# Yale University
# Department of Computer Science

**Experience with Two Systems Supporting Proofs**

Dana Angluin
Yale University

Michael Bell
Yale University

YALEU/DCS/TR-1278
May 2003

# Experience with Two Systems Supporting Proofs

Dana Angluin
Yale University

Michael Bell
Yale University

**Abstract**

We report on our initial experience exploring Mizar and Isabelle, two quite different systems that support constructing and checking mathematical proofs. We also discuss some issues in the design of such systems.

## 1 Introduction

As ubiquitous as computers have become in our everyday lives, in many respects the computer revolution is still in its earliest phases. Martin Davis has made a persuasive case that mathematics, particularly logic, is the father of computing [1]. However, efforts to integrate the use of computers into the practice of mathematics, while vigorous, varied, and interesting, cannot be described as a mature technology.

Part of the bread-and-butter activities of those who practice mathematics is the construction, communication, and checking of proofs. This note is a report on our initial experience with two publically available systems, Mizar and Isabelle, intended to support these mathematical activities. We are not logicians, and in both cases we used the system actively for a few weeks and a few hundred lines of proof, so our observations are those of novices. However, because human beings rapidly habituate themselves (to the point of blindness) to the quirks of artificial systems, we believe it is useful to record our novice observations and discuss their design implications.

## 2 Mizar

Mizar[1] consists of a formal language for expressing mathematical definitions, theorems, and proofs, and an automated system for checking the correctness of proofs in the language. The Mizar project includes an extensive library of verified definitions, theorems, and proofs available in the on-line articles of the *Journal of Formalized Mathematics*.[2]

Mizar's formal language was designed to be close to the "mathematical vernacular," the informal but reasonably precise usage of current mathematical practice. Figure 1 presents a simple theorem and proof from the 1990 Mizar article "Factorial and Newton coefficients" by Rafał Kwiatek [2].

[1] Main website: www.mizar.org.

[2] The *Journal* has been published since 1989 and included over 2,000 definitions of mathematical concepts and over 30,000 theorems as of August 2002.

```
theorem
Th15: for s holds 1|^s = 1
 proof
  A1: 1|^0 = 1 by Th9;
  A2: for s st 1|^s = 1 holds 1|^(s+1) = 1
   proof
    let s;
    assume  1|^s = 1;
    then 1|^(s+1) = 1 * 1 by Th11;
    hence thesis;
   end;
  thus thesis from Nat_Ind(A1,A2);
 end;
```

Figure 1: Example Mizar theorem and proof

This is readily understandable to someone familiar with English and current mathematical usage. Declarations and definitions elsewhere establish that `s` is a natural number and that the character combination `|^` denotes an infix power function. The theorem then states that for all natural numbers $s$, $1^s = 1$. This is a low-level fact that must be proved in any comprehensive formal development.

The proof is by induction over the natural numbers, invoked by the next-to-last line, which gives the labels of statements of the base case ($1^0 = 1$) and the inductive step (if $1^s = 1$ then $1^{s+1} = 1$). The base case is justified by an earlier theorem in the same article. The inductive step has a subproof that assumes the hypothesis and applies an earlier theorem and some implicit arithmetic simplification to establish the conclusion.

In addition to being close to informal mathematical usage, this sequence of characters has been processed by the Mizar system in the course of verifying the article, which has been incorporated into the Mizar mathematical library. This result can be used in the justification of further theorems; from other articles it is specified by the label `NEWTON:15`.

## 2.1 Using Mizar

Not content with just reading Mizar, we downloaded and installed the system (including the available emacs mode) and attempted some proofs of our own. The recommended method of learning Mizar is to spend a month in Poland with the main Mizar community, so our approach may be seen as a bit rash. One casualty of our approach was that we never found a description of the approved procedure for defining new notation, predicates, and functions. However, the Mizar library provides a rich source of notation and definitions, so this obstacle was not fatal.

The available documentation rapidly cleared up any confusion about the syntax of the language, though the semantics appeared to have no comparably complete documentation. Taking advantage of the emacs mode, we used the system in an interactive style, adding a line to a proof and then

running the proof checker to see what error message(s) were produced. This feedback loop allowed us to begin to understand by example what proof steps would be accepted. We found it quite useful to be able to direct the proof checker to accept certain assertions without proof, for example, to check the feasibility of a particular overall structure for a proof.

The underlying proof system is natural deduction, and although natural deduction mimics some aspects of informal mathematical proof, it became clear that we needed a more detailed and formal understanding of both natural deduction and the specific implementation of it in the Mizar system. For example, informally speaking, to prove an implication one may assume the hypothesis and show that the conclusion follows. More formally, if the hypothesis is a conjunction, one may assume each of the conjuncts individually. In Mizar, the conjuncts must be assumed in the left-to-right order in which they appear, otherwise, the result is the error message "invalid assumption."

At each point in a Mizar proof, the proof-checker has a current value for the "thesis," the statement that remains to be proved. If the system is used in an interactive style, it would be helpful to be able to inspect the value of the thesis at any particular point in the proof.

### 2.2 Searches

Most steps of a Mizar proof require an explicit justification in terms of axioms, definitions or theorems from the library (or results proved earlier in the same article.) Hence it is important to be able to locate relevant items in the library. Proofs from the library may also serve as models for the construction of new proofs. Search tools to assist the user in finding related results and proofs would be particularly helpful to the novice, who is largely unfamiliar with the contents of the library. However, as on-line libraries grow to include a significant fraction of all of mathematics, automated search tools will be helpful to the expert as well.

To get some sense of what kinds of searches might be feasible over the Mizar library, we constructed a simple tool to extract the statements of theorems from the ASCII library files and tried various regular expression searches over the full library files and the extracts.

In the simplest case, a user may know the name of a theorem and wish to find examples of its use. For example, a search for the theorem name `REAL_1:54` found 412 references in the full library: clearly a useful theorem.

```
theorem :: REAL_1:54
  x-z<=y-z implies x <= y;
```

Another simple kind of search involves a specific keyword. For example, a search for the string `choose` in the extract of all theorems matched the following lines.

```
0 choose 0 = 1
for s holds (s choose 0) = 1
(for r st r = s-t holds (s choose t) = (s choose r))
for s holds s choose s = 1
(((t+1) choose (s+1)) = (t choose (s+1)) + (t choose s)
```

```
& ((t+1) choose (s+1)) = (t choose s) + (t choose (s+1)))
for s st s >= 1 holds s choose 1 = s
for s,t st s>=1 & t = s-1 holds s choose t = s
for s holds (for r holds (s choose r) is Nat)
holds F.i = l choose n) holds Sum F = (n+m) choose (n+1)
for k,n st k <= n holds n choose k >= ((n+1) choose k) / (n+1)
(n choose (k+1))=((n-k)/(k+1))*(n choose k)
((x,y) In_Power n).(k+1)=(n choose k)*(x.^.(n-k))*(y.^.k)
```

This suggests that there are not many theorems in the library about binomial coefficients, unless they use some other notation.

As another experiment, we imagined that there might be a theorem something like

```
i + k < j + k implies i < j
```

somewhere in the library, and tried to devise an appropriate search. Of course, the variable names, the direction and strictness of the inequalities, and the spacing might all be different, so a literal search for this string is not likely to succeed. Ideally, a search might be based on the semantics of the formulas involved. However, the following simple syntactic search was surprisingly effective:

```
grep "+.*<.*+.*implies.*<"
```

This matches lines with the features: plus, less than, plus, implies, less than, in that order and separated by any characters. The following lists all the matching lines from the extract of all theorems; the second line is closest to our original target.

```
x + y <' +infty implies x <> +infty & y <> +infty
x+z<=y+z implies x <= y
(a+-b<0 or b-a>0 or -a+b>0 or a-e<b+-e or a+-e<b-e or e-a>e-b) implies a<b
a+b<=e+d implies a-e<=d-b
a+b<e+d implies a-e<d-b
(a<0 implies a+b<b & b-a>b) & (a+b<b or b-a>b implies a<0)
(a<=0 implies a+b<=b & b-a>=b) & (a+b<=b or b-a>=b implies a<=0)
K+`M <` K+`N implies M <` N
```

We concluded that even a simple search tool would be helpful, especially if it could take advantage information about types and semantics. However, we did not continue this line of investigation because we were reluctant to duplicate the effort of a full parser for Mizar.

### 2.3 Pascal's Triangle

We attempted our own proof of the theorem that the sum of row $n$ of Pascal's Triangle is $2^n$. In mathematical notation:

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n.$$

```
reserve  i, j, for Nat;
reserve n for natural number;
reserve F for FinSequence of REAL;

theorem Th1:
  for n holds
        (for F st (len F = n + 1) &
                  (for i, j st (i in dom F) &
                               (j = i - 1)
                   holds
                   (F.i = n choose j))
         holds
         Sum F = 2 |^n)
```

Figure 2: Our Mizar statement of the Pascal Triangle theorem

Our Mizar statement of the theorem is in Figure 2.

We were initially unaware that this theorem is proved as a corollary of Newton's binomial theorem, which is the main theorem of Kwiatek's article [2]. Also, our first guess of the difficulty of a Mizar proof was a serious underestimate.

Some background may illuminate our Mizar formulation of the theorem. Mizar is based on set theory; every object in the theory is ultimately a set. The development of the natural numbers, integers, rationals, reals, and complex numbers is the one familiar to most mathematicians. For example, 0 is literally the set $\{\}$ and the natural numbers is literally the set:

$$\{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}, \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\}, \ldots\}$$

though of course less cumbersome notation is introduced.

Binary relations are sets of ordered pairs, and functions are binary relations with the restriction that no two pairs have the same first element. The domain of a function `F` is `dom F`, and `F.i` denotes the value of `F` on argument `i`. A finite sequence of reals `F` is a function from an initial segment of the positive natural numbers to the real numbers; the cardinality of its domain is `len F`. The `Sum` function returns the sum of a finite sequence of real numbers. Thus, the theorem statement says that for all natural numbers $n$, if $F$ is the finite sequence of numbers

$$\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$$

then the sum of $F$ is $2^n$.

One complication is that the `choose` function is defined for natural numbers and the natural numbers are not closed under subtraction; as a result Mizar's type system is not happy with the expression `n choose (i - 1)`. This is why we use the auxiliary variable `j` in the theorem statement above. Our experience suggests that a more explicit type system would be quite helpful to the user.

Our informal proof of the Pascal Triangle theorem is a simple induction on $n$ that relies on one central lemma (that an entry in row $n$ is the sum of the adjacent two entries in the previous row) and appropriate manipulation of the summation operator. The central lemma is available in Kwiatek's article [2]:

```
theorem :: NEWTON:32
 for s,t st s<t holds
        (((t+1) choose (s+1)) = (t choose (s+1)) + (t choose s)
        & ((t+1) choose (s+1)) = (t choose s) + (t choose (s+1)));
```

The remaining work, of splitting off the two end terms, applying the lemma, rearranging the sum into two sums, shifting indices appropriately, and incorporating the two end terms again, is relatively straightforward but unexpectedly tedious; it forms the bulk of the three and a half page proof of Newton's Binomial Theorem in Kwiatek's article [2].

The summation operator permits considerable economy of thought and expression in mathematical reasoning. Mathematical concepts and notations have been developed over thousands of years; it is important to find ways of preserving or enhancing their power in systems designed to assist a user with mathematical proofs.

Mizar's proofs are somewhat reminiscent of assembly-language programs: capable of nearly anything in the right hands, given enough patience and organizational discipline in the management of low-level detail. However, just as programming was not widely practiced before the advent of higher-level languages, proof assistants will probably have to support a higher-level approach to become widely used.

## 3 Isabelle

Isabelle[3] is a proof assistant capable of supporting a variety of logics. We used the system as specialized for higher order logic (HOL), which has a substantial library of definitions, lemmas, theorems, and proofs, and a helpful tutorial text [3]. Here is how the main lemma for the Pascal Triangle theorem appears in the HOL library file Power:

```
theorem binomial_Suc_Suc:
  Suc n choose Suc k = n choose k + (n choose Suc k)
```

Here `binomial_Suc_Suc` is the label of the theorem, `n` and `k` are natural numbers, and `Suc` is the successor operation. Thus, the theorem states that

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}.$$

[3]Main website: www.cl.cam.ac.uk/Research/HVG/Isabelle.

### 3.1 Proofs in Isabelle

Isabelle is designed to be used interactively in the development of a theory, that is, a related collection of definitions, lemmas, theorems, and verified proofs. The resulting proofs do not resemble the current mathematical vernacular. For example, a statement and proof that $1^n = 1$ for all natural numbers $n$ look as follows on the page:

```
lemma power_1: "(1 :: nat)^(n :: nat) = 1";
apply(induct_tac n);
apply(auto);
done;
```

The type notation `(n :: nat)` declares that `n` is of type natural number, which is required in this case by the overloading of numeric constants and operators. The lines of the proof are directives to the proof checking system rather than proof steps in the traditional sense. To the human reader who is somewhat familiar with Isabelle, they convey that the proof is by induction on the variable $n$ and some automatically discovered steps.

By interactively checking the proof a line at a time, the user can get more information about it. Perhaps this interactive process is the analog in this setting of reading a proof on paper. We record the successive steps of the interactive verification:

```
> lemma "(1 :: nat)^(n :: nat) = 1";
proof (prove): step 0
fixed variables: n
goal (lemma, 1 subgoal):
1 ^ n = 1
 1. 1 ^ n = 1
```

At this point, the system has accepted the statement of the lemma as syntactically correct and set it up as the only required subgoal.

```
> apply(induct_tac n);
proof (prove): step 1
fixed variables: n
goal (lemma, 2 subgoals):
1 ^ n = 1
 1. 1 ^ 0 = 1
 2. !!n. 1 ^ n = 1 ==> 1 ^ Suc n = 1
```

The induction tactic has used the inductive definition of the natural numbers to replace the previous subgoal with two new subgoals: the base case ($1^0 = 1$) and the inductive step (if $1^n = 1$ then $1^{n+1} = 1$.)

```
> apply(auto);
```

```
proof (prove): step 2
fixed variables: n
goal (lemma):
1 ^ n = 1
No subgoals!
```

The auto tactic has succeeded in proving both subgoals, so no subgoals remain.

```
> done;
lemma 1 ^ ?n = 1
```

The final line indicates the end of the proof, and the now verified lemma is restated with a schematic variable `?n` signifying that appropriate substitution instances of the lemma are true. Once a theorem or lemma is verified, it may be used in subsequent proofs.

While this line by line development conveys some more information about the proof to the user, it seems that there is no convenient means to investigate the verification found by the auto tactic.[4] Of course, the user could attempt a different proof using less powerful individual steps, but it is tantalizing that the system has found a detailed proof that cannot be inspected.

As in Mizar, the underlying proof framework is natural deduction, which is covered in a chapter of the tutorial. The preferred style is backward, that is, reducing goals to subgoals until all the subgoals are trivially satisfied. This direction is perhaps more natural for proof discovery and informal exposition, but printed mathematical proofs are often presented in forward style, that is, starting with simple truths and building up to more complex ones. Some system support is offered for a forward proof style.

### 3.2 An Isabelle Example

Isabelle/HOL offers direct support for defining and reasoning about functional programs. For example, here is our recursive definition of a sum operator.

```
consts sum_start_len :: "nat => nat => (nat => nat) => nat";
primrec
"sum_start_len i 0 f = 0"
"sum_start_len i (Suc n) f = f(i + n) + (sum_start_len i n f)";
```

This operator takes a starting value $i$, a length $n$, and a function $f$ and sums up the $n$ values:

$$f(i), f(i+1), \ldots, f(i+n-1).$$

The following lemma (with its complete proof) states that the sum of the first $n$ odd positive numbers is $n^2$. In this expression `%` is the ASCII representation of lambda, so `(%i. i + i - 1)` denotes the function that maps $i$ to $2i-1$.

[4] The tutorial indicates that the steps of the simp tactic can be traced.

```
lemma "sum_start_len 1 n (%i. i + i - 1) = n * n";
apply(induct_tac n);
apply(auto);
done;
```

# 4 Discussion

Clearly we have only scratched the surface of these two very interesting systems for proof writing and checking. Below, we summarize several issues uncovered in our initial attempts to use these systems.

We used both systems in a heavily interactive mode, in part because we were exploring their capabilities. With Mizar, sufficient familiarity with how the checker works might eventually make the success or failure of a proposed proof step predictable to the user, removing some of the motivation for interactive use. By contrast, in Isabelle it seems very difficult to predict what subgoals an application of the built-in tactics simp, auto, or blast will produce from a given goal, making interactive use essentially unavoidable. In general, the capacity for interactive proof construction seems desirable, at least for the novice user. Mizar lacks a useful indicator of the current proof state, which Isabelle supplies in the form of a list of subgoals. A facility like Mizar's `@proof` to indicate to the system to treat an assertion as proved is quite useful in developing and checking the overall structure of a proof.

The readability of constructed proofs is a very important issue for the human user of a proof assistant. The design of Mizar is constrained to make proofs that "read" like informal mathematical proofs on paper. This produces the illusion that Mizar proofs are very similar to informal proofs, of which the novice user is disabused when attempting to construct Mizar-acceptable proofs from informal proofs. One thing that detracts from the readability of Mizar proofs is their tendency to contain a very large number of very small steps. Some facility to view a proof at different levels of detail would perhaps help this.

Isabelle proofs have the opposite tendency – they are often a very small number of very large steps, and the very large steps are often "opaque" to the user, in the sense that the user often cannot effectively resolve the step to the next lower level of detail, an operation for which there is largely no system support. To augment the bare sequence of directives to the checker in Isabelle, it helps to add (as comments) the intermediate lists of subgoals generated in the proof. This restores some of the missing detail, but in general there is no way to resolve the steps into sub-steps. This task is complicated by the fact that the proof steps found by automatic methods may in general be unintelligible to a human user. A system like Isabelle also suggests a different interpretation of "reading" a proof; it may be an interactive process of selectively looking at intermediate goal states, rather than a line-by-line scanning of a static proof on paper.

It would be very desirable for a proof system to have a simple, powerful and accessible intermediate representation of formulas and partial proofs. Mizar's representation system requires a lot of parsing, which seems to have discouraged the development of auxiliary tools for the system. A clear semantics and explicit type system for the representations used by the system, as well a clear

account of the capabilities of the checker, would be very desirable. It would be helpful to be able to control the automatic steps in a flexible way, so that their effective granularity could be varied for different purposes. Also desirable would be the ability to capture in a relatively natural way existing mathematical notations, for example, summation, that have been developed and refined over centuries of use.

For the benefit of novice users, interactive tutorial documentation and sophisticated methods of searching the libraries for related definitions, theorems, and proofs would be very helpful. Searches should be possible using both keywords and the semantics of mathematical statements and proofs.

We also note certain pedagogical issues for computer science and other fields. Should we be teaching our students a standard form of natural deduction, the underlying proof formalization used in both Mizar and Isabelle? To what extent should students be exposed to proof assistants in their courses? Such experience could help students develop their own abilities to read and construct proofs. It could also familiarize them with what is likely to become part of the standard toolkit of workers in the mathematical sciences. Usability issues would be central to any such effort.

## 5 Acknowledgements

The authors thank Carsten Schürmann and Drew McDermott for their suggestions and encouragement.

## References

[1] Martin Davis. *The Universal Computer: The Road from Leibniz to Turing.* Norton, 2000.

[2] Rafał Kwiatek. Factorial and Newton coefficients. *Journal of Formalized Mathematics*, 2, 1990. URL: http://mizar.org/JFM/Vol2/newton.html.

[3] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.