ON THE EFFECT OF USING AUTOMATED REASONING IN TEACHING DISCRETE MATHEMATICS

Mohammed ALMULLA
Department of Mathematics and Computer, Kuwait University
P. O. Box 5969 Safat, 13060, Kuwait
e-mail: almulla@sci.kuniv.edu.kw

Hans LOEPER
Department of Mathematics and Computer, Kuwait University
P. O. Box 5969 Safat, 13060, Kuwait
e-mail: loeper@sci.kuniv.edu.kw

ABSTRACT

Programs that reason are playing an increasing role in teaching mathematics at the undergraduate level. This paper is concerned with teaching topics in discrete mathematics for computer science students. It aims to increase the likelihood of using computer programs to understand, represent, and solve problems with the help of automated reasoning. Topics of the course "Discrete Mathematics in Computer Science" include logic, set theory, relations and graphs as well as counting techniques. It is the authors' view that computer scientists must have substantial training in using discrete mathematics if they are to understand these topics and use them well.

1 Introduction

The field of automated reasoning is concerned with the ability of computer programs to reason about a given knowledge and deduce a new one. A number of automated reasoning programs (tools) can provide great assistance in solving a wide variety of problems, answering open questions, designing hardware circuits, and verifying correctness of theorems' proofs. Such tools are rich enough to be used in teaching computer science students various mathematical concepts. These concepts include problem representation (in first-order predicate calculus), quantification, simplification, substitution, splitting hard cases into smaller solvable ones, proof justification, and different ways of deduction like resolution and factoring.

The effectiveness of the automated reasoning programs is amply demonstrated by examining their role in answering open questions, designing and/or validating the design of logic circuits, verifying the correctness of proofs and programs, and constructing bases for domains which students need to understand before working vigorously on those domains.

An analysis to general problem solving leads to the identification of three types of problems: numerical, data-processing, and reasoning. Some problems depend on some combination of the three for a solution to be found. Although, most problem-solving programs currently in use focus on the first two types, there do exist programs that reason. Some of these programs are of commercial value, while others are either shareware or freeware. Examples of such programs are OTTER (McCune 1994), GANDALF (Tammet 1997) , SETHEO (Moser *et al.* 1997), and THEO (Newborn 1997). Except the last one, all of the above examples are freeware and can be obtained from the Internet. Any of these programs can be given some axioms and a statement to be shown correct.

The strength of a computer program that is capable of reasoning depends on how the problem being solved is represented, the completeness of rules employed to draw conclusions, and on the effectiveness of the strategies used to control the reasoning process. These three areas - representation, inference rule, and strategy - are key issues to students learning discrete mathematics.

2 Mathematical System: Axioms, Definitions, and Theorems

With respect to representation, first-order predicate calculus give computer science students the skills needed not only to understand and work on theorems, but also to write computer programs in order to verify the correctness of their proofs. Usually, computer applications deal with finite discrete sets of data items such as arrays and files. Notice that even the set of real numbers is finite in the digital world, because of the limited accuracy of their internal computer representation. Therefore, it is important to show the relationship between the notations used in first-order predicate calculus and their equivalent program codes. For example, let the universe of discourse be the finite discrete set $U = \{x_1, x_2, \ldots, x_n\}$.

- $\forall x[x \in U \Rightarrow p(x)]$

The Scope of x starts with the quantifier, therefore x is local in the function **for_all**. The predicate $p(x)$ is passed as a parameter to the function for_all. T_PF is of type pointer to a function with the profile (x: in U) return BOOLEAN;

```
function for_all(p:in T_PF)return BOOLEAN is
 begin
  for x in U loop
   if not p(x)
    then return FALSE;
   end if;
  end loop;
  return TRUE;
 end for_all;
```

- Similarly, $\exists x[x \in U \wedge p(x)]$

```
function for_some(p:in T_PF) return BOOLEAN is
 begin
  for x in U loop
   if p(x)
    then return TRUE;
   end if;
  end loop;
  return FALSE;
 end for_some;
```

The program code that checks if one and only one element x that belongs to the discrete set of data items U satisfies a specific predicate $p(x)$ is given next.

$\exists! x[x \in U \wedge p(x)] \Leftrightarrow \exists x[x \in U \wedge p(x) \wedge \forall y[y \in U \wedge y \neq x \Rightarrow \neg p(y)]]$

Programming the above well-formed formula suggests nesting a new predicate as shown below:

$\exists x[x \in U \wedge p(x) \wedge q(x,p)]$, where $q(x,p) \Leftrightarrow \forall y[y \in U \wedge y \neq x \Rightarrow \neg p(y)]]$.

Note: scope of y is local, bound to q, whereas x and p are free in q.

```
function for_one(p:in T_PF) return BOOLEAN is
 begin
  for x in U loop
   if p(x) and q(x,p)
    then return TRUE;
   end if;
  end loop;
  return FALSE;
 end for_one;

function q(x:in U; p:in T_PF) return BOOLEAN is
 begin
  for y in U loop
   if y/=x and p(y)
    then return FALSE;
   end if;
```

```
  end loop;
  return TRUE;
 end q;
```

or by substitution of the body of the function q for the call of $q(x, p)$ in the function **for_one**:

```
function for_one(p:in T_PF) return BOOLEAN is
 begin
  for x in U loop
   if p(x)
    then for y in U loop
          if y/=x and p(y)
           then return FALSE;
          end if;
         end loop;
         return TRUE;
   end if;
  end loop;
  return FALSE;
 end for_one;
```

The above presented examples suggest combining the theoretical course Discrete Mathematics for Computer Science with hand-on programming experience in the form of laboratory work associated with the course.

3 Background

In the course "Discrete Mathematics in Computer Science", the students usually practice modelling combinatorial puzzles as well as mathematical theories by using the first-order predicate calculus, and then by transforming these models into corresponding computer programs (as shown above) that solve these puzzles. Some of the existing programs employ domain-dependent knowledge in their aim to model the reasoning process. While other programs hope to achieve the same objective by using domain-independent systems for modelling the puzzles. As an example, in this paper we model the theory of natural numbers and ask to prove the commutative property of addition.

The difficulty and/or complexity of mathematics should encourage students to use the assistance of automated reasoning programs. These programs can help in proving theorems, checking proofs, developing conjectures, and answering open questions in various domains such as number theory, set theory, group theory, ring theory, field theory, and lattice theory (McCharen *et al.* 1976; Wos *et al.* 1991). Other involved domains include combinatory logic, finite semigroups, Robbin's algebra, and equivalential calculus (Wos 1993; Wos 1988). Subsequently, many problem sets were developed for the purpose of underlying the theory behind those domains; for instance, the Stickel Test Set (Stickel 1988), the Quaife sets (Quaife 1992a; Quaife 1992b; Quaife 1991), the seventy-five theorems for testing automatic theorem provers (Pelletier 1986), and the TPTP problem set (Sutcliffe 1997). In this paper, we present as an example a fundamental system of well-structured theorems in elementary number theory based on

the axioms provided by G. Peano in 1889, and which gave raise to the Peano Arithmetic. Our theorems range from quite trivial to moderately difficult. A refutation was obtained for each theorem in the system by THEO.

4 The Peano Axioms

This section illustrates how the program THEO proves the commutative law of addition. But first, we need to formalize the Peano axioms in first-order predicate logic. We will compare the proof obtained with the one usually given to students in classrooms. We use two primitive function symbols: **0()** to denote the constant zero (note that a constant is a function with no parameters), and the successor function **s**. The predicate symbol **N** represents the set of natural numbers and the predicate symbol **EQ** means equal. The logical operation OR is represented by the symbol | and the logical operation NOT is represented by the symbol $\sim$. The fifth Peano axiom is concerned with the induction principle. Anything follows a semicolon **;** is considered as a comment.

```
;;;;;;;;;;;;;;;;;;;;;;;;; Peano Axioms ;;;;;;;;;;;;;;;;;;;;;;;;

A1: N(0())

A2: ~N(x) | N(s(x))          ; N(x) --> N(s(x))

A3: ~N(x) | ~EQ(0(),s(x))    ; N(x) --> ~EQ(0(),s(x))

A4: ~EQ(s(x),s(y)) | EQ(x,y) ; EQ(s(x),s(y)) --> EQ(x,y)

A5: ~EQ(x,y) | EQ(s(x),s(y)) ; EQ(x,y) --> EQ(s(x),s(y))
```

The axioms A4 and A5 indicate that the successor **s** is a one-to-one function. The equality relation **EQ** is an *equivalence* relation. This adds three extra axioms to the system.

```
;;;;;;;;;;;;;;;;;;;;;;;;; Equality Relation ;;;;;;;;;;;;;;;;;;;;

A6: EQ(x,x)                        ; (x=x)

A7: ~EQ(x,y) | EQ(y,x)             ; (x=y)-->(y=x)

A8: ~EQ(x,y) | ~EQ(y,z) | EQ(x,z) ; (x=y)&(y=z)-->(x=z)
```

Addition is defined by structural recursion as follows:

$$\{\forall x : x + 0 = x\} \wedge \{\forall x, y : x + s(y) = s(x + y)\}$$

The above well-formed formulae, defining addition over natural numbers, are converted into the first-order predicate clauses in order to introduce the definition to the theorem prover. The conversion algorithm can be found in (Newborn 1997).

```
;;;;;;;;;;;;;;;;;;;;;;;; Addition Axioms ;;;;;;;;;;;;;;;;;;;;;;;;

A9: +(x,y,A(x,y))                              ; Closure property

A10: +(x,0(),x)                                ; x + 0 = x

A11: ~+(x,y,z) | ~+(x,s(y),u) | EQ(s(z),u)     ; x+s(y) = s(x+y)

A12: ~+(x,y,z) | ~+(x,y,u) | EQ(z,u)           ; Uniqueness property 1

A13: ~+(x,y,z) | +(x,y,u) | ~EQ(z,u)           ; Uniqueness property 2
```

Next, we compare the difference between the proof of the commutative law of addition given to students in class (hand-written proof) with that of the theorem prover. Note that the hand-written proof requires the use of associative law of addition: $m + (n + p) = (m + n) + p$.

Hand-written Proof:

m + s(n) = s(m + n) by definition of addition,
n + s(m) = s(n + m) by definition of addition,
since s(n) = 1 + n
We have: m + s(n) = m + (1 + n) = (m + 1) + n by associative law
Thus, m + s(n) = s(m) + n
Which means s(m + n) = s(n + m) as stated above
m + n = n + m. by Peano Axiom A4 QED.

The proof obtained by THEO's is given next:

```
Theorem:CommAddition.thm

Given axioms:

 1# N0
                        } Peano
 2# ~Nx Nsx             }
                        } Axioms
 3: ~EQxy EQsxsy

 4: ~EQsxsy EQxy

 5# ~Nx ~EQ0sx
------------------------------
 6: EQxx
                         } Equality
 7 >~EQxy EQyx           }
                         } Axioms
 8 >~EQxy ~EQyz EQxz
------------------------------
 9 >+xyAxy

10: +x0x
```

```
11 >EQsxy ~+zsuy ~+zux  } Addition
                        } Axioms
12: EQxy ~+zux ~+zuy

13 >~EQxy ~+zux +zuy
------------------------------
14: +0xx                } 0 + x = x

15 >~+axy +xay          } assume a+x = x+a for some constant a

16 >+abc                } let a+b=c, where a, b,& c are constants

17 >+sabd               } let s(a)+b=d

18 >+bsad               } and b+s(a)=d

Negated conclusion:

19S>+sasbk              } assume s(a)+s(b)=k

20S>+sbsal              } and s(b)+s(a)=l

21S>~EQkl               } prove that k = l by contradiction

Inferred clauses: Proof: 25: (21a,8c)  ~EQkx ~EQxl
 26: (25a,7b)  ~EQxl ~EQxk
  27: (26a,11a) ~EQsxk ~+yszl ~+yzx
   28: (27c,15b) ~EQsxk ~+ysal ~+ayx
    29: (28b,20a) ~EQsxk ~+asbx
     30: (29b,9a)  ~EQsAasbk

31: (18a,11b) EQsxd ~+bax
 32: (31b,15b) EQsxd ~+abx
  33: (32a,7a)  EQdsx ~+abx
   34: (33a,13a) ~+abx ~+yzd +yzsx
    35: (34b,17a) ~+abx +sabsx
     36: (35a,16a) +sabsc

37: (16a,11c) EQscx ~+asbx
 38: (37b,9a)  EQscAasb
  39: (38a,13a) ~+xysc +xyAasb
   40: (39b,11c) EQsAasbx ~+yzsc ~+yszx
    41: (40c,19a) EQsAasbk ~+sabsc
     42: (41a,30a) ~+sabsc
      43: (42a,36a) []
```

To begin, the top line is the name of the theorem. The given axioms follow next and then the negated conclusion. Clauses in the proof are printed next. Following each clause number is a :

> to denote the clause is used in the proof.

\# to denote the clause is eliminated during the simplification phase.

S to denote the clause is derived from the negated conclusion.

Each inferred clause is either a binary resolvent or a binary factor. In the former case, the parents of the clause are printed out and then the clause. For example, the first clause in the above proof:

```
25: (21a,8c)  ~EQkx ~EQxl
```

is clause number 25, it was derived by resolving the first literal "a" of clause number 21 with the third literal "c" of clause number 8. Clause 43 is the NULL clause (denoted by []).

5 Conclusion

In addition to providing evidence that automated reasoning has made rigorous contributions in the theoretical foundation of mathematics, the presented work identifies its significance in teaching computer science students various skills needed in discrete mathematics. As it gives students the ability to express knowledge and test its correctness by writing computer programs and analyze their execution time. This is a good opportunity for combining theory with practice while teaching mathematics that can be expressed in first-order predicate calculus.

Acknowledgement: The authors would like to thank Monty Newborn (the author of THEO) for his continuous efforts in working on the field.

REFERENCES

-Hutter, D. 1992. *Synthesis of induction orderings for existence proofs.* 12th CADE, LNCS, Springer.

-McCharen, J., Overbeek, R., & Wos, L. 1976. Problems and experiments for and with automated theorem-proving programs. *IEEE Transactions on Computers*, **C-25** (8).

-McCune, W. 1994.] Otter 3.0 Reference Manual and Guide. *Technical Report ANL-94/6.* Argonne National Laboratory, Argonne, Illinois.

-Mendelson, E. 1987. *Introduction to Mathematical Logic.* Third Edition, Wadsworth & Brooks, Cole Advanced Books & Software, Pacific Grove, California.

-Newborn, M. 1997. *The Great Theorem* Prover. Version 3.0, Newborn Software.

-Moser M., Ibens O., Letz R., Steinbach J., Goller C., Schumann J., & Mayr K. 1997. SETHEO and E-SETHEO: *The CADE-13 Systems, Journal of Automated Reasoning* **18**(2), pp.237-246.

-Pelletier, F. J.1986. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, **2** (1): 191-216.

-Quaife, A. 1992a. Automated deduction in Von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, **8** (1): 91-147.

-Quaife, A. 1992b. *Automated Development of Fundamental Mathematical Theories.* Kluwer Academic Publishers.

-Quaife, A. 1991. Unsolved problems in elementary number theory. *Journal of Automated Reasoning*, **7** (1): 97-118.

-Stickel, M. 1988. A Prolog technology theorem prover: implementation by extended Prolog compiler. *Journal of Automated Reasoning*, **4**: 353-380.

-Sutcliffe, G. & Suttner, C. 1997. The TPTP (Thousands of Problems for Theorem Provers) *Problem Library for Automated Theorem Proving.*
http://www.cs.jcu.edu.au/ftp/pub/research/tptp-library/ReadMe-v2.1.0. -Tammet T. (1997). Gandalf, *Journal of Automated Reasoning* **18** (2), pp.199-204.
-Wos, L. 1993. Automated reasoning answers open questions. *Notices of the AMS* **1**: 15-26.
-Wos, L., & McCune, W. 1991. Automated theorem-proving and logical programming: a natural symbiosis. *Logic Programming*, **11** (1): 1-53.
-Wos, L. 1988. *Automated Reasoning: 33 Basic Research Problems.* Englewood Cliffs, New Jerzy, Prentice-Hall.
-Yu, Y. 1990. Computer proofs in group theory. *Journal of Automated Reasoning*, **6**: 251-286.