A comparison of Mizar and Isar

Markus Wenzel (wenzelm@in.tum.de) *
Institut für Informatik, Technische Universität München, Germany

Freek Wiedijk (freek@cs.kun.nl) †
Department of Computer Science, University of Nijmegen, the Netherlands

Abstract. The mathematical proof checker Mizar by Andrzej Trybulec uses a proof input language that is much more readable than the input languages of most other proof assistants. This system also differs in many other respects from most current systems. John Harrison has shown that one can have a *Mizar mode* on top of a tactical prover, allowing one to combine a mathematical proof language with other styles of proof checking. Currently the only fully developed Mizar mode in this style is the Isar proof language for the Isabelle theorem prover. In fact the Isar language has become the official input language to the Isabelle system, even though many users still use its low-level tactical part only.

In this paper we compare Mizar and Isar. A small example, Euclid's proof of the existence of infinitely many primes, is shown in both systems. We also include slightly higher-level views of formal proof sketches. Moreover a list of differences between Mizar and Isar is presented, highlighting the strengths of both systems from the perspective of end-users. Finally, we point out some key differences of the internal mechanisms of structured proof processing in either system.

Keywords: Formalized mathematics, structured proof languages, proof sketches.

1. Introduction

1.1. Overview

We compare two systems for formalizing mathematics in the computer: the Mizar system by Andrzej Trybulec from Białystok, Poland (Rudnicki, 1992; Trybulec, 1993; Muzalewski, 1993; Wiedijk, 1999), and the Isar interface by Markus Wenzel from Munich, Germany (Wenzel, 1999; Wenzel, 2002a; Wenzel, 2002b) to the Isabelle system by Larry Paulson and Tobias Nipkow (Paulson, 1994; Nipkow et al., 2002). From the perspective of recipients of formal proof documents, human readers, the proof languages of both systems appear very similar. The key differences (concerning the underlying concepts and various technical issues) become more relevant once that users wish to compose their own formalizations, or explore the internal principles of structured proof processing.

* Research supported by DFG project 'Isar' and EU working group 'TYPES'
† Research supported by EU working group 'TYPES' and EU network 'Calculemus'

The names of the systems can be used on four different levels (e.g., in the sentence 'in Isar the Hahn-Banach theorem is available' the word 'Isar' is used as referring to the Isabelle/HOL library):

the Mizar language	the Isar language
the Mizar system	the Isabelle system
	the Isabelle/HOL logic
the Mizar library	the Isabelle/HOL library

These four levels are conceptually unrelated: there could be multiple implementations of the Mizar and Isar languages, there are many logics for the Isabelle logical framework, and one could write different libraries for both systems. However, in practice all four levels are used together, as a whole. Almost all users of Mizar or Isar are using the current version of language, system, logic and library. Therefore, when discussing Mizar and Isar from a user's point of view, the Mizar and Isar languages should not be separated from the corresponding systems. For this same reason we will use the term 'Isar system' to refer to the Isabelle system with the Isar proof language, and with the Isabelle/HOL logic and library. (Although the Isar language can be used with all Isabelle logics, the large majority of Isabelle users work in Isabelle/HOL.)

In this paper we compare the current states of the Mizar and Isar systems. Both systems are under active development and probably will change significantly in the coming years. This paper should be seen as a snapshot of the current situation.

The intended audiences for this paper are those who want to know more about Mizar and Isar and the people who work on the implementation of Mizar or Isar or of a similar declarative proof assistant. We think that systems can benefit from the experiences of other systems.

The aims of this paper are:

- *To compare the Mizar and Isar proof languages.* We demonstrate the similarity between proofs in these two languages through a simple example from number theory (Sections 2 and 3), but we also show that the internal principles of processing those proofs are significantly different in both systems (Section 5).

- *To compare the Mizar and Isar systems from the experience of the end-user.* We give a high level comparison of the Mizar and Isar systems from a user's point of view, by pointing out the relative strengths and weaknesses of both systems (Section 4).

- *To attract more attention to declarative provers.* The declarative proof style of the Mizar and Isar languages can draw more people to formal proofs. The small example that we present in both languages acts as a gentle introduction to declarative proof.

1.2. Related work

The concept of human-readable formal proof was pioneered by de Bruijn, who coined the term 'mathematical vernacular' (de Bruijn, 1987) for a formal language that has a structure similar to informal mathematical text. However on the surface de Bruijn's MV language still looks more like a formal language than like natural language. Recently research into 'declarative' proof assistants (as opposed to the 'procedural' proof assistants that have been exercised for several decades now) has become popular. Declarative systems not only follow the structure but also the language of informal mathematics. Apart from the Mizar and Isar developments, in recent years research into the declarative proof style has resulted in several experimental systems.

The 'Mizar mode for HOL' (Harrison, 1996) provides an alternative interface for interactive proof composition in HOL (notably HOL Light), transferring useful ideas from the Mizar proof language into the tactical setting of HOL. Harrison introduces separate concrete syntax for structured proof commands that are translated to special tactics inside, which perform basic transformations according to natural deduction schemes of raw first order logic. Harrison also spends substantial effort on automated reasoning support, for solving 'trivial' situations implicitly (the concrete procedure may be exchanged by the user). The Mizar mode also covers a calculational reasoning style, which refers to a collection of mixed transitivity rules declared in the context (using $=/</\leq$ or similar relations). The system has been sufficiently developed to conduct some example proofs from classical analysis, covering a few pages of text. It has not been applied any further, though.

DECLARE (Syme, 1997; Syme, 1998) is a stand-alone prototype system for 'declarative' proof development, which acts like a compiler for formal documents consisting of theory specifications and structured proof outlines. The proof language is based on three main principles, namely 'first-order decomposition and enrichment', 'second-order schema application', and 'appeals to automation'. DECLARE has been advertised as 'three tactic theorem proving' (Syme, 1999). The system draws from the general experience of the HOL family (and Harrison's Mizar mode), but renounces established principles like full reduction to basic logical principles inside. DECLARE has been successfully applied by its author in some significant case-studies on Java type-safety and operational semantics (Syme, 1998). In fact, many concepts of

DECLARE have been specifically designed towards such applications of language modeling, with particular support for inductive definitions and proof schemes. DECLARE did not aim at more general applications, and has not been evaluated any further in practice (the system is not publicly available anyway).

The 'Structured Proof Language' (SPL) (Zammit, 1999a; Zammit, 1999b) aims at providing another interface for proof construction in mainstream HOL, drawing from Mizar ideas and the experience with Harrison's Mizar mode. SPL has been intended for larger scale applications, just like DECLARE, but is more careful to stay within the logical foundations of HOL. All high-level concepts of SPL are reduced to primitive HOL tactics. Zammit also spends significant effort on powerful first-order proof tools in HOL, in order to support reasoning in large steps. Another focus is on implicit simplifications (via rewriting). The SPL/HOL system has been evaluated by its author by formalizing some portions of group theory, attempting to achieve the same level of abstraction seen in the informal proofs of a given textbook.

'Mizar Light for HOL Light' (Wiedijk, 2001) represents a minimal system experiment that achieves a readable view on first-order tactical proof schemes, mainly by exhibiting propositions explicitly in the text instead of implicitly in goal configurations. It shows that it is not necessary to change the notion of proof state of a goal oriented system to be able to support declarative proofs.

Systems in the important class of 'teaching tools for formal logic' often provide readable textual representations of proofs as well, although most seem to prefer graphical views. Typically, such systems are restricted to primitive inferences in pure logic, where users may occasionally specify their own set of rules, but advanced proof procedures are unavailable.

The teaching tool ProveEasy (Burstall, 1998) provides an interactive editor for primitive natural-deduction proof texts presented in a strictly backward manner. The underlying structure is oriented towards the well-established λ-calculus view of type theory. Here the main idea is to make the types of sub-terms (local propositions) visible in the text.

Tutch (Abel et al., 2001) is a strictly text-oriented proof-checker intended for teaching constructive logic. The system deliberately excludes any kind of user interface, but acts like a batch-mode compiler of proof texts written in plain ASCII. Thus students are encouraged to focus on the task of actually writing proofs, rather than play with fancy point-and-click interfaces. Proof steps in Tutch range from primitive natural deduction to more abstract arrangements of the 'assertion level'. Nevertheless, the system refrains from arbitrary proof search, but implements an efficient algorithm for structured proof checking.

2. Example: there are infinitely many prime numbers

In a talk at the Dutch proof tools day 2002 in Utrecht, Herman Geuvers presented two slides that were meant to show the difference in style between informal and formal proofs. The informal proof on the first slide looked like this:

A romantic proof

THEOREM There are infinitely many primes:
for every number n there exists a prime $p > n$

PROOF [after Euclid]
Given n. Consider $k = n! + 1$, where $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.
Let p be a prime that divides k.
For this number p we have $p > n$: otherwise $p \leq n$;
but then p divides $n!$, so p cannot divide $k = n! + 1$,
contradicting the choice of p. QED

The formal proof on the other slide was a big and unreadable 'proof term' (taken from a Lego formalization by Mark Ruys):

A 'real' proof (proof-object)

PROOF-OBJECT [Formalization: Ruys]

$\lambda n{:}|\mathbb{N}|.[\leq= \mathit{ap2\_\mathbb{N}\_\mathbb{N}\_\Omega}\ \mathit{lessEq}][<= \mathit{ap2\_\mathbb{N}\_\mathbb{N}\_\Omega}\ \mathit{lessE}][k{=}\mathit{succ}(\mathit{fac}\ n)$
$]\ \mathit{has\_prime\_factor}\ k(\leq\mathit{\_elim\_2}(\mathit{fac}\ n)(\mathit{le\_one\_fac}\ n))(\exists y{:}|\mathbb{N}|.(n{<}y\ \&\ y$
$\leq k\ \&\ \mathit{prime}(y)))(\lambda y{:}|\mathbb{N}|.\lambda P{:}\mathit{is\_prime\_factor}\ y\ k.[D{=}\mathit{fst}(y|k)(\mathit{prime}(y)$
$)P][Q{=}\mathit{snd}(y|k)(\mathit{prime}(y))P][H{=}\mathit{fst}(1{<}y)(\Pi u{:}|\mathbb{N}|.(1{<}u)\rightarrow(u{<}y)\rightarrow(u$
$\nmid y))Q]\mathit{ExIntro}(|\mathbb{N}|)\ y\ (\lambda v{:}|\mathbb{N}|.(n{<}v\ \&\ v{\leq}k\ \&\ \mathit{prime}(v))(\mathit{pair}(n{<}y\ \&\ y$
$\leq\ k\ (\mathit{prime}(y))(\mathit{pair}(n{<}y)\ (y{\leq}k)(<\mathit{\_intro}\ y\ n\ (\lambda F{:}y{\leq}n.<\mathit{\_irrefl}\ 1\ ($
$\mathit{extenRel\_\mathbb{N}\_\mathbb{N}} < 1\ (\mathit{Eq\_refl\_\mathbb{N}}\ 1)y\ 1\ (\mathit{divides\_lemma\_3}\ y\ (\mathit{fac}\ n)\ ($
$\mathit{fac\_divides}\ y\ n(\leq\mathit{\_intro\_2}\ y\ (<\mathit{\_succ\_intro}\ 1\ y\ H))F)D)H)\bot))($
$\mathit{divides\_lemma\_1}\ y\ k\ D\ (y{\leq}k\ (\mathit{Id}\ (y{\leq}k))\ (\lambda G{:}\mathit{Eq\_\mathbb{N}}\ k\ 0.\mathit{Succ\_not\_zero}\ ($
$\mathit{fac}\ n)G\ (y{\leq}k))))\ Q))(\exists y{:}|\mathbb{N}|.(n{<}y\ \&\ \mathit{prime}(y))\ (\lambda y{:}|\mathbb{N}|.\lambda H{:}n{<}y\ \&\ y{\leq}k$
$\&\ \mathit{prime}(y).\mathit{ExIntro}(|\mathbb{N}|)\ y(\lambda w{:}|\mathbb{N}|.(n{<}w\ \&\ \mathit{prime}(w))(\mathit{pair}(n{<}y)\ ($
$\mathit{prime}(y))\ (\mathit{fst}(n{<}y)\ (y{\leq}k)(\mathit{fst}(n{<}y\ \&\ y{\leq}k)(\mathit{prime}(y))\ H))(\mathit{snd}(n{<}y$
$\&\ \ y{\leq}k)\ (\mathit{prime}(y))H))))))))$

But a formal proof does not need to look like this! In a Mizar-style proof language the formal proof of this theorem will be quite similar to the 'informal' version from Herman's first slide. Subsequently we will use this example (the infinity of the set of prime numbers) as the proof for which we will show both Mizar and Isar versions.

2.1. Mizar version of the formalization

Here is the final part of a Mizar formalization of the 'romantic proof'. The full Mizar formalization is 44 lines long. It builds on the Mizar Mathematical Library (`http://mizar.org/JFM/`), which is part of the Mizar system. Here we have used Mizar version 6.1.10.

```
reserve n,p for Nat;

theorem Euclid: ex p st p is prime & p > n
proof
 set k = n! + 1;
 n! > 0 by NEWTON:23;
 then n! >= 0 + 1 by NAT_1:38;
 then k >= 1 + 1 by REAL_1:55;
 then consider p such that
A1: p is prime & p divides k by INT_2:48;
A2: p <> 0 & p > 1 by A1,INT_2:def 5;
 take p;
 thus p is prime by A1;
 assume p <= n;
 then p divides n! by A2,NAT_LAT:16;
 then p divides 1 by A1,NAT_1:57;
 hence contradiction by A2,NAT_1:54;
end;

theorem {p: p is prime} is infinite
 from Unbounded(Euclid);
```

Note that the proof of the first theorem has to contain low level 'algebraic facts' like `n! >= 0 + 1`, `k >= 1 + 1`, `p <> 0` and `p > 1`. Apart from the part of the text that is shown, the Mizar formalization contains an '`environ`' header and a proof of the following rule scheme:

```
scheme Unbounded {P[Nat]}: {n: P[n]} is infinite
 provided for m ex n st P[n] & n > m;
```

It is proved from a similar theorem from the Mizar library.

The Mizar system is not interactive. One runs the main executable `mizf` like a batch-mode compiler and it inserts the error messages, if any, as comments inside the original Mizar source text.

2.2. Isar version of the formalization

Here is the final part of an Isar formalization of the 'romantic proof' again. The full Isar formalization is 95 lines long. It includes the theory *Main* from the standard Isabelle/HOL library (`http://isabelle.in.tum.de/library/HOL/`), as well as the theory *Factorization* by Thomas Rasmussen (from the Isabelle examples). Here we use Isabelle2002.

theorem *Euclid*: $\exists\, p \in \mathit{prime}.\ n < p$
proof −
  let $?k = n! + 1$
  obtain p **where** *prime*: $p \in \mathit{prime}$ **and** *dvd*: $p\ \mathit{dvd}\ ?k$
    using *prime-factor-exists* **by** *auto*
  have $n < p$
  proof −
    have $\neg\ p \leq n$
    proof
      assume $p \leq n$
      with *prime-g-zero* **have** $p\ \mathit{dvd}\ n!$ **by** (*rule dvd-factorial*)
      with *dvd* **have** $p\ \mathit{dvd}\ ?k - n!$ **by** (*rule dvd-diff*)
      then have $p\ \mathit{dvd}\ 1$ **by** *simp*
      with *prime* **show** *False* **using** *prime-nd-one* **by** *auto*
    qed
    then show *?thesis* **by** *simp*
  qed
  from *this* **and** *prime* **show** *?thesis* ..
qed

corollary $\neg$ *finite prime*
 using *Euclid* **by** (*fastsimp dest*!: *finite-nat-set-is-bounded simp*: *le-def*)

Note that the Isabelle library does not know the $>$ relation. Therefore, instead of writing $p > n$ one has to write $n < p$. Also the relationship between $<$ and $\leq$ is not hard-wired into the system, making an extra level of **proof** ... **qed** in the proof of $n < p$ necessary here. The *fastsimp* method that is used in the proof of the final corollary is not one of the three most commonly used ones: *simp*, *blast* and *auto*. The corollary may also have been proved with a three step Isar proof that only uses the *auto* method.

Apart from the part of the text that is shown here, the Isar formalization contains the definition of the factorial function and the following five lemmas (including hints for automated tools):

lemma [*iff*]: $0 < n!$

lemma *dvd-prod* [*iff*]: $n\ \mathit{dvd}\ \mathit{prod}\ (n\ \#\ \mathit{ns})$

lemma *prime-factor-exists*: $1 < n \Longrightarrow \exists\, p \in prime.\ p\ dvd\ n$

lemma *dvd-factorial*: $0 < m \Longrightarrow m \leq n \Longrightarrow m\ dvd\ n!$

lemma *finite-nat-set-is-bounded*:
finite $M \Longrightarrow \exists\, n{::}nat.\ \forall\, m \in M.\ m \leq n$

Isar is an interactive system. The preferred interface to Isar is Proof General (Aspinall, 2000), which is the 'proof assistant mode' for the (X)Emacs editor. When editing an Isar formalization, at each point in the text the interface shows relevant bits of the proof state.

2.3. Relating the two languages

The Mizar and Isar languages use different keywords for similar notions. Here is a rough translation table that relates major language elements.

Mizar	*Isar*
`let`	**fix**
`assume`	**assume**
`set`	**let**
`set`	**def**
`consider` ... `such that`	**obtain** ... **where**
`take`	*no equivalent*
`per cases` ...	**proof** ... **qed**
`suppose`	**next assume**
no keyword	**have**
`thus`	**show**
`hence`	**then show**
`thesis`	*?thesis*
`proof` ... `end`	**proof** ... **qed**
`@proof` ... `end`	**sorry**
`now` ... `end`	**{** ... **}**
`then`	**then**
`then` ... `by`	**with**
`by`	**from**
`by`	**using**
`;`	**.**
`;`	**..**
`;`	**by** *auto*
`from` ... `;`	**by** (*rule* ...)

Note that the mapping between language elements is not exact. For instance the distinction between `by` and `from` (the first is first order while the second is higher order) is not reflected in the Isar equivalent.

Also note that in the translation of Mizar's '`by` ... `;`' to Isar's '`using` ... `by` *auto*', we have actually separated '`by`' and '`;`'. However in Mizar these two parts are not considered to have separate meanings (in fact '`;`' is just a syntactic terminator).

Despite the similarities between the two languages, we observe some stylistic differences, stemming from different philosophies of language design. The Isar proof language is *compositional*, it emerges in *bottom-up* fashion from only a few basic primitives following pure principles of natural deduction — according to the existing Isabelle framework (Paulson, 1989). In contrast, Mizar does not particularly try to reduce everything to a few primitive language elements.

Without going into details, just consider the following examples:

1. Mizar's `let ... such that` is almost equivalent to a combination of `let` and `assume`, but not exactly. In the first case one is not allowed to follow the step by a `then`, while in the second case one is. According to Isar's philosophy, the first really should be an abbreviation of the second, providing a more uniform language view (Wenzel, 2002a, §5.3).

2. Mizar has built-in automated justification of `by`, while Isar uses explicit references to particular proof methods such as the automated *simp*, *blast*, *auto*, or single-step *rule* applications. Therefore the combination '**by** *auto*' becomes a canonical automated justification, while '**by** (*rule* ...)' corresponds to rule scheme applications, which are treated separately in Mizar.

Also note that the above table merely covers the 'logical' parts of the two languages. Both Mizar and Isar also support a notion of 'algebraic reasoning', using chains of equalities (and inequalities in Isar). See also (Bauer and Wenzel, 2001) for further discussion of this particular aspect of structured proofs.

3. A higher-level view: formal proof sketches

Formal proof sketches address the issue of presenting formal texts in a more abstract manner, and support top-down development.

In this section we present proof skeletons in the formal languages of Mizar and Isar that are as close as possible to the informal natural language proof on page 5. Also, in this section we show the full files including the headers, instead of only showing the final theorem.

3.1. Mizar version of a formal proof sketch

The Mizar system is able to continue to check a text after it finds the first error. A Mizar text that contains only justification errors, which can be eliminated by adding more text, is in (Wiedijk, 2002) called a *formal proof sketch*. Here is a formal proof sketch that corresponds to the Mizar formalization in Section 2.1. It is the full text of the formal proof sketch, including the 'environ' header.

```
        environ
         vocabulary FINSET_1, FILTER_0, QC_LANG1, ARYTM_3;
         notation INT_2, FINSET_1, ARYTM, NAT_1, NAT_LAT;
         constructors NAT_1, NAT_LAT;
         clusters ARYTM, NAT_1;
         requirements SUBSET, ARYTM;

        begin
         reserve n,p for Nat;

        theorem ex p st p is prime & p > n
        proof
         set k = n! + 1;
         consider p such that
*4 →      p is prime & p divides k;
         take p;
*4 →     thus p is prime;
         assume p <= n;
*4 →     p divides n!;
*4 →     p divides 1;
*1 →     thus contradiction;
        end;

*4 →    theorem {p: p is prime} is infinite;
```

Mizar gives six justification errors for this text. These have been indicated by arrows in the left margin. The numbers in front of the arrows are the error codes. The explanation of error `*1` is '`It is not true`' and the explanation of error `*4` is '`This inference is not accepted`'. Those two errors are the two justification errors of the Mizar system.

3.2. Isar version of a formal proof sketch

The Isar system stops checking after the first error that it finds. However, one can skip justification errors by inserting the 'fake proof' **sorry**. Therefore, the Isar system also can be used to write and check formal proof sketches. Here is a formal proof sketch that corresponds to the Isar formalization in Section 2.2. The **sorry** elements are pretty-printed as '⟨*proof*⟩'. This is the full text of the formal proof sketch, including the '**theory** ... **end**' brackets.

theory *Sketch* = *Main* + *Factorization*:

consts *fact* :: *nat* ⇒ *nat* ((-!) [1000] 999)
primrec
  $0! = 1$
  $(Suc\ n)! = n! * Suc\ n$

theorem $\exists\, p \in prime.\ n < p$
proof –
  let $?k = n! + 1$
  obtain p **where** $p \in prime$ **and** $p\ dvd\ ?k$
    ⟨*proof*⟩
  have $n < p$
  proof –
    have $\neg\ p \leq n$
    proof
      assume $p \leq n$
      then have $p\ dvd\ n!$ ⟨*proof*⟩
      then have $p\ dvd\ 1$ ⟨*proof*⟩
      then show *False* ⟨*proof*⟩
    qed
    then show *?thesis* ⟨*proof*⟩
  qed
  then show *?thesis* ⟨*proof*⟩
qed

corollary ¬ *finite prime* ⟨*proof*⟩

end

4. The user experience: eighteen differences

We will now list eighteen differences between the Mizar and Isar systems. For each of these differences we will state which system (in our opinion) is the more attractive one. In nine cases it will turn out to be the Mizar system, and in nine cases it will turn out to be the Isar system. But of course not all differences are of the same importance.

4.1. Better documentation: Isar

The Isar system has much better documentation than the Mizar system.

For Mizar there are introductory texts, like (Muzalewski, 1993), but they are too brief to be of real use, and they generally describe old versions of the system.

The Isabelle/Isar documentation is clear and thorough. There is a tutorial for Isabelle/HOL (Nipkow et al., 2002) and there are reference manuals for Isabelle (Paulson, 2002a), Isar (Wenzel, 2002b) and for further logics of the system, notably FOL and ZF (Paulson, 2002b). All of these are distributed with Isabelle (`http://isabelle.in.tum.de/dist/`). The tutorial for Isabelle/HOL is also available as a printed book from Springer. Finally there is Wenzel's PhD thesis (Wenzel, 2002a) that describes the development of Isar in detail, including many practical proof patterns, as well as some reference applications. A dedicated Isabelle/Isar tutorial for beginners is still missing.

4.2. Available on more platforms: Isar

The Isar system is available on more different kinds of platforms than the Mizar system.

The Mizar system is written in Borland Delphi Pascal and has been ported to Free Pascal. Those compilers cannot generate code for many machines. Therefore, the Mizar system is only available on Intel systems (both Windows and Linux). Some old DOS legacy is still visible, e.g. the limitation of Mizar article names to 8 characters. The binaries of the Mizar system can be freely downloaded on the Internet. The source of the Mizar system is only available to members of the Association of Mizar Users.

The Isar system is written in standard ML. The Isabelle2002 system can be run on all major Unix platforms, including Intel, Sun and Apple. The binaries and source of the Isabelle system can be freely downloaded on the Internet.

4.3. More users: Mizar

The Mizar language has more serious users than the Isar language.

It is easy to determine who are the serious Mizar users. Those are exactly the people who have contributed an 'article' to the Mizar Mathematical Library. In the Mizar community these are called 'Mizar authors'. A Mizar article is a significant piece of Mizar text. To write a Mizar article one needs to understand the Mizar language thoroughly: even authors who are not active anymore can still be considered to be Mizar experts. Currently there are more than 120 Mizar authors.

The Isar proof language is now the official input language of the Isabelle system. However, most Isabelle users only use the 'improper commands' **apply** and **done** of the Isar language, effectively writing an 'old style' tactic proof encapsulated in a thin Isar wrapping. Presently only few people write 'new style' Isar proofs, although some substantial applications are distributed with Isabelle2002, such as the Hahn-Banach Theorem for real vector spaces (Bauer and Wenzel, 2000).

4.4. Bigger mathematical library: Mizar

The Mizar system has a bigger library than the Isabelle/Isar system.

It is difficult to measure the size of a formal library in an objective way. However, even from the small example from Section 2 it is clear that the Mizar system has more background theory available than the Isabelle system, as far as classical mathematics is concerned.

For the Mizar proof above, only one lemma needed to be proved, while for the Isar proof five lemmas were needed. Also the Mizar library already has the definition of the factorial function, while the Isabelle library does not have it as a standard function. Finally, for the example the Mizar library was sufficient, while the Isar library needed to be supplemented by a theory about factorization.

The size of the Mizar library is about 50 megabytes, while that of Isabelle is about 10 megabytes (including the sources of the system and the example theories).

4.5. Better library lookup: Isar

The Isar system has better support for finding theorems in the library than the Mizar system.

The main difficulty when writing mathematical formalizations is finding the relevant theorems in the library. For the Mizar system most people use the `grep` program for this. There are more specific tools, like Grzegorz Bancerek's MML Query (`http://megrez.mizar.org/mmlquery/`), but they are not easy to understand and they are not integrated into the official Mizar system.

The Isabelle/Isar system, being interactive, provides a 'live' view of the current theory context. Its **thms-containing** command finds theorems in the library: for example, **thms-containing** $x < y \;\; x \leq y$ retrieves all facts involving the $<$ and $\leq$ relations. This is a much more structured way to look for theorems than the `grep` program.

4.6. Shorter formalizations: Mizar

The Mizar language gives slightly shorter formalizations than the Isar language. (This is the least objective of the choices in this list.)

Basically the Mizar and Isar languages are very similar and will lead to proofs that have similar length. However, when looking at the examples from Sections 2 and 3, the Mizar proofs seem a bit more terse. For instance the Isar justification '**using** *prime-factor-exists* **by** *auto*' is longer than the Mizar justification '`by INT_2:48;`'. Also the Isar formalization from Section 2.2 has more than twice as many lines as the Mizar formalization.

There are two effects causing a difference in size between Mizar and Isar formalizations. These two effects counteract each other. In the Mizar library much more has been proved already, leading to shorter Mizar formalizations. But the standard Isabelle/Isar proof methods are more powerful than Mizar justifications (see 4.9 below) leading to shorter Isar formalizations.

4.7. Simpler proof state: Mizar

The Mizar system has a much simpler proof state than the Isar system.

The Isar system divides the proof state in a 'static' and a 'dynamic' context, and it also contains a set of 'using this' facts to move information from the static to the dynamic world. However, in the way that the Isar proof state is used, the static and dynamic contexts are duplicates of each other. To prove a subgoal one has to build a copy of the corresponding part of the dynamic context in the static context. Then when giving the **show** command the system verifies whether both contexts are compatible. If this turns out not to be the case, then the user gets an error message and has to search for himself which of the preceding **assume** lines caused the incompatibility. The duplication of contexts does not give fundamental advantages concerning basic goal refinement steps, but provides a slightly more 'declarative' view.

In Mizar there is only a single context, which is closer to the traditional goal-oriented concept of stepwise problem refinement.

4.8. Simpler module system: Isar

The Isabelle/Isar system has a much simpler module system than the Mizar system.

In the Mizar system one has to give an '`environ`' header that specifies the articles from the library that are going to be used. For each of the various notions that an article exports (notation, definitions, theorems, etc.), the `environ` contains a separate import list. Generally those lists contain more or less the same articles. Also, the Mizar `environ` refers to *two* kinds of modules, the articles and the vocabularies. Both use different name spaces. In practice it turns out to be difficult to get a Mizar `environ` right.

In the Isar system one just gives a single list of theories to be imported. Also the import of theories is transitive. This gives a much simpler module system.

4.9. More powerful justifications: Isar

The Isar system has more powerful justifications than the Mizar system.

In Mizar there are only two justification methods: the `by` justification and the `from` justification. Both methods have been hard-wired into the system, and have intentionally been kept simple. Therefore they can be checked in a fast way and have a relatively clear semantics.

In Isar there are many justification methods, and a user can add justification methods of his own to the system. The most common ones are *rule*, *simp*, *blast*, *auto* and *arith*. These proof tools are generally more powerful than the Mizar justification methods. This explains why the 'algebraic facts' (`n! >= 1`, `k >= 2`, `p <> 0`, `p > 1`) from the Mizar proof do not need to be stated explicitly in the Isar proof.

4.10. Simpler justifications: Mizar

Mizar justifications are easier to work with than the Isar justifications.

The Mizar justifications only use `by` (light-weight automated reasoning) or `from` (single rule scheme application). That means that if a certain justification does not work, then there is nothing to be done on the justification level. This actually makes writing proofs easier, since one focuses on the steps in the proof and not on internal details.

In Isar there are two ways to write proofs: the old way of 'unstructured scripts' and the new way of 'structured texts'. The old way consists of applying tactics to the goal, using sequences of the **apply** command followed by the **done** command. The new way consists of writing Mizar-style proofs. But in fact when writing proofs in the new way, one still generally discovers the justifications in the old way, before turning the (local) tactic proof that one discovered into a single **by** justification. In development one still uses the **apply** command, although this does not show in the final proof text. This means a lot of experimentation with various tactics and their parameters, focusing on the machinery of the justifications and not on the mathematics of the proof. Because of the diversity of Isar justifications, users are tempted to spend considerable effort into tweaking the final text!

4.11. Easier to work with incomplete text: Mizar

Mizar can deal better with incomplete text than the Isar system.

The Mizar system will keep checking a text no matter what errors it encounters (as long as the '`environ`' header is correct, but that is a different matter). It will recover from syntax errors, type errors, missing definitions, justification errors. This means that if one is in the middle of a complicated proof (even one that is not syntactically correct yet), one can switch to a different part of the file *after* that complicated proof, and continue to do useful work there.

For instance, consider the following fragment of a Mizar text, in which the proof of the scheme has not yet been finished:

```
scheme Unbounded {P[Nat]}: {n: P[n]} is infinite
 provided
A1: for m ex n st P[n] & n > m
proof
::> *214
 now let m;
::>*214
  consider n' such that
A2: P[n'] & n' > m by A1;
  take n';
::>      *215,215

theorem Euclid: ex p st p is prime & p > n
::>   *70
proof
 set k = n! + 1;
 n! > 0 by NEWTON:23;
```

etcetera

```
::> 70: Something remains to be proved
::> 214: "end" missing
::> 215: No pairing "end" for this word
```

All errors in this example are related to the fact that the scheme `Unbounded` is not finished. The remaining text is checked as expected.

The Isar system cannot do anything like this. One can skip missing justifications with the **sorry** command. For all other errors, one needs to change the text (for instance by putting it in comment brackets) to make it correct, before being able to work on text after the error. This is less convenient than the way Mizar treats errors. Here we experience a disadvantage of Isar's incremental approach vs. Mizar's batch-mode.

4.12. More feedback from the system: Isar

The Isar system gives more feedback to the user than the Mizar system.

The Mizar system will only give natural numbers to the user: error codes and the positions in the file to which these codes apply. For instance, the errors in the formal proof sketch from Section 3 correspond to the 18 natural numbers:

16	26	4
18	16	4
20	13	4
21	12	4
22	19	1
25	35	4

(The meaning of these is: line numbers, offsets into the line, and error codes.) In practice it is often not very convenient just to be told codes, because with more specific information the errors would be easier to understand. In particular it would be good to know what the `thesis` is at specific points in the text (in other systems called the 'goal').

The Isar system is a full-fledged interactive tactic-based prover. It is able to print a large amount of information at any point in the text, including the goal that has to be proved. For instance, just before the '**with** *prime* **show** *False*' line in the example of Section 2.2, the proof state of the Isar system looks like this:

```
proof (state): step 19

fixed variables: n, p = p
prems:
  p ∈ prime
  p dvd n! + 1
  p ≤ n

this:
  p dvd 1

goal (have, 1 subgoal):
¬ p ≤ n
 1. p ≤ n ⟹ False
```

4.13. Logically more general: Isar

The Isar language can represent proofs of many different logics, unlike the Mizar language which only can represent proofs of classical first order predicate logic.

The proof steps of the Mizar language are the natural deduction steps of first order predicate logic, including built-in classical principles.

Isabelle is a *logical framework* where the logic can be chosen by the user. The Isar language works on the generic level of the meta-logic, and is compatible with common object-logics (e.g. HOL, HOLCF, FOL, ZF). For instance, the **obtain** element (which corresponds to Mizar's `consider`) directly refers to Isabelle's meta-logic, bypassing any particular notion of existential quantifier in the object-logic.

4.14. More mathematical type system: Mizar

The Mizar type system is much richer and much more mathematical than the type system that Isar inherits from Isabelle.

Mizar has structure types, dependent types, various kinds of subtyping and type modifiers called *attributes*. Mizar uses the types for automatic deduction and for overloading of predicates and operators.

Isar uses the HOL type system. In particular it does not have dependent types. Dependent types are very important to be able to model mathematical practice in a natural way, but prevent conveniences like Hindley-Milner type inference.

4.15. Simpler type system: Isar

Isar types are easier than Mizar types.

The Mizar type system is difficult to use. Regularly one needs 'clusters' or 'redefinitions' in the environment to get the typing of expressions right. Finding these clusters and redefinitions in the Mizar library takes a significant amount of time.

The Isabelle types are much simpler than the Mizar types, and rarely cause problems for the user. Occasionally Isabelle's type inference can lead to confusion when the inferred types turn out to be more general than has been anticipated by the user.

4.16. Better support for mathematical symbols: Isar

Isar offers the use of mathematical symbols while Mizar does not.

Until recently Mizar articles contained symbols from the high ASCII part of the DOS character set. These symbols have been eliminated from the Mizar library and currently Mizar articles only use standard ASCII characters.

Isar formalizations contain many mathematical symbols. Using the Emacs + Proof General combination these can be displayed on screen by the X-Symbol package (`http://x-symbol.sourceforge.net/`). In the final pretty-printed document, these symbols are printed using the

LaTeX typesetting system. In the Isar files symbols are represented using special escape sequences. For instance the statement of the theorem from Section 2 is represented in the ASCII file as:

```
theorem Euclid: "\<exists>p \<in> prime. n < p"
```

4.17. Less meaningless typographic noise: Mizar

Mizar texts contain less mathematically meaningless symbols than Isar texts.

In Isar formalizations there are some low-level 'noise' characters that do not mean anything mathematically. There are quotes (") around formulas (they are not shown in the pretty-printed version of Isar but are seen in the input and need to be typed), there are question marks (*?*) for term bindings, there are minuses ($-$) and dots (**.** and **..**).

4.18. More natural language-like: Mizar

Mizar texts resemble natural language more than Isar texts.

In Mizar both the proof steps and the formulas use English keywords. In Isar the formulas are written with symbols. This causes Mizar texts to be closer to mathematical English. For instance, in Mizar one might write:

`assume that for n being natural number such that` ...

while in Isar this would look more symbolic:

assume $\forall\, n{::}nat.\ \ldots$

The ratio of verbal expressions versus formulas is a matter of taste. Traditional mathematics used to be mostly verbal for several hundred years, even for expressions like '$a = b + c$'.

5. Internal mechanisms: key differences of proof processing

We shall now take a closer look at the internal machinery of both systems, in order to gain some understanding of the basic principles underlying structured proof processing encountered here. Mizar and Isar differ considerably inside, despite the similar high-level view of final proof texts shown to the recipients.

Roughly speaking, Mizar is based on *procedural transformations* of a pending proof problem, with primitive elements stemming from classical first-order logic. In contrast, the most fundamental Isar operations are *applicative refinements* of pending problems according to the built-in principle of higher-order resolution (and unification) of the Isabelle/Pure framework.

Subsequently, the fact $(\exists x.\, \forall y.\, P\, x\, y) \longrightarrow (\forall v.\, \exists u.\, P\, u\, v)$ shall serve as an example. This is sufficiently characteristic for quantifier reasoning, since both introductions and eliminations of $\forall$ and $\exists$ are involved.

5.1. Mizar

Technically, a Mizar proof consists of a claim at the head (theorem or local statement etc.), together with a body consisting of several transformations of the remaining problem. Within the body, the special proposition `thesis` is updated dynamically to reflect the pending goal; at the same time a context of local variables and facts is being built up. In the example below we indicate the course of value of `thesis`.

```
theorem (ex x st for y holds P[x,y]) implies
  (for v holds ex u st P[u,v])
proof
 assume ex x st for y holds P[x,y];
  :: thesis = for v holds ex u st P[u,v]
 then consider x such that A: for y holds P[x,y];
  :: thesis unchanged
 let v;
  :: thesis = ex u st P[u,v]
 take x;
  :: thesis = P[x,v]
 thus P[x,v] by A;
  :: thesis = empty conjunction
end;
```

(The combination `assume ex x st` ... `then consider x such that` ... can be abbreviated as `given x such that` ... but for clarity we have not done this.) Mizar proof body elements may be understood as standard transformations according to well-known principles of natural deduction. This simplified model of Mizar's approach to structured proof processing has been elaborated further in (Wiedijk, 2000). Nevertheless, Mizar's built-in procedure actually admits slightly more involved transformations, based on a theory of so-called 'semantic correlates', which provides a certain algebraic view on top of classical first-order logic.

For example, α-conversion of bound variables works as expected (above we have sticked to the original names x, y, u, v nevertheless). Moreover, conjunctions may be considered as split, both in assumptions and conclusions (so `assume A & B` is the same as `assume A` followed by `assume B`, which is the same as `assume A and B`). Further equivalences allow to swap a final `thus not A` with the combination `assume A` ... `thus contradiction`.

Unfortunately, Mizar's semantic correlates do not cover commutativity, so the order of basic transformations in the above proof is essentially dictated by that of the initial statement. The arrangement in the body may not just swap assumptions or skip unused ones. In practice, one would often wish to change the order that local facts emerge, arriving at a mostly linear chain that reflects the course of reasoning in the body, rather than the original goal statement. Failing to do so, one typically requires additional labels and references, which are apt to spoil the structure of Mizar texts.

5.2. Isar

In Isar a proven fact consists of a claim at the head (similar to Mizar), followed by a *proof* according to Isar's formal syntax. In general this consists of an initial **proof** step (with optional *method* specification), followed by a sequence of body *statements*, followed by a terminal **qed** step (again with optional *method*). Isar's **by** (with one or two *method* arguments) merely abbreviates a proof with an empty body.

The initial and terminal methods are the only places where arbitrary operational transformations of the pending problem may take place, say an initial induction followed by a terminal method to solve any remaining sub-problems automatically. Within the body text, the only way to affect the pending goal configuration is via explicit **show** statements. This format is again illustrated by our example of quantifier reasoning.

lemma $(\exists\, x.\ \forall\, y.\ P\ x\ y) \Longrightarrow (\forall\, v.\ \exists\, u.\ P\ u\ v)$
proof
  — rule $(\bigwedge v.\ \exists\, u.\ P\ u\ v) \Longrightarrow (\forall\, v.\ \exists\, u.\ P\ u\ v)$
 fix v
 assume $\exists\, x.\ \forall\, y.\ P\ x\ y$
 then obtain u **where** $\forall\, y.\ P\ u\ y$ **..**
  — rule $\bigwedge C.\ (\exists\, x.\ \forall\, y.\ P\ x\ y) \Longrightarrow (\bigwedge u.\ (\forall\, y.\ P\ u\ y) \Longrightarrow C) \Longrightarrow C$
 then have $P\ u\ v$ **..**
  — rule $(\forall\, y.\ P\ u\ y) \Longrightarrow P\ u\ v$
 then show $\exists\, u.\ P\ u\ v$ **..**
  — rule $P\ u\ v \Longrightarrow (\exists\, u.\ P\ u\ v)$
  — composition $\bigwedge v.\ (\exists\, x.\ \forall\, y.\ P\ x\ y) \Longrightarrow (\exists\, u.\ P\ u\ v)$
qed

The notation used for rule statements is that of the Isabelle/Pure framework (Paulson, 1989): $\bigwedge$ and $\Longrightarrow$ refer to meta-level universal quantification and implication, respectively. Both goal states and inference rules are represented as 'Hereditary Harrop Formulae' built from these connectives; the general form may be specified inductively as $H = \bigwedge x_1 \ldots x_m. H_1 \Longrightarrow \ldots H_n \Longrightarrow A$, for atomic propositions A. This is a natural generalization of Horn clauses $A_1 \Longrightarrow \ldots A_n \Longrightarrow A$. The internal Isabelle machinery uses a generalization of Prolog-style backwards resolution as the most basic reasoning principle.

The implicit rule applications indicated in the above example are easily recognized as instances of $\forall$ introduction, $\exists$ elimination, $\forall$ elimination, and $\exists$ introduction, respectively. The final composition step fits the result of the **show** into the enclosing goal; the local context of **fix**-**assume** has already been discharged in the obvious manner.

The key idea underlying the Isar interpretation process is to provide a structured discipline to drive standard inferences of the underlying Isabelle/Pure framework (arbitrary automated proof tools may enter the scene much later). As it happens, the Isabelle kernel is able to record the resulting course of internal inferences as *proof-objects* in canonical λ-term representation (Berghofer and Nipkow, 2000). For our proof this looks as follows:

$$
\begin{array}{l}
\boldsymbol{\lambda} H \colon \exists\, x.\ \forall\, xa.\ P\ x\ xa. \\
\quad \mathit{allI}\ \cdot\ \mathit{TYPE}('b)\ \cdot\ (\lambda v.\ \exists\, x.\ P\ x\ v)\ \bullet \\
\quad (\boldsymbol{\lambda} v.\ \mathit{exE}\ \cdot\ \mathit{TYPE}('a)\ \cdot\ (\lambda x.\ \forall\, xa.\ P\ x\ xa)\ \cdot\ \exists\, x.\ P\ x\ v\ \bullet\ H\ \bullet \\
\qquad\quad (\boldsymbol{\lambda} u\ H \colon \forall\, x.\ P\ u\ x. \\
\qquad\qquad \mathit{exI}\ \cdot\ \mathit{TYPE}('a)\ \cdot\ (\lambda u.\ P\ u\ v)\ \cdot\ u\ \bullet \\
\qquad\qquad\ (\mathit{allE}\ \cdot\ \mathit{TYPE}('b)\ \cdot\ P\ u\ \cdot\ v\ \cdot\ P\ u\ v\ \bullet\ H\ \bullet\ (\boldsymbol{\lambda} H \colon P\ u\ v.\ H))))
\end{array}
$$

Thus Isabelle/Isar is able to cover both human-readable and foundationally clean representations of formal proofs. Observe how the latter has lost much 'redundant' structure of the original text (due to internal β-normalization.

Similar to Mizar, Isar proof processing involves a number of liberalities in the arrangement of proof body elements, but is biased towards the generic concepts of the Isabelle/Pure framework rather than classical principles. These proof text equivalences directly correspond to basic facts of the meta-logic, including α-conversion of parameters $(\bigwedge x. P\, x) \equiv (\bigwedge y. P\, y)$, permutation of parameters $(\bigwedge x\, y. P\, x\, y) \equiv (\bigwedge y\, x. P\, x\, y)$ and assumptions $(A \Longrightarrow B \Longrightarrow C) \equiv (B \Longrightarrow A \Longrightarrow C)$, commutation of parameters and assumptions $(A \Longrightarrow (\bigwedge x. B\, x)) \equiv (\bigwedge x. A \Longrightarrow B\, x)$. Moreover, the composition phase of fitting the result of **fix**-**assume**-**show** back into a goal admits projection of the context, i.e. the body may have omitted unused assumptions. Sub-goals may usually be solved in any order, too.

In practice, Isar proof texts require much less labeling of intermediate facts than Mizar.

Generally speaking, the proof composition scheme of Isar is somewhat more 'generic' and 'declarative' than Mizar. On the other hand, Isar is reluctant to let the user operate directly on pending problems, resulting in slower progress of consecutive transformations. Here Isar typically requires additional local statements (with separate sub-proofs). This has only been avoided in the above example, since we have used $\Longrightarrow$ of the meta-logic at the outermost level (conforming to common Isabelle practice); using $\longrightarrow$ of the object-logic would have demanded an additional **assume-show** layer in the beginning, to accommodate implication introduction.

6. Conclusion

We have compared the proof languages of the Mizar and Isar systems, listing various properties of those systems, and showing their relative strong points. By presenting a small but realistic mathematical example in both systems side by side, we hope to have escaped the common delusions of artificial 'benchmark problems'. Our formalizations of Euclid's proof of the existence of infinitely many primes is intended to represent quite typical applications of either proof system.

The system evaluation shows that both Mizar and Isar provide a solid environment for formalized mathematics. Despite the rather different background and design philosophies, the end user experience is quite similar in general. Recall that only the differences have been pointed out explicitly.

Mizar provides proven technology for formalizing classical mathematics, with a huge body of 'articles' being collected over the last 10 to 20 years. Isar takes the existing Isabelle framework as a starting point to achieve a generic environment for human-readable proof documents, where classical mathematics is just one potential application domain.

Certainly, both systems have their inherent advantages and disadvantages. Speaking in terms of the system itself, the best choice for users is probably a matter of taste. In reality, the availability of existing background theory and proof tools is probably more important.

Concerning future work, it would be interesting to study how much of the advantages of Mizar can be added to Isar, and vice versa. The present paper may serve as a guideline for any such efforts towards better acceptance of formalized mathematics in broader circles, by providing convincing computer assistance.

References

Abel, A., B.-Y. E. Chang, and F. Pfenning: 2001, 'Human-Readable Machine-Verifiable Proofs for Teaching Constructive Logic'. IJCAR Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs (PTP-01), `http://www.tcs.informatik.uni-muenchen.de/~abel/ptp01.ps.gz`.

Aspinall, D.: 2000, 'Proof General: A Generic Tool for Proof Development'. In: *European Joint Conferences on Theory and Practice of Software (ETAPS)*.

Bauer, G. and M. Wenzel: 2000, 'Computer-Assisted Mathematics at Work — The Hahn-Banach Theorem in Isabelle/Isar'. In: T. Coquand, P. Dybjer, B. Nordström, and J. Smith (eds.): *Types for Proofs and Programs: TYPES'99*, Vol. 1956 of *LNCS*.

Bauer, G. and M. Wenzel: 2001, 'Calculational reasoning revisited — an Isabelle/Isar experience'. In: R. J. Boulton and P. B. Jackson (eds.): *Theorem Proving in Higher Order Logics: TPHOLs 2001*, Vol. 2152 of *LNCS*.

Berghofer, S. and T. Nipkow: 2000, 'Proof terms for simply typed higher order logic'. In: J. Harrison and M. Aagaard (eds.): *Theorem Proving in Higher Order Logics: TPHOLs 2000*, Vol. 1869 of *LNCS*.

Burstall, R.: 1998, 'Teaching people to write Proofs: a Tool'. In: *CafeOBJ Symposium, Numazu, Japan*.

de Bruijn, N.: 1987, 'The Mathematical Vernacular, a language for mathematics with typed sets'. In: P. Dybjer et al. (eds.): *Proceedings of the Workshop on Programming Languages*. Marstrand, Sweden.

Harrison, J.: 1996, 'A Mizar Mode for HOL'. In: *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics, TPHOLs '96*, Vol. 1125 of *LNCS*. Springer.

Muzalewski, M.: 1993, *An Outline of PC Mizar*. Brussels: Fondation Philippe le Hodey. `http://www.cs.kun.nl/~freek/mizar/mizarmanual.ps.gz`.

Nipkow, T., L. Paulson, and M. Wenzel: 2002, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, Vol. 2283 of *LNCS*. Springer.

Paulson, L.: 1994, *Isabelle: a generic theorem prover*, Vol. 828 of *LNCS*. Springer.

Paulson, L.: 2002a, 'The Isabelle Reference Manual'. `http://isabelle.in.tum.de/doc/ref.pdf`.

Paulson, L. C.: 1989, 'The foundation of a generic Theorem Prover'. *Journal of Automated Reasoning* **5**(3).

Paulson, L. C.: 2002b, 'Isabelle's Logics: FOL and ZF'. `http://isabelle.in.tum.de/doc/logics-ZF.pdf`.

Rudnicki, P.: 1992, 'An Overview of the MIZAR Project'. In: *1992 Workshop on Types for Proofs and Programs*. Bastad.

Syme, D.: 1997, 'DECLARE: A Prototype Declarative Proof System for Higher Order Logic'. Technical Report 416, University of Cambridge Computer Laboratory.

Syme, D.: 1998, 'Declarative Theorem Proving for Operational Semantics'. Ph.D. thesis, University of Cambridge.

Syme, D.: 1999, 'Three Tactic Theorem Proving'. In: *Theorem Proving in Higher Order Logics, TPHOLs '99, Nice, France*, Vol. 1690 of *LNCS*. Springer.

Trybulec, A.: 1993, 'Some Features of the Mizar Language'. Presented at a workshop in Turin, Italy.

Wenzel, M.: 1999, 'Isar — a Generic Interpretative Approach to Readable Formal Proof Documents'. In: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L.

Thery (eds.): *Theorem Proving in Higher Order Logics: TPHOLs '99*, Vol. 1690 of *LNCS*.

Wenzel, M.: 2002a, 'Isabelle/Isar — a versatile environment for human-readable formal proof documents'. Ph.D. thesis, Institut für Informatik, Technische Universität München. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html.

Wenzel, M.: 2002b, *The Isabelle/Isar Reference Manual*. TU München. http://isabelle.in.tum.de/doc/isar-ref.pdf.

Wiedijk, F.: 1999, 'Mizar: An Impression'. http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz.

Wiedijk, F.: 2000, 'The Mathematical Vernacular'. http://www.cs.kun.nl/~freek/notes/mv.ps.gz.

Wiedijk, F.: 2001, 'Mizar Light for HOL Light'. In: R. J. Boulton and P. B. Jackson (eds.): *Theorem Proving in Higher Order Logics: TPHOLs 2001*, Vol. 2152 of *LNCS*.

Wiedijk, F.: 2002, 'Formal proof sketches'. http://www.cs.kun.nl/~freek/notes/sketches.ps.gz.

Zammit, V.: 1999a, 'On the Implementation of an Extensible Declarative Proof Language'. In: *Theorem Proving in Higher Order Logics, TPHOLs '99, Nice, France*, Vol. 1690 of *LNCS*. Springer.

Zammit, V.: 1999b, 'On the Readability of Machine Checkable Formal Proofs'. Ph.D. thesis, University of Kent.