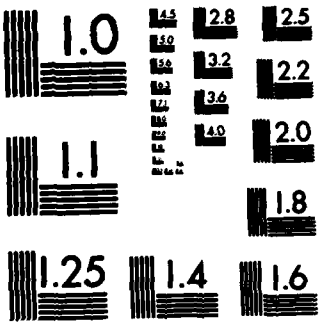
AD-A144 356 REHOSTING THE THEOREM PROVER(U) TEXAS UNIV AT AUSTIN 1/1
INST FOR COMPUTING SCIENCE AND COMPUTER APPLICATIONS
R S BOYER ET AL. 13 JUL 84 N00014-81-K-0634

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

THE UNIVERSITY OF TEXAS AT AUSTIN
INSTITUTE FOR COMPUTING SCIENCE AND COMPUTER APPLICATIONS
AUSTIN, TEXAS 78712

2100 Main Building
(512) 471-1901

13 July 1984

Dr. Robert Grafton
Information Systems Program
Mathematicial & Information Sciences Division
Attn: Code 437
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

Dear Dr. Grafton:

This letter represents the annual summary report for ONR Contract N00014-81-K-0634 for the period June 1, 1983 to May 31, 1984.

We have pursued two different research activities during this period: rehosting our theorem prover onto several different computers and extending the formal logic supported by the system.

REHOSTING THE THEOREM PROVER

For the past decade of its development our theorem prover has been implemented in Interlisp under the TOPS-20 time sharing operating system on a DEC-2060. Because of address limitations of the 2060 and the need for more LISP cycles than delivered by a time-shared machine, we acquired a Symbolics 3600 Lisp Machine on an NSF/CER Grant to the University of Texas. The machine arrived in May, 1983.

Much of our time the subsequent Summer was spent converting the system from Interlisp to Zetalisp so that it would run on the 3600. Since Zetalisp is a close relative of Maclisp, we decided to limit, insofar as possible, our use of Zetalisp primitives to those also in Maclisp so that we might more easily port the system to other machines supporting Maclisp dialects. Throughout the Fall and Winter we continued rewriting the system to make it run under the TOPS-20, Multics, SAIL operating systems. By further restricting the set of LISP primitives used we have been able to port the system to the VAX and the Sun (in Franzlisp). As a result of this work the system is now available on a wide variety of relatively inexpensive machines and the number of users of the system has increased by at least an order of magnitude.

It is interesting to note that our theorem prover runs faster in Zetalisp on the 3600 than

it does in Interlisp on a dedicated DEC 2060. In particular, the 3600 does our standard benchmark of about one thousand theorems in roughly 61% of the time taken by the 2060.

QUANTIFIERS

We have developed an extension to our quantifier-free computational logic that permits the use of bounded quantifiers such as "the sum as i runs over L of f(i)" and "the list of those x in L with property p(x)." Such quantified expressions are extremely common in both mathematics and computing. For any particular choice of the schematic expressions f(i) or p(x) it is possible to avoid the use of quantified expressions by the introduction of an ad hoc recursive function. However, the ability to manipulate quantified expressions involving schemators such as f and p is extremely useful for it permits the statement and proof of simple but powerful lemmas about quantifiers. For example, one can prove that, for any schematic expression f(i), the sum of f(i) as i ranges over the concatenation of A and B is the sum of f(i) over A plus the the sum of f(i) over B. It does not matter whether f(i) is i*i or 3+i or gcd(i,x). The lack of information about f aids the mechanical discovery of the proof — because the system does not get confused by a possibly vast amount of information about some particular f. Furthermore, the extremely general statement of the lemma permits its application in a wide variety of situations where particular f's are involved.

Our method of adding quantifiers to the logic is novel in that it does not alter the term structure of the language or the rules of inference. Instead we simply add some additional axioms defining an "interpreter" for the logic in the logic and a function, FOR, which iteratively "evaluates" a "term" in a succession of environments. Thus, the expression:

(for I in L when (EVEN I) sum (TIMES I I)),

which is an abbreviation for

(FOR 'I L '(EVEN I) 'SUM '(TIMES I I) NIL),

is a term whose value is the sum of the squares of the even elements of the list L.

Note that FOR takes as arguments "expressions" in the logic — expressions represented by list constants. The definition of FOR requires the addition to the logic of an interpreter for the logic. The primary research issue here was how we could embed into the logic a sufficiently powerful interpreter without rendering the formal system inconsistent. This problem is faced in all "sufficiently powerful" formal systems such as higher

order logic and set theory. The protection mechanisms in such systems (e.g., hierarchies of types or indexed variable symbols) prevent the utterance of certain potentially useful sentences but also prevent inconsistency. There seems to be a tradeoff between the expressibility permitted by a given mechanism and the complexity of the mechanism itself. Often the mechanisms adopted are so clumsy that their authors immediately introduce notational conventions to suppress them in informal proofs. However, in formal proofs — and in particular, in mechanical proofs — these mechanisms cannot be ignored without risking the validity of the enterprise itself. We were therefore driven to investigate protection mechanisms that were more limiting in their restrictions but less obtrusive in formal proofs. We discovered an interesting and novel formal device for avoiding inconsistently while providing what we believe is acceptable quantificational power. The device is employed in the enclosed description of our new logic and is discussed in a paper now in preparation.

Having invented a suitable extension to the logic we then extended our mechanical theorem prover. It is in the implementation that we see the advantages of preserving the original term structure and rules of inference of the logic: the old theorem prover is sound for the new logic. Indeed, in order to implement a theorem prover for the quantified logic all we had to do was add certain axioms to the initial data base. We did not have to inspect or modify the half-megabyte of code that implements the old theory. The result was a machine that proved some interesting elementary theorems about quantifiers, such as that the sum as i runs over L of f(i)+g(i), is the sum as i runs over L of f(i) plus the sum as i runs over L of g(i). To improve the performance of the system on quantified theorems it was only necessary to add heuristics that deal with FOR. We did not have to alter existing heuristics or proof techniques.

The new system is as fast as the old when dealing with "old" theorems — indeed, it executes exactly the same code; the new system is as smart as the old when dealing with unquantified fragments of a quantified formula; and, because of our new heuristics for dealing with FOR, the new system is reasonably powerful for quantified expressions. The most interesting proof produced thus far involving quantifiers is the proof of the Binomial Theorem:

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}.$$

The proof requires several interesting lemmas about quantifier manipulation, e.g., moving constants in and out of sigmas and changing the range of the quantifiers.

PAPERS

During the current reporting period we wrote the enclosed paper on program verification. The paper is an overview of the field written at the invitation of Larry Wos for inclusion

in the inaugural edition of the new Journal of Automated Reasoning.

The new quantified logic is described in the enclosed draft "A Computational Logic with Quantifiers," intended for inclusion in the user's manual to accompany the new version of the system. Both the user's manual and a paper describing our introduction of quantifiers are in preparation.

Yours,

Bob Boyer

Robert S. Boyer

J Strother Moore

END

FILMED

9-84

DTIC