

Automated theorem proving in the
Mathematica language

New Mexico Adventures in
Supercomputing Challenge
Final Report
April 7, 2004

Team #68
Santa Fe Preparatory School

Team Members:

Dylan G. Allegretti

Teacher(s):

James C. Taylor

Project Mentor:

James C. Taylor

Table of contents

Executive summary	3
Introduction	4
Project description	5
Results	7
Conclusions	8
Recommendations	9
Acknowledgments	10
References	11
APPENDIX A: Examples	12
APPENDIX B: Program code	14

Executive summary

Automated theorem proving is an important field in computer science that involves using computer programs to prove theorems automatically. Although it is recognized that the subject has little relevance to mathematics, theorem-proving programs occasionally prove interesting theorems, and they have led to efficient programming techniques that are useful in other fields. In this project, a new theorem prover was developed using the *Mathematica* computing software. The program made use of *Mathematica*'s unique computing capabilities to prove theorems very efficiently: The final program proved theorems for a general set of axioms using an algorithm that *Mathematica* reads as only nine lines of code.

In addition to being written in the *Mathematica* language, the theorem prover considered here used a method of proving theorems known as “resolution theorem proving.” This simplifying method, invented by Alan Robinson in 1963, allows proofs to be thought of as sequences of statements about the consistency of the axioms together with the theorem's negation. Theorem provers written in this fashion eliminate inconsistent statements from the given information, attempting to prove the theorem's negation false, and thereby prove the theorem true.

The program developed in this project was successful in proving theorems for a wide variety of logical systems. Experiments were performed in purely logical domains as well as important mathematical fields like number theory and set theory. Proofs in these systems can be very complicated however, because the number of variables involved is very great. Further work on this topic might be to develop more effective methods for proving theorems in these domains.

Introduction

The idea for this project came from a statement made by Stephen Wolfram [5] regarding progress in automated theorem proving. "...unlike systems such as *Mathematica* that emphasize explicit computation none of these efforts have achieved widespread success in mathematics." In this project a new theorem prover was written in the *Mathematica* language. Its purpose was to prove real mathematical theorems in any domain.

There are four reasons why *Mathematica* is so well-suited for programming theorem provers. These all have to do, more or less, with the architecture of the program. *Mathematica* uses the following basic rules [1] for evaluating mathematics:

1. Everything is treated as an expression.
2. Evaluation works by applying rules associated with symbols.
3. Rules are applied until there are no more applicable rules.
4. Parts of expressions are evaluated in a specific order.

In the past, theorem provers were generally written in languages such as Prolog or LISP. No theorem prover to my knowledge has ever taken full advantage of *Mathematica's* highly mathematical syntax.

Project description

In order to prove theorems in conventional logic, computers need some way of following dense sequences of logical implications. In resolution theorem proving, this can be done if the theorem and all of the axioms are first translated into a special form called the *conjunctive normal form*. A statement is in conjunctive normal form if it is a *conjunction*, (AND statement) consisting of one or more conjuncts, each of which is a *disjunction* (OR statement) of one or more literals [3]. In the conjunctive normal form, an implication $a \rightarrow b$ is rewritten as $\neg a \text{ OR } b$.*

Resolution works by searching the axioms for a single axiom that contains any literal found in the theorem's negation. The theorem prover then conjoins the theorem's negation with this axiom and removes contradictory literals. The program repeats this process with the resulting formula until every literal drops out of the formulas leaving an empty set. At this point, the theorem is considered proven.* In order to generate proofs by the resolution method, all statements have to be entered into the theorem prover in conjunctive normal form. This project does not attempt to convert statements to their conjunctive normal forms on the computer.

After the coding for the theorem prover had been perfected, several experiments were run to determine the prover's effectiveness for various axiom systems. The most successful tests were based on different types of syllogistic logic. The most basic type of syllogism consists of a major and minor premise and a conclusion, *e.g.*, "every virtue is laudable; kindness is a virtue; therefore kindness is laudable." Experiments were also performed with well-known axiom systems from mathematics. The systems used were the Zermelo-Fraenkel (**ZF**) system of set theory and the Peano axioms of arithmetic. The particular formalism of arithmetic that was used in this project is due to Wolfram [5] and may be represented as follows:

* For more information, see APPENDIX A: Examples.

(* Peano Axioms Conjunctive Normal Form *)

```

¬ Equal[a + 1, 0];
¬ Equal[a + 1, b + 1] ∨ Equal[a, b];
Equal[a + 0, 0];
Equal[a + (b + 1), (a + b) + 1];
Equal[a × 0, 0];
Equal[a × (b + 1), (a × b) + a];
¬ (Equal[α[0], 1] ∧
    (¬ Equal[α[b], 1] ∨ (Equal[α[b + 1], 1]))) ∨
    Equal[α[b], 1];

```

The problem with using these types of axiom systems is, as will be shown, that they are too general, and their abstractness limits the degree to which the theorem prover can actually search for proofs.

Since the theorem prover was less effective for proving theorems in these systems, other kinds of axioms were attempted. These proved mathematical theorems in ways closer to the syllogistic logic systems described above.

Results

The theorem prover that was developed in this project took advantage of *Mathematica*'s special abilities. The program proved theorems by generating two mathematical lists: *step* and *test*. The list called test contained all of the axioms that the theorem prover sought to unify with each active line in the proof. The list called step contained all of the resulting statements.

Below are some examples from the theorem prover.

1. A theorem prover is presented with the following axioms based on the example proof in Appendix A.

Axioms

```
axioms = {man[Marcus], Pompeian[Marcus], ¬ Pompeian[x1] ∨ Roman[x1], ruler[Caesar],
  ¬ Roman[x2] ∨ loyalto[x2, Caesar] ∨ hate[x2, Caesar], loyalto[x3, f1[x3]},
  ¬ man[x4] ∨ ¬ ruler[y1] ∨ tryassassinate[x4, y1] ∨ ¬ loyalto[x4, y1],
  tryassassinate[Marcus, Caesar]};
replacables = {x1, x2, x3, f1[x3], x4, y1};
```

It correctly reproduces the first four lines of the proof before encountering a glitch.

```
{!hate[Marcus, Caesar]}
{loyalto[Marcus, Caesar], !Roman[Marcus]}
{loyalto[Marcus, Caesar], !Pompeian[Marcus]}
{loyalto[Marcus, Caesar]}
{loyalto[Marcus, Caesar]}
```

In all of the output proofs considered here, the elements of the sets represent literals whose truth value is still in question. At each step the theorem prover unifies these clauses with one of the literals in order to resolve more statements.

2. For a more trivial theorem like **man[Marcus]** a proof is found immediately and the prover returns null for the next line

```
{!man[Marcus]}  
nextstep
```

Conclusions

Although the theorem prover developed in this project had some apparent shortcomings it was successful in proving simple statements and demonstrated the essential properties of a resolution theorem prover. The remaining kinks in the code are problems for the programmer; they are not fundamental flaws. I conclude that *Mathematica* is indeed a powerful language for programming theorem provers. My code is extremely brief, and I believe that more development would make it a very powerful classroom tool for teaching the logical processes of automated theorem proving.

Recommendations

Individuals wishing to reproduce the results of this project should know that the goal of automated theorem proving is not to imitate the human process of proving theorems. It is conceivable to use a resolution theorem prover to prove theorems of set theory and number theory, but not very practical. A good program that can consistently produce proofs for syllogistic logic systems has all the necessary functions for proving theorems.

References

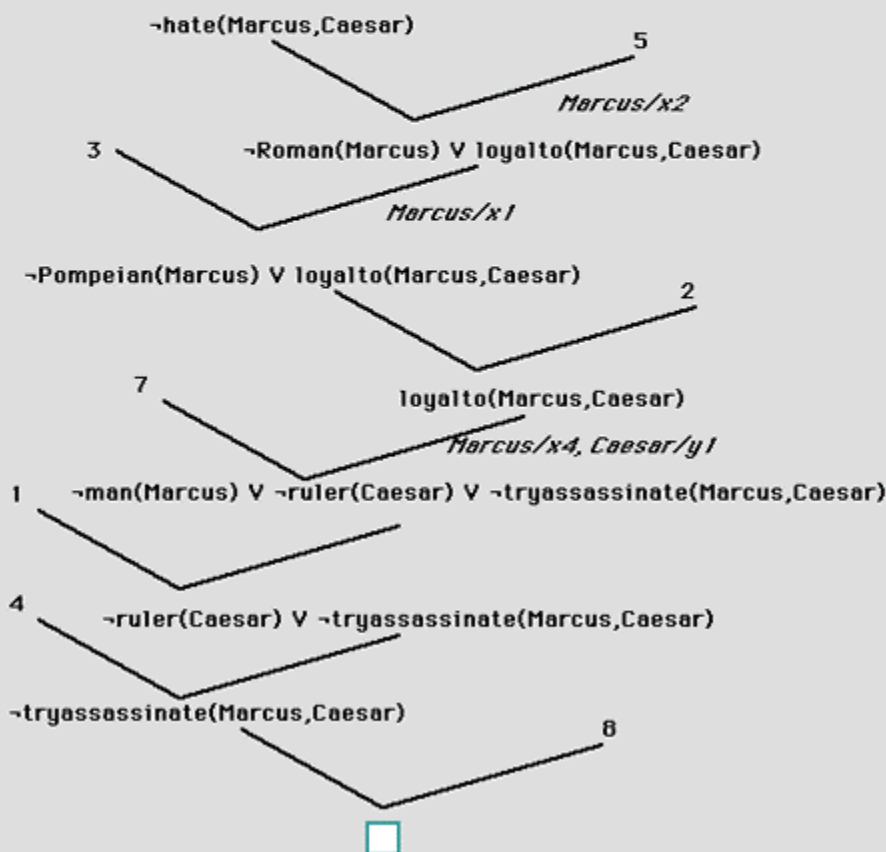
- [1] *M101: A First Course in Mathematica*. Wolfram Research, Inc., 2002.
- [2] Bernays, P. (1968). *Axiomatic Set Theory*. New York, North-Holland Publishing Company.
- [3] Mendelson, E. (1997) *Introduction to Mathematical Logic*. 4th ed. London, Chapman & Hall.
- [4] Rich, E., and Knight, K. (1991). *Artificial Intelligence*. 2nd ed. New York, McGraw Hill. <http://www.rci.rutgers.edu/~cfs/472_html/Logic_KR/resolution.html>.
- [5] Wolfram, S. (2002). *A New Kind of Science*. Winnipeg, Wolfram Media, Inc.
- [6] ---. (2003). *The Mathematica Book*. 5th ed. Wolfram Research Inc.

APPENDIX A: Examples

Database for Resolution Example	
English	
Conjunctive Normal Form	
Standard First Order Logic	
Marcus is a man	$\text{man}(\text{Marcus})$
Marcus is a Pompeian	$\text{Pompeian}(\text{Marcus})$
All Pompeians are Romans	$\neg \text{Pompeian}(x1) \vee \text{Roman}(x1)$ $\forall x \text{ Pompeian}(x) \rightarrow \text{Roman}(x)$
Caesar is a ruler	$\text{ruler}(\text{Caesar})$
All Romans are either loyal to Caesar or hate Caesar	$\neg \text{Roman}(x2) \vee \text{loyalto}(x2, \text{Caesar}) \vee \text{hate}(x2, \text{Caesar})$ $\forall x \text{ Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})$
Everyone is loyal to someone	$\text{loyalto}(x3, f1(x3))$ $\forall x \exists y \text{loyalto}(x, y)$
People only try to assassinate rulers they are not loyal to	$\neg \text{man}(x4) \vee \neg \text{ruler}(y1) \vee \neg \text{tryassassinate}(x4, y1) \vee \neg \text{loyalto}(x4, y1)$ $\forall x \forall y \text{ person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)$
Marcus tries to assassinate Caesar	$\text{tryassassinate}(\text{Marcus}, \text{Caesar})$ $\text{tryassassinate}(\text{Marcus}, \text{Caesar})$

1. man(Marcus)
2. Pompeian(Marcus)
3. \neg Pompeian(x1) \vee Roman(x1)
4. ruler(Caesar)
5. \neg Roman(x2) \vee loyalto(x2,Caesar) \vee hate(x2,Caesar)
6. loyalto(x3,f1(x3))
7. \neg man(x4) \vee \neg ruler(y1) \vee tryassassinate(x4,y1) \vee \neg loyalto(x4,y1)
8. tryassassinate(Marcus,Caesar)

Prove: hate(Marcus,Caesar)



APPENDIX B: Program code

Automated Theorem Prover

by Dylan G. Allegretti

Axioms

```
axioms = {man[Marcus], Pompeian[Marcus], ¬ Pompeian[x1] ∨ Roman[x1], ruler[Caesar],
  ¬ Roman[x2] ∨ loyalto[x2, Caesar] ∨ hate[x2, Caesar], loyalto[x3, fl[x3]},
  ¬ man[x4] ∨ ¬ ruler[y1] ∨ tryassassinate[x4, y1] ∨ ¬ loyalto[x4, y1],
  tryassassinate[Marcus, Caesar]};
replacables = {x1, x2, x3, fl[x3], x4, y1};
```

¬(theorem to be proved)

```
step = Table[0, {8}];
test = Table[0, {8}];
step[[1]] = negTheorem = {! hate[Marcus, Caesar]};
```

Resolution Algorithm

```

Conjuncts[expr_] :=
  If[Length[expr] == 1,
    {expr},
    Extract[expr, Table[{i}, {i, 1, Length[expr]}]]
  ];
Unify[step_, test_] :=
  Do[
    Do[
      If[test[[n]] == ¬ step[[m]],
        nextstep = (Delete[step, m]) ∪ (Delete[test, n])
      ],
      {m, Length[step]}],
    {n, Length[test]};
SearchAxioms[step_] :=
  If[Length[step] == 1,
    If[Length[Conjuncts[axioms[[n]]]] == 1,
      If[Head[step[[1]]] == Not,
        If[Head[axioms[[n]]] == Head[¬ step[[1]]],
          test[[counter]] = Conjuncts[axioms[[n]]]
        ]
      If[Head[¬ axioms[[n]]] == Head[step[[1]]],
        test[[counter]] = Conjuncts[axioms[[n]]]
      ]
    ],
    If[Head[step[[1]]] == Not,
      If[Head[Conjuncts[axioms[[n]]][[m]]] == Head[¬ step[[1]]],
        test[[counter]] = Conjuncts[axioms[[n]]]
      ],
      If[Head[¬ Conjuncts[axioms[[n]]][[m]]] == Head[step[[1]]],
        test[[counter]] = Conjuncts[axioms[[n]]]
      ]
    ]
  ],
  If[Length[Conjuncts[axioms[[n]]]] == 1,
    If[Head[axioms[[n]]] == Head[¬ step[[p]]],
      test[[counter]] = Conjuncts[axioms[[n]]]
    ],
    If[Head[Conjuncts[axioms[[n]]][[m]]] == Head[¬ step[[p]]],
      test[[counter]] = Conjuncts[axioms[[n]]]
    ]
  ]
];
ReplaceVariables[test_] :=
  Do[
    If[Position[test, replacables[[n]]] != {},
      Return[test /. replacables[[n]] → Marcus],
      If[n == Length[replacables],
        Return[test]
      ]
    ],
    {n, Length[replacables]};
counter = 1;
Clear[nextstep]
negTheorem

```

```
]
  ], {n, Length[replacables]}}];
counter = 1;
Clear[nexttest]
Clear[nextstep]
{negTheorem}

For[counter = 1, counter < 4,
  Do[Do[Do[SearchAxioms[step[[counter]]], {m, Length[Conjuncts[axioms[[n]]]}],
    {n, Length[axioms]}], {p, Length[step[[counter]]]}];
  test[[counter]] := nexttest;
  If[counter == 1,
    Unify[{step[[counter]]}, ReplaceVariables[test[[counter]]],
    Unify[step[[counter]], ReplaceVariables[test[[counter]]]];
  ];
  step[[counter + 1]] = nextstep;
  Print[nextstep];
  counter = counter + 1]
```