and J. A. Derksen, as well as the author of this report, have been continual contributors.

During the next year we plan to extend this research by first finishing the interpreter, and then testing and refining our formalisms by building problem-solving programs within the language.

A. Theorem-Proving Protocols

We used protocol analysis as a vehicle to explore problem-solving methods. Thus the project has entailed the construction and reworking of hand simulations of a proposed theorem prover. Each simulation includes a problem statement, relevant definitions and advice, and a protocol for the solution, all stated within the QA4 language formalisms.

We see the theorem prover, written in the QA4 language, working at the same level of detail as a human mathematician, and a finished proof should resemble a proof in a mathematical textbook. A typical protocol demonstrates the conversion of a recursive program for calculating Fibonacci numbers to an equivalent iterative program.[1*] From the recursive definition, approximately ten rules of inference, and some constraints on the final program, the protocol presents a hypothetical solution we eventually expect the problem-solver to produce automatically.

This method of exploration has permitted us to take simultaneously two points of view. Our protocols could be intuitive and loosely organized. The notations could be natural and concise. And the inference rules could be exceptionally high-level for mechanical theorem-proving. At the same time, by analyzing the plans, we could work out formalisms that had enough precision to permit their implementation in a programming language. Thus the simulations have provided a focus for the language development and prompted explorations into theoretical foundations of the problem areas.[2] As soon as the interpreter is able to execute some of the strategies, the solution that has been hand-constructed in the past will be produced by an interactive program. The method of introspective problem-solving and simulation, however, will remain a major technique in future extensions of the project.

B. QA4 Language and Interpreter

A careful analysis of these protocols revealed that a natural, intuitive language required many formalisms not found in already existing programming languages. The protocols relied heavily on both obvious features such as sets and far more subtle capabilities such as hypothetical reasoning. For the remainder of this Section, we briefly discuss the new formalisms that were especially designed for the QA4 language. To implement many of these formalisms we were required to take some totally new approaches to programming language interpretation

---

*References are to the publications abstracted in Section III of this report.

and invent new symbolic-expression manipulation methods. Many of these inventions are discussed with the formalisms they implement.

1. Overview

Seemingly simple axioms and inference rules normally presented in a mixed English-logic language in textbooks often become lengthy, complex formulas when converted to the notation of first-order predicate calculus. In order to stimulate the intuition of the programmer, as well as to simplify the specification of the symbol manipulation tasks performed by problem-solving programs, the QA4 language is more a formal (but natural to use) mathematical language than a strictly logical language. Declarative statements in the language therefore have the convenience of infix-style syntax coupled with such useful extensions of omega-order calculus as sets, special quantifiers, and extended primitive operators.[3] Additional imperative statements permit a concise notation for program control and organization.

As expressions are composed, they are converted to a canonical form so that semantic and pragmatic properties attached to them can be associated automatically with all equivalent expressions.[1,4,5] Composition takes place whenever a particular data structure is constructed by a program. For example, the process of interpreting the statement

$$X \leftarrow \{RED,\ BLUE,\ GREEN\}$$

not only assigns X to be a set, but also composes a canonical representation of the set. The composition process ensures that if the datum described by the expression (in this case, the set) has ever been previously constructed, the original value is used and no new equal but different structure is constructed. This identification of equivalence is even made between expressions that include bound variables. Thus, the functions

$$(LAMBDA\langle X,\ Y\rangle,\ (X+Y)*(Y+1))$$

and

$$(LAMBDA\langle U,\ V\rangle,\ (1+V)*(V+U))$$

will both be converted to the same internal canonical form, and information known about one is always available to strategies that may deal with the other.

Retrieval and decomposition of expressions is accomplished by template pattern matching.[4,5] Decomposition occurs during the process of assigning arguments to functions. For example, the interpretation of the statement

$$\{?X,\ ?Y,\ ...\} \leftarrow \{RED,\ BLUE,\ GREEN,\ YELLOW\}$$

assigns one of the four color words to X and to Y. The triple dots denote a fragment, and permit the sets on opposite sides of the arrow to be of different cardinality. Since sets are involved, X and Y may be assigned the same word or different words. The statement

EXISTS {?X, RED, ...}

will retrieve from the data base all sets that contain the word RED. X is then assigned some element from one of the retrieved sets. This form of pattern matching permits programs to be nondeterministic. The backtracking necessary to interpret the programs is handled automatically by the interpreter.

The storage mechanism for QA4 expressions is implemented as a discrimination net.[1,4,5] While the theoretical characteristics of this type of structure are understood, considerable improvements in efficiency should still be achievable by observing its operation during problem solutions and making appropriate modifications. The pattern matcher, which is the last major component to be integrated into the interpreter, has been designed and is partially implemented.[4,5] Many of the heuristics that guide the pattern matcher depend on the specific form of the patterns. Thus, as the problem-solving programs take form, a continual effort can be made to improve the internal operation of the pattern matcher, as well as to incorporate new syntactic features that enhance the template notation.

2. Program Organization

Problem-solving programs written in QA4 will be organized around strategies such as

> Whenever a sentence follows from previous assertions by modus ponens, assert that sentence.

Heuristics such as these are easily introduced into a problem-solving system by using a WHEN statement. The statement directs the interpreter that whenever any expression that matches this pattern is assigned a particular property execute this program.

Internally, strategies are expected to accomplish their task by dividing their assigned problems into appropriate subgoals. Each subgoal can be specified by a single GOAL statement such as

GOAL PROVE, P(A) & P(B) .

Through the use of other QA4 statement forms, the interpreter can be informed which strategies apply to certain types of goals. As new goals arise, the interpreter has the responsibility to choose and apply strategies to accomplish the goal.

Thus, the problem-solving programs operate by means of the interaction between goal-directed strategies.[1] The GOAL and WHEN statements permit independent specifications of these strategies, and the interpreter permits parallel and distributed-control programming. Rather than being called by name, GOAL and WHEN programs are activated by conditions that arise during the course of problem solution.[4,5] This programming facility is a major contribution toward flexible, natural strategy organization.

Another class of statements provide iteration over sets guided by pattern matching. They permit the concise, elegant representation of individual strategies. For example, a statement that finds three boxes that contain green blocks is described simply by the following QA4 code:

```
FIND 3, COLLECT $X,
     DO    GOAL PROVE, BOX(←X);
           GOAL PROVE, CONTAINS ⟨$X,←B⟩;
           GOAL PROVE, GREEN($B) END    .
```

The portions of the interpreter that handle these program forms have been designed and are currently being implemented. Care is being taken to provide flexibility in the interpreter so that changes in the language can be accommodated easily as they develop during the construction of the problem-solving system.

3. Program Control

Whenever one is attempting to establish an implication, a natural method is to assume the antecedent and attempt to establish the consequent. The task may be to prove a logical implication such as

If Socrates is a man, then he is fallible .

or to establish a casual implication such as

If a block is moved, then its position is changed.

In either case a problem-solving strategy must create a "context," make hypothetical assumptions, and derive conclusions. When it is finished, however, it must erase intermediate results, for they depend on the truth of the antecedent, which may not be true even though the implication has been established.

As strategies are invoked in a QA4 program, they are assigned a "context" in which they operate. These running strategies may operate independently in parallel, or may cooperate in a high degree of synchronization.[4,5] The backtracking, side-effects, and communication paths

of these processes are highly controllable.[6] Moreover, the control may be either handled automatically by the interpreter, or manipulated by the strategies themselves.

These abilities are implemented by means of a generalized context and process-control scheme. The exploration of various forms of process interactions in the problem solver should lead to the discovery of those methods that are useful and natural. As the dominant control forms appear, efficiency in the interpreter can be gained by refining and tuning the context scheme.

III ABSTRACTS OF REPORTS

The following reports were all at least partially supported under this project. Among them, they contain the technical details of the supported research. Copies of Artificial Intelligence Group Technical Notes are available from their authors and may also be obtained from the Publications Department of Stanford Research Institute.

1. J. F. Rulifson, R. J. Waldinger, and J. A. Derksen, "QA4 Working Paper," Technical Note 42, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (October 1970).

ABSTRACT

This note begins with an outline of control statements that will be the framework of the QA4 language. Next, follows a detailed protocol of the solution of a program-synthesis problem. The protocol uses high-level inference rules and relies on the intelligent selection and application of the rules. Finally, the overall structure of a problem solver capable of the preceding solution is sketched.

2. Z. Manna and R. J. Waldinger, "On Program Synthesis and Program Verification," Technical Note 52, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California (November 1970); paper presented at 4th Hawaii International Conference on Systems Sciences, Honolulu, Hawaii, 12-14 January, 1971.

ABSTRACT

Certain similarities between program verification and program synthesis are pointed out. The analogy is illustrated using a "bubble-sort" program.

3. J. F. Rulifson, "Preliminary Specification of the QA4 Language," Artificial Intelligence Group Technical Note 50, Stanford Research Institute, Menlo Park, California (April 1970).

ABSTRACT

The declarative facet of the QA4 language is an extension of higher-order logic. Set, bag, and tuple expressions and operations have been added, and the bound variable occuring in LAMBDA expressions has been significantly modified. This note describes initial specifications for the logic-language and briefly discusses an approach to implementation.

4. J. F. Rulifson, R. J. Waldinger, and J. A. Derksen, "The QA4 Language," Artificial Intelligence Group Technical Note 57, Stanford Research Institute, Menlo Park, California (in preparation).

ABSTRACT

This note is a guide to the complete QA4 language. The form and semantics of all logical expressions, primitive operations, patterns, and control statements is informally presented. Unusual forms and programming methods are illustrated with brief examples. Discussions of needed design work and further expansion are interspersed throughout the note.

5. J. F. Rulifson, R. J. Waldinger, and J. A. Derksen, "A Language for Writing Problem Solving Programs," Artificial Intelligence Group Technical Note 48, Stanford Research Institute, Menlo Park, California (April 1971); paper to be presented at IFIP Congress '71, Ljubljana, Yugoslavia 23-28 August, 1971.

ABSTRACT

This paper describes a language for constructing problem-solving programs. The language can manipulate several data structures, including ordered and unordered sets. Pattern-matching facilities may be used in various ways, including the binding of variables. Implicit backtracking facilitates the compact representation of search procedures. Expressions are treated analogously to atoms in LISP. A "context" device is used to implement variable bindings, to effect conditional proofs, and to solve the "frame" problem in robot planning.

6. J. F. Rulifson, "QA4 Note: The Context Mechanism," Artificial Intelligence Group Technical Note 58, Stanford Research Institute, Menlo Park, California (in preparation).

ABSTRACT

Variable bindings are implemented in the QA4 interpreter with a "context" mechanism. This method of storing all the changeable properties of expressions simplifies backtracking and the execution of parallel processes. The same facilities, moreover, are made available to the users as a method of data manipulation in programs dealing with conditional proofs or robot planning programs confronted with the frame problem. This note describes the algorithms used to bind variables, illustrates with extensive examples the operation of these algorithms, and concludes with a brief discussion of the implementation.