A Prover for Propositional Logic in Natural (Deduction) Style: Implementation in C

Jun Matsuda
Master's Program in Science and Engineering, University of Tsukuba,
Tsukuba 305, Japan
jun@score.is.tsukuba.ac.jp

Abstract

This paper presents a theorem prover for propositional logic. We implement a prover, written by B. Buchberger in *Mathematica*, in C and connect it with *Risa/Asir* which is the mathematical software system developed by Fujitsu. We describe all details of this implementation including an additional output feature which is different from the "nested cells" output of the prover in *Mathematica*. The time efficiency of these two provers are compared. This work is part of the *Theorema* project, which will be also described briefly in this paper.

1 Introduction

Automated theorem proving is one of the most important areas in computer science and mathematics. There are many researchers who develop and study theorem provers, and various kinds of theorem provers can easily be accessed and used (for example Isabelle [Paulson and Nipkow, 1990]). In spite of this, there are few users who really use these systems when *proving* mathematical statements because the proofs produced by these provers do not reflect a "natural" style of proving. Also, most of these provers are too general and therefore can only prove simple theorems.

In contrast, for *solving* mathematical problems, mathematical software systems like *Mathematica* are heavily used in both education and research.

The purpose of the "*Theorema*" project (see [Buchberger, 1996b],[Buchberger, 1996d], [Buchberger, 1996e],[Buchberger, 1996f]) is to provide a system which helps users to prove mathematical statements in natural language. In more detail, the important features of *Theorema* are:

- A theorem *prover is designed for each subfield* of mathematics.
- The provers proceed by *imitating the proof techniques that humans usually use.*
- The proofs are produced *in natural language* which is easy to read.
- The provers are *implemented in an existing mathematical software system* (namely *Mathematica* 3.0, see [Wolfram, 1996]) for establishing a strong relation between "proving","solving" and "computing"(see [Buchberger, 1996d]).

There are already implemented some provers such as a prover for propositional logic [Buchberger, 1996c], an induction prover for natural numbers [Buchberger, 1996a] and so

on. The other provers are being implemented, like a prover for predicate logic, an induction prover for lists and an induction prover for polynomials, etc. All of them are implemented in *Mathematica* because of its advanced features: many built-in functions for handling formulae, excellent graphical user interface, sophisticated mathematical typesetting, and, most importantly, a rule-based programming style which lends itself to the implementation of the inference rules.

However, there are two reasons why we also want to experiment with a C version of the provers:

- *Speed*: It may turn out that, in the long run, the provers implemented in *Mathematica* are too slow.
- *Availability in Risa/Asir*: Since the Fujitsu company who designed *Risa/Asir* is interested in having the provers of the *Theorema* project available directly in their system (i.e. as subroutines without intermediate file transfer or process-to-process communication for data transfer), we need an implementation of the provers in C.

In this paper, we will describe an implementation of one of the provers of *Theorema*, namely the propositional prover based on an extended version of natural deduction (we call this prover PND-prover), in C. In a later stage of the project, it may be decided that a *Mathematica* to C translator will be implemented in order to automate the process of obtaining C versions of the provers. For this, however, a detailed study of the C code to be produced from a *Mathematica* version, as it is done in the present paper, is a necessary and useful preliminary step.

The basic structure of the PND-prover in *Mathematica* will be conserved also in our C version. But *Risa/Asir* does not hold any good output feature like "nested cells" in *Mathematica*, first we generate a "linear" presentation of the proofs in natural language (which will use, however, systematic indentations for displaying the structure of proofs to simulate nested cells). Next we generate a "hyper-text" presentation by HTML. In a future stage of the project we may also study how the prover can be connected with D-ABDUCTOR [Miue, 1996], which is a graphical user interface system also developed by Fujitsu that could maybe replace the *Mathematica* front-end for producing nested cells.

Timing of *Mathematica* programs versus C programs has not been studied carefully in the literature. The only systematic investigation was so far given in [Buchberger, 1991] where it turned out that, for simple programs (for example, adding two vectors of integers), the ratio of computing time in *Mathematica* versus computing time in C is 1000:1 and more. Hence, it is of enormous interest to see how this ratio behaves in more involved programs (for example the prover described in this paper), in which coarse-grained subroutines of the *Mathematica* C library are heavily used. Therefore, in the last section of our paper, we compare the *Mathematica* and the C version of PND-prover with respect to computing times. The result is that, roughly, our prover in C is 10 times faster than that in *Mathematica*.

2 The PND-Prover in *Mathematica*

In this section, The PND-prover in mathematica is introduced before describing its implementation in C.

2.1 Propositional Logic

In the PND-prover, the logic is restricted to propositional logic. Propositional logic deals with propositions that are built up from elementary propositions that can be *true* or *false*

by the connectives $\neg, \vee, \wedge, \Rightarrow, \Leftarrow$ and $\Leftrightarrow$.

Although propositional logic is quite limited in its expressive power, studying and implementing proof techniques for propositional logic already reveals some fundamental aspects of automated proving.

2.2 Problem

The PND-prover solves the following problem.

- Input:
  - A formula F of propositional logic as goal.
  - A finite sequence A of formulae of propositional logic as assumptions.
- Output:
  - A proof of F from A if F is true under A.
  - A partial proof of F from A that indicates the points of failure if F is not true under A.

2.3 Method

Inference rules are applied to proof situations (consisting of a proof goal and assumptions) until the trivial proof situation (whose goal occurs among the assumptions) is obtained. The particular inference rules used in the PND-prover are the ones that are described in [Buchberger and Lichtengerger, 1981], which expand the usual natural deduction rules given in texts of formal logic (e.g. [Hayashi, 1989]) by many additional rules that make proofs to imitate human proofs in as much detail as possible. (In fact, in [Buchberger and Lichtengerger, 1981] these rules are used for training the proving ability of undergraduate students.) The details of the inference rules are described on Section 2.6. If one wants to implement the fundamental part of the prover (which is the part that generates the abstract proof objects), we just have to implement these inference rules.

We study an example in Figure 1. It shows the proof of the formula $A \Rightarrow C$ under the assumptions $B \Rightarrow C$ and $\neg A \vee B$. (In this figure, the number under "$\rightarrow$" indicates the inference rule which is applied. The **bold** formula is the formula to which the inference rule is applied.)

$$
\begin{array}{lll}
\mathrm{PND}[\mathbf{A} \Rightarrow C, \{B \Rightarrow C, \neg A \vee B\}] & \rightarrow_{(4d)} & \mathrm{PND}[C, \{\mathbf{A}, B \Rightarrow C, \neg \mathbf{A} \vee \mathbf{B}\}] \\
 & \rightarrow_{(3b)} & \mathrm{PND}[C, \{A, \mathbf{B} \Rightarrow \mathbf{C}, \mathbf{B}\}] \\
 & \rightarrow_{(3c)} & \mathrm{PND}[\mathbf{C}, \{A, \mathbf{C}, B\}] \\
 & \rightarrow_{(1a)} & \texttt{proved}
\end{array}
$$

Figure 1: Example of Proof

We have to mention that this figure shows the structure of the proof only. In the PND-prover, all the formulae are labeled (with automatic generation of new labels during the proof) and references to these labels are made when the inference rules are applied. This makes proofs easier to understand.

2.4 The Structure of the PND-Prover

The PND-prover is divided into two parts: The proof generation part and the proof representation part. The proof generation part generates "abstract proof objects" that contain all the relevant information on the proof steps. We show an example of the proof object of Figure 1 below:

```
[[proof by propositional natural deduction,[sample,=>,[A],[C]],
[[1,=>,[B],[C]],[2,or,[not,[A]],[C]]]],[[deductionrule,sample,
sample.C,[C],sample.H,[A]],[[resolution,sample.H,2,2.C,[C]],
[[conclusion is assumption,sample.C,2.C],proved],proved],proved],
proved]
```

The proof presentation part of the prover produces a pretty printing from the abstract proof object. The advantages in splitting the prover into two parts is that we can present proofs in different form by changing the representation part only.

2.5 The Structure of Formula

Propositional formulae are defined below:

$$
\begin{array}{rcl}
< \text{formulae} > & ::= & [\,] \mid < \text{formula} > \{, < \text{formulae} >\} \\
< \text{formula} > & ::= & < \text{proposition} > \mid < \text{truth} > \mid < \text{not} > [< \text{formula} >] \mid \\
 & & < \text{implication} > [< \text{formula} >, < \text{formula} >] \mid \\
 & & < \text{andor} > [< \text{formulae} >] \\
< \text{andor} > & ::= & \wedge \mid \vee \\
< \text{implication} > & ::= & \Rightarrow \mid \Leftarrow \mid \Leftrightarrow \\
< \text{not} > & ::= & \text{not} \\
< \text{proposition} > & ::= & \textit{string} \\
< \text{truth} > & ::= & \text{true} \mid \text{false}
\end{array}
$$

string is any finite sequence of characters. This definition of the formulae is the internal definition. When we use the prover, we may type the formulae in the same way as this is normally done in mathematical books, i.e. [Hayashi, 1989], by using infix notation. What we type is then transformed automatically into the internal form by the *Mathematica* parser.

2.6 Inference Rules

First we introduce some notational conventions used in the formulation of the inference rules. We define *situation* as a pair consisting of a goal and assumptions. One of the inference rules is applied to such a pair and derives one or more new situations in each step.

We use multisets for the formulation of the inference rules. A *multiset* is a collection in which elements are allowed to occur more than once. The notation $\uplus$ is the unordered union of two arguments that are multisets. For example, $\{A, B, C\} \uplus \{B, B, D\} = \{A, B, B, B, C, D\}$.

In the inference rules, we have to mention that:

- The inference rule must be applied in the order given below. (This is different from the convention usually made in logic texts.)
- A **bold** symbol denotes a multiset containing any finite sequence of formulae.
- Assumptions and arguments of $\vee$ and $\wedge$ are lists, not multisets. For describing the inference rules, we use multisets instead of lists since it makes representation simple.
- In some inference rules, two or more individual *situations* are derived. If all the derived *situations* can be proved, then the original *situation* is proved, otherwise it is not proved.

In fact, in *Mathematica*, the rules given on Appendix A can basically be programmed exactly in the way as shown. This is so because *Mathematica* allows a rule-based programming style (with the particular feature of "sequence variables" that allows to handle multisets. In section 3.4, we describe how this feature can be implemented by programs in C).

3 The Implementation of the PND-Prover

In this section we describe the details of implementation of the PND-prover in C.

3.1 *Risa/Asir*

Risa/Asir [Noro and Shimoyama, 1994] is a free mathematical software system developed by Fujitsu. In some respects, it is superior to other mathematical packages as *Mathematica* or Reduce, in particular in the speed of calculating in special areas of algebra like Gröbner bases. It can be divided into two parts, which are called *Risa* and *Asir*:

- *Risa*

  Risa is a library for a computer algebra system. It can treat many kinds of types which are necessary to solve mathematical problems. It also includes a memory manager, which is the public domain software developed by Boehm [Boehm and Weiser, 1988].

- *Asir*

  Asir is a standard language interface for *Risa*. It has the following features:

  - a C-like programming language,
  - a dbx-like debugger,
  - plotting of implicit functions,
  - some useful user-defined functions (e.g. for calling and organizing the Gröbner basis package).

The advantages of developing theorem provers on *Risa/Asir* as compared to plain C are the following:

- *Risa/Asir* includes a parser program. The strings which are typed by the user are parsed and can be translated into an internal list representation.
- *Risa/Asir* contains a memory management.
- We can call C programs directly from within *Risa*.

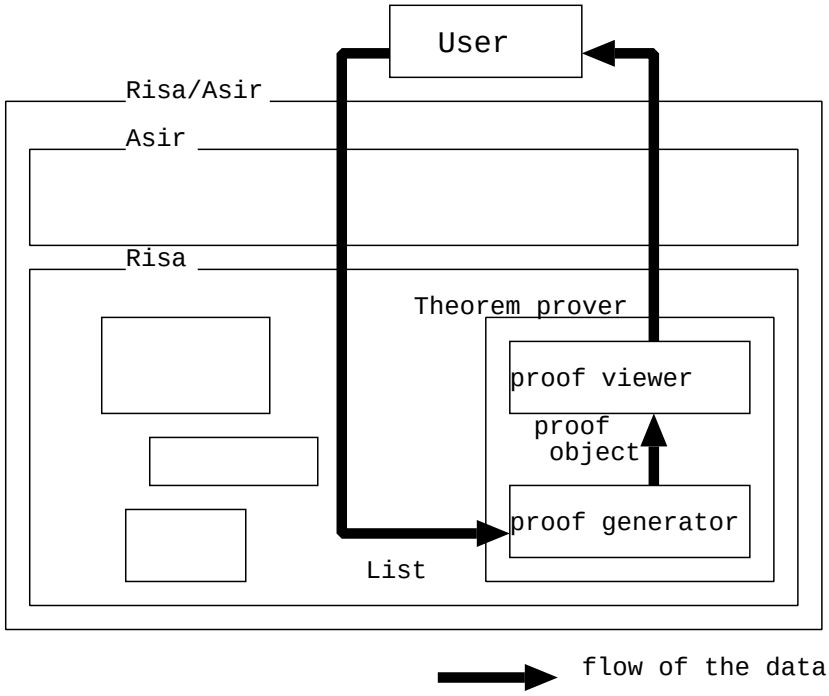

Figure 2: Overview between theorem prover and *Risa/Asir*

3.2 The Structure of the Prover in *Risa/Asir*

The overview of the prover can be seen in Figure 2. The prover is included in *Risa/Asir* but, except for the parser part and the memory manager, it is independent of *Risa/Asir* and, thus, could also be used together with other systems. For *Risa/Asir*, the theorem prover is one of the functions that can be called. As seen in Figure 2, and in accordance with the *Mathematica* implementation, the prover is separated into the "proof generator" and "proof viewer".

3.3 Data Type

Risa/Asir has many kinds of the data types: numbers, polynomials, lists, vectors and so on. There is no extra type for logical formulae and, thus, we encoded formulae as lists. The disadvantage is that we lose the easy readability of the formula. However, this is not really a problem in representation because we will write pretty printing for the proofs anyway. We show the definition of formula on list representation below:

$$
\begin{aligned}
< \text{labeled formula} > &::= [< \text{label}, \text{formula} >] \\
< \text{formula} > &::= < \text{proposition} > \mid < \text{truth} > \mid < \text{not} >, [< \text{formula} >] \mid \\
&\quad < \text{implication} >, [< \text{formula} >], [< \text{formula} >] \mid \\
&\quad < \text{andor} > \{, [< \text{formula} >]\} \\
< \text{andor} > &::= \texttt{"and"} \mid \texttt{"or"} \\
< \text{implication} > &::= \texttt{"=>"} \mid \texttt{"<="} \mid \texttt{"<=>"} \\
< \text{not} > &::= \texttt{"not"}
\end{aligned}
$$

$$
\begin{aligned}
< \text{proposition} > &::= \text{"}string\text{"} \\
< \text{truth} > &::= \texttt{"true"}|\texttt{"false"} \\
< \text{label} > &::= \text{"}string\text{"}
\end{aligned}
$$

We must mention that the double quotation must be applied with strings since all strings without the double quotation are considered to be variables in *Risa/Asir*.

For example, the labeled formula

$$A \Rightarrow A \vee B \tag{1}$$

is presented in *Risa/Asir* as follows:

```
["1","=>",["A"],["or",["A"],["B"]]]
```

It is transformed by *Risa/Asir* into the structure which is very similar to LISP structure. We show the internal structure of the data which includes goal and assumptions on the list in Figure 3.

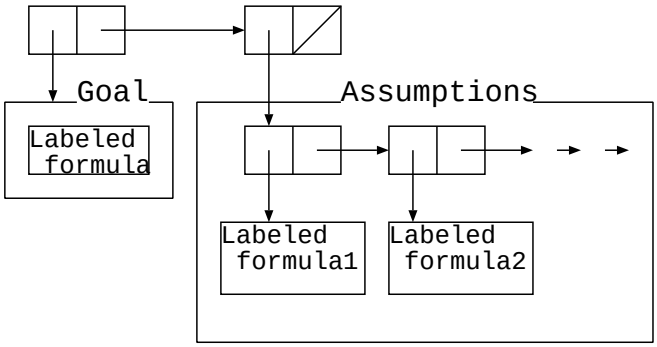

Figure 3: The internal structure of situation in *Risa/Asir*

3.4 Pattern Matching

In *Mathematica* it is easy to write a rule-based program by using ordinary and sequence variables.

- x_: any single expression to be named x (ordinary variable),
- x___: sequence of zero or more expressions named x (sequence variable).

Let us look at the following two programs of the PND-prover. They implement the inference rule (1b).

```
PND[<label_,formula_>,
    <labeledformulae1___,<label1_,formula1_>,
     labeledformulae2___,<label2_,not@formula1_>,
     labeledformulae3___>] :=
     <<"contradictory assumptions",label,label1,label2>,"prove">;
```

```
PND[<label_,formula_>,
    <labeledformulae1___,<label1_,not@formula1_>,
     labeledformulae2___,<label2_,formula1_>,
     labeledformulae3___>] :=
     <<"contradictory assumptions",label,label1,label2>,"prove">;
```

Because there are no sequence variables in C, we must describe this by organizing loops over the elements of the assumptions list. The following is the program of the inference rule (1b) in C:

```
 1   if (FML(arg2)){
 2     for(ntmp21=arg2;ntmp21;ntmp21=CDR(ntmp21)){
 3       if(!strcmp(LF_HSTR(FML(ntmp21)),NOT)){
 4         for(ntmp22=arg2;ntmp22;ntmp22=CDR(ntmp22)){
 5           if (is_sfml(CDR(FML(ntmp22)),FML(LF_A0(FML(ntmp21))))){
 6             infrule1_B(arg,ntmp21,ntmp22,rp);
 7
 8             return ;
 9           }
10         }
11       }
12     }
13   }
```

In the first line we check whether there still exists a formula in the assumption list. In the next two lines, we look for a formula whose head is "not". And in the following two lines, we look for the same formula without "not".

In a similar way, we implement the other rules of the prover.

3.5 The Representation by Normal Text

To simulate the nested cell structure of proofs, we use, instead, indentation. In the following cases, indentation should be done.

- The goal is drastically changed ((4a), (4c) and (4d)).
- The proof situation is divided into more than two situations ((4b), (4e) and (5a)).

Let us prove the following formula using our theorem prover which is based on *Risa/Asir*:

$$A \wedge (B \vee C) \Rightarrow (A \wedge B) \vee (A \wedge C) \tag{1}$$

Here is the output of our C version of the prover in *Risa/Asir*.

```
[1] X;
["1","=>",["and",["A"],["or",["B"],["C"]]],
["or",["and",["A"],["B"]],["and",["A"],["C"]]]
[2] pnd(X);
```

```
Prove
  (1)  (A and (B or C)) => ((A and B) or (A and C))
with no assumptions.

We prove (1) by natural deduction.

We prove (1) by the deduction rule.

.We assume
.  (1.H)  A and (B or C)
.and show
.  (1.C)  (A and B) or (A and C).

.We prove (1.C) by proving the first alternative negating the other(s).

.We assume
.  (1.C.neg.1)  not(A and C).
.We now show
.  (1.C.C)  A and B.

..Formula (1.C.neg.1) is simplified to
..  (1.C.neg.1')  not(A) or not(C).

..From (1.H.1) and (1.C.neg.1') we obtain by resolution
..  (1.C.neg.1'.C)  not(C).

..From (1.C.neg.1'.C) and (1.H.2) we obtain by resolution
..  (1.H.2.C)  B.

..We prove the individual conjunctive parts of (1.C.C):

..Proof of  (1.C.C.1)  A:

...Formula (1.C.C.1) is true because it is identical to (1.H.1).

..Proof of  (1.C.C.2)  B:

...Formula (1.C.C.2) is true because it is identical to (1.H.2.C).
```

We see that the text produced (including the automatically generated labels) is identical to the output of the *Mathematica* version. However the nested cells are replaced by an appropriate number of indentation dots.

3.6 The Representation by Hyper Text

The output of our prover seems to be very weak as compared with the PND-prover using "nested cell". The proofs generated by this prover will be far from easy-to-read representation if they become long and complicated. To solve this problem, we use the *Hyper Text Markup Language* (HTML for short) as output language. The reason why we use is that it can divide a proof into several parts and link them. One can use many different kinds of colors and browse the files in HTML style by using a browser. Hence, by using HTML,

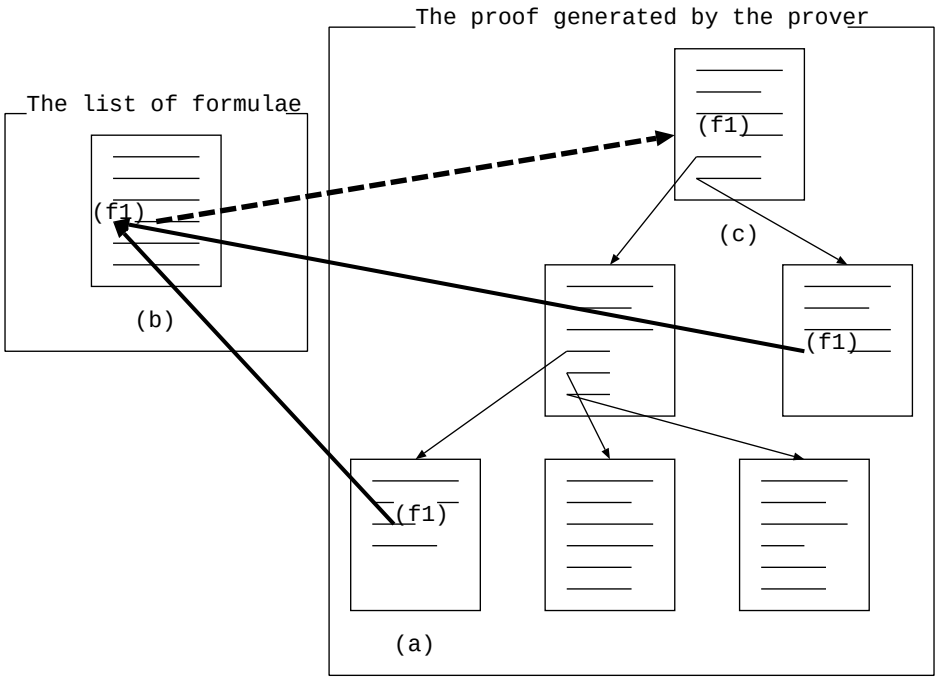

Figure 4: Reference of label

we expect that the easy-to-read proofs are provided. To make the best use of the HTML style output, we add some features.

- We use colors in the following way.
  - normal text (black)
  - label of formula (green)
  - formula (brown)

  This is not essential, but it makes proofs much more readable.

- We separate a proof into parts in order to avoid displaying lengthy proofs in one piece. We divide a proof if the inference rules (4b), (4e), (5a) are applied. These three inference rules derive more than two individual situations when they are applied.

- We produce labels which are linked to the relevant files. We show an example in Figure 4.

  1. When we browse a proof with a reference to a formula(f1), we want to obtain the referenced formula. (a)
  2. In our prover labels are linked to the file which include all the formulae that are appeared in the proof, so we just click the label and find the formula which we look for. (b)
  3. If the formula is clicked in the list, we find the point that is derived. (c)

We show an example in figure 5 to indicate how a proof written in HTML is displayed.

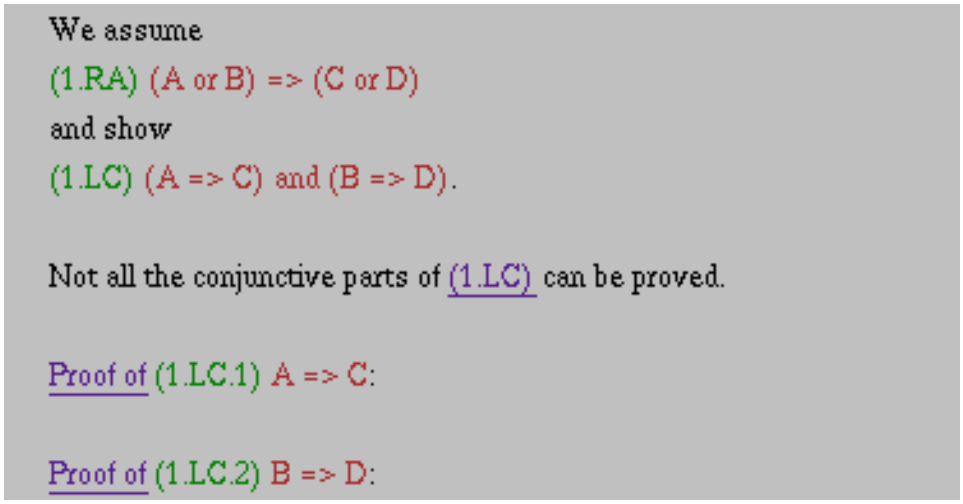

Figure 5:

4 The Comparison between the Two Provers

We used a computer with the following specifications:

- Pentium 133MHz,
- RAM 64Mbyte,
- Windows 95 (for *Mathematica*),
- Linux (for *Risa/Asir*).

To measure the time efficiency of these two provers, we define the following formulae.

$$((A_1 \Rightarrow A_2) \wedge (A_3 \Rightarrow A_4)) \quad \Rightarrow \quad ((A_1 \vee A_3) \Rightarrow (A_2 \vee A_4)) \tag{1}$$

$$(A_1 \Rightarrow (A_2 \Rightarrow A_3)) \quad \Leftrightarrow \quad ((A_1 \wedge A_2) \Rightarrow A_3) \tag{2}$$

We replace some of the propositions $A_1, \ldots, A_4$ by some tautologies. The results are shown in Table 1 using formula (1) and Table 2 using formula (2). By these two tables, we see that PND-prover in C is approximately 8 times to 10 times, faster than that in *Mathematica*.

However, this speed-up is in fact much less than is expected to be possible by switching from a user-defined program in *Mathematica* to the equivalent program in C: In [Buchberger, 1991] it is reported that, for "fine-grain" programs (i.e. programs that do not call any built-in programs of the C library of *Mathematica*) the speed-up by going from *Mathematica* to C can be as much as 1000 times and even more. Also, in [Buchberger, 1997], it is reported that by going from recursion to iteration in *Mathematica*, one can achieve a speed-up by 10 times to 30 times. Thus, the relative small speed-up in the case of this prover must be explained by the fact that the prover in *Mathematica* is a relatively coarse-grain *Mathematica* program: This means that, from within the program relatively efficient C programs of the *Mathematica* library are called and they do most of the job.

In more details, the reasons for the relatively small speed-up gained by the C implementation are the followings:

Propositions	*Risa/Asir*(sec)	*Mathematica*(sec)	*Mathematica* / *Risa/Asir*
8	0.017	0.06	3.53
10	0.017	0.06	3.53
12	0.017	0.06	3.53
15	0.017	0.17	10.00
19	0.103	0.88	8.54
24	0.328	2.14	6.52
26	0.328	2.52	7.68
28	0.414	3.35	8.09
31	0.449	3.85	8.57
35	0.690	6.16	8.92

Table 1: Example 1

- A close analysis of the *Mathematica* program and the C program shows that most of the time spend in both programs is on the analysis of the various proof situations to apply the inference rules. For this, *Mathematica* can use its built-in "pattern matcher", which is a C program that is part of the *Mathematica* kernel. Instead, in our C prover, we have to go through a relatively long sequence of if-then-elses and, in some of the cases within this case analysis, we have to go through loops for simulating the effect of *Mathematica*'s "sequence variables". It seems that the way how the *Mathematica* kernel organizes this pattern-matching process is very efficient. In fact, in [Buchberger, 1997] it is also reported that the use of sequence variables in *Mathematica* is a relatively efficient way of organizing loops when compared with user-programmed loops by iteration.
- The other main part of the time in the *Mathematica* program is spent on calls to coarse-grain built-in functions like "Join". These functions are, again, implemented in C in the *Mathematica* kernel.

Thus, our experiment of implementing a prover in C shows that, for the particular application class of programming provers, *Mathematica* is in fact a relatively efficient tool although, in general, *Mathematica* may be very slow in certain applications. This result of our work is quite unexpected and provides good materials for the future discussion about both the comparative strength and weakness of a high-level language like *Mathematica* and a low-level language like C.

5 Conclusion

We can summarize this thesis as follows:

- We implemented a prover for propositional logic in C taking a *Mathematica* implementation as a model.
- We measured the time efficiency of this implementation.
- We developed a new representation part of this prover using HTML so that we can read the proofs generated by this theorem prover with any WWW browser.

Possible next steps are:

Propositions	*Risa/Asir*(sec)	*Mathematica*(sec)	*Mathematica* / *Risa/Asir*
6	0.017	0.05	2.94
8	0.017	0.11	6.47
10	0.034	0.27	7.94
12	0.086	0.83	9.65
14	0.120	1.04	8.67
19	0.275	2.80	10.20
23	0.842	7.31	8.68
25	0.877	9.34	10.64
27	1.290	12.42	9.62
29	2.080	19.33	9.30

Table 2: Example 2

- Investigation of the reasons of the small speed-up in this prover before constructing a translator from *Mathematica* to C.
- More detailed timing statistic about *Mathematica* and C program.
- A formal termination and completeness proof for the prover.

Acknowledgments

I would like to thank my supervisor Bruno Buchberger for the definition of the task and for suggestions, comments and discussions on this paper. I would like to thank my supervisor Tetsuo Ida for his guidance and encouragement. I'm grateful to Hitoshi Ohsaki for many valuable discussion on my research.

This work was supported by the Advanced Information Technology Program (AITP) of the Information-Technology Promotion Agency (IPA), Japan, in the frame of the project "Coordination Programming in Open Computing Environments" and by a project grant on "Computer Algebra" of Fujitsu Labs, ISIS, Numazu, Japan.

References

[Boehm and Weiser, 1988] H. Boehm and M. Weiser. Garbage collection in an uncooperative environment. *Software Practice & Experience*, pages 807–820, September 1988.

[Buchberger and Lichtengerger, 1981] B. Buchberger and F. Lichtengerger. *Mathematics for Computer Science I - The Method of Mathematics (German).* Springer, Berlin, Heidelberg, New York, 315 S., 2nd edition, 1981. (First Edition 1980).

[Buchberger, 1991] B. Buchberger. Gröbner Bases in Mathematica: Enthusiams and Frustration. In *In Proceedings of IFIP Working Conference on Programming Environments for High-Level Scientific Problem Solving*, pages 80–91, September 23–27 1991.

[Buchberger, 1996a] B. Buchberger. An induction prover for equalities over the natural numbers: implementation in mathematica 3.0. Technical report, Reseach Institute of Symbolic Computation, A4232 Schloss Hagenberg,Austria, October 1996.

[Buchberger, 1996b] B. Buchberger. Mathematica as a rewrite language. In T. Ida et al., editors, *In Proceedings of the Fuji Conference on Functional Logic Programming*, pages 1–13. World Scientific Publishing Co., to appear 1996.

[Buchberger, 1996c] B. Buchberger. A prover for propositional logic in natural (deduction) style: implementation in mathematica 3.0. Technical report, Reseach Institute of Symbolic Computation, A4232 Schloss Hagenberg,Austria, September 1996.

[Buchberger, 1996d] B. Buchberger. Proving, solving, computing: A language environment based on mathematica. In T. Ida and Y. Guo, editors, *In Proceedings of the Multi-Paradigm Logic Programming Conference*. Springer Vienna, September 5 1996.

[Buchberger, 1996e] B. Buchberger. Symbolic computation: Computer algebra and logic. *Frontiers of Combining Systems*, pages 193–220, 1996.

[Buchberger, 1996f] B. Buchberger. Using mathematica for doing simple mathematical proofs. In *In Proceedings of the 4th Tokyo Mathematica Users' Conference*, pages 80–96. Wolfram Media Publishing, November 2 1996.

[Buchberger, 1997] B. Buchberger. A note on computing times in different programming style in mathematica. Technical report, Reseach Institute of Symbolic Computation, A4232 Schloss Hagenberg,Austria, February 1997.

[Hayashi, 1989] S. Hayashi. *Mathematical Logic*. COLONA publishing coorporation LTD, Tokyo, Japan, 1989.

[Miue, 1996] K. Miue. *D-ABDUCTOR 2.0 User Manual*. International Institute for Advanced Study of Social Information Science, FUJITSU LABORATORIES LIMITED, Numazu, Japan, May 1996.

[Noro and Shimoyama, 1994] M. Noro and T. Shimoyama. *Asir User's Manual:Edition 2.0 for Asir-94-0420*. International Institute for Advanced Study of Social Information Science, FUJITSU LABORATORIES LIMITED, Numazu, Japan, May 1994.

[Paulson and Nipkow, 1990] L. Paulson and T. Nipkow. *Isabelle Tutorial and User's Manual*. ftp://ftp.cl.cam.ac.uk/ml/, 1990.

[Wolfram, 1996] S. Wolfram. *The Mathematica book, Third edition*. Wolfram Media Inc and Cambridge University Press, 1996.

A Inference rules for propositional logic

The followings are the inference rules for propostional logic which is based on the PND-prover.

Rules for the trivial situations

$$\frac{\mathrm{PND}[\, X, \mathbf{A} \uplus \{X\}\,]}{\texttt{Proved}} \quad (1a)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X, \neg X\}\,]}{\texttt{Proved}} \quad (1b)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\phi]\}\,]}{\mathrm{PND}[\, G, \mathbf{A}\,]} \quad (1c)$$

$$\frac{\mathrm{PND}[\, \vee[\phi], \mathbf{A}\,]}{\texttt{Proved}} \quad (1d)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\wedge[\mathbf{X} \uplus \{Y, Y\}]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\wedge[\mathbf{X} \uplus \{Y\}]\}\,]} \quad (1e)$$

$$\frac{\mathrm{PND}[\, \wedge[\{X\}], \mathbf{A}\,]}{\mathrm{PND}[\, X, \mathbf{A}\,]} \quad (1f)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X, X\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X\}\,]} \quad (1g)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X} \uplus \{Y, Y\}]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X} \uplus \{Y\}]\}\,]} \quad (1h)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\{X\}]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X\}\,]} \quad (1i)$$

$$\frac{\mathrm{PND}[\, \vee[\{X\}], \mathbf{A}\,]}{\mathrm{PND}[\, X, \mathbf{A}\,]} \quad (1j)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\texttt{False}\}\,]}{\texttt{Proved}} \quad (1k)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg\texttt{True}\}\,]}{\texttt{Proved}} \quad (1l)$$

$$\frac{\mathrm{PND}[\, \texttt{True}, \mathbf{A}\,]}{\texttt{Proved}} \quad (1m)$$

$$\frac{\mathrm{PND}[\, \texttt{False}, \{\texttt{True}\}\,]}{\texttt{False}} \quad (1n)$$

$$\frac{\mathrm{PND}\,[\texttt{False}, \phi\,]}{\texttt{False}} \quad (1o)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\texttt{True}\}\,]}{\mathrm{PND}[\, G, \mathbf{A}\,]} \quad (1p)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg\texttt{False}\}\,]}{\mathrm{PND}[\, G, \mathbf{A}\,]} \quad (1q)$$

$$\frac{\mathrm{PND}[\, \Leftarrow[X, Y], \mathbf{A}\,]}{\mathrm{PND}[\, \Rightarrow[Y, X], \mathbf{A}\,]} \quad (1r)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\Leftarrow[X, Y]\}\,]}{\mathrm{PND}[G, \mathbf{A} \uplus \{\Rightarrow[Y, X]\}\,]} \quad (1s)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\wedge[\mathbf{X}]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \mathbf{X}\,]} \quad (1t)$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\Leftrightarrow[X, Y]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\Rightarrow[X, Y], \Rightarrow[Y, X]\}\,]} \quad (1u)$$

Rules for negated formulae as assumptions

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg\neg X\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X\}\,]} \tag{2a}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg[\wedge[\{X_1, \ldots, X_n\}]]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\{\neg X_1, \ldots, \neg X_n\}]\}\,]} \tag{2b}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg[\vee[\{X_1, \ldots, X_n\}]]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg X_1, \ldots, \neg X_n\}\,]} \tag{2c}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg[\Rightarrow[X, Y]]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X, \neg Y\}\,]} \tag{2d}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg[\Leftrightarrow[X, Y]]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\{\wedge[\{\neg X, Y\}], \wedge[\{X, \neg Y\}]\}]\}\,]} \tag{2e}$$

Rules for resolution and modus ponens

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X} \uplus \{Y\}], \neg Y\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X}], \neg Y\}\,]} \tag{3a}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X} \uplus \{\neg Y\}], Y\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\vee[\mathbf{X}], Y\}\,]} \tag{3b}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X, \Rightarrow[X, Y]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{X, Y\}\,]} \tag{3c}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{Y, \Rightarrow[X, \neg Y]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{Y, \neg X\}\,]} \tag{3d}$$

$$\frac{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg Y, \Rightarrow[X, Y]\}\,]}{\mathrm{PND}[\, G, \mathbf{A} \uplus \{\neg Y, \neg X\}\,]} \tag{3e}$$

Rules for handling compound goals

$$\frac{\mathrm{PND}[\ \neg X, \mathbf{A}\ ]}{\mathrm{PND}[\ \texttt{False}, \mathbf{A} \uplus \{X\}\ ]} \tag{4a}$$

$$\frac{\mathrm{PND}[\ \wedge[\{X_1, \ldots, X_n\}], \mathbf{A}]}{\mathrm{PND}[\ X_1, \mathbf{A}]\ \ \cdots\ \ \mathrm{PND}[X_n, \mathbf{A}]} \tag{4b}$$

$$\frac{\mathrm{PND}[\ \vee[\{X_1, X_2, \ldots, X_n\}], \mathbf{A}\ ]}{\mathrm{PND}[\ X_1, \mathbf{A} \uplus \{\neg X_2, \ldots, \neg X_n\}\ ]} \tag{4c}$$

$$\frac{\mathrm{PND}[\ \Rightarrow[X, Y], \mathbf{A}\ ]}{\mathrm{PND}[\ Y, \mathbf{A} \uplus \{X\}\ ]} \tag{4d}$$

$$\frac{\mathrm{PND}[\ \Leftrightarrow[X, Y], \mathbf{A}\ ]}{\mathrm{PND}[\ Y, \mathbf{A} \uplus \{X\}\ ]\quad \mathrm{PND}[\ X, \mathbf{A} \uplus \{Y\}\ ]} \tag{4e}$$

Rules for general situation of disjunction or implication as assumption

$$\frac{\mathrm{PND}[\ G, \mathbf{A} \uplus \{\vee[\{X_1, \ldots, X_n\}]\}\ ]}{\mathrm{PND}[\ G, \mathbf{A} \uplus \{X_1\}\ ]\ \ \cdots\ \ \mathrm{PND}[\ G, \mathbf{A} \uplus \{X_n\}\ ]} \tag{5a}$$

$$\frac{\mathrm{PND}[\ G, \mathbf{A} \uplus \{\Rightarrow[X, Y]\}\ ]}{\mathrm{PND}[\ G, \mathbf{A} \uplus \{\vee[\{\neg X, Y\}]\}\ ]} \tag{5b}$$

Rule for the default situation

$$\frac{\mathrm{PND}[\ G, \mathbf{A}\ ]}{\texttt{NotProved}} \tag{6}$$