Formal Proof Sketches

Freek Wiedijk

Department of Computer Science, University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract. Formalized mathematics currently does not look much like informal mathematics. Also, formalizing mathematics currently seems far too much work to be worth the time of the working mathematician. To address both of these problems we introduce the notion of a *formal proof sketch*. This is a proof representation that is in between a fully checkable formal proof and a statement without any proof at all. Although a formal proof sketch is too high level to be checkable by computer, it has a precise notion of correctness (hence the adjective *formal*).
We will show through examples that formal proof sketches can closely mimic already existing mathematical proofs. Therefore, although a formal proof sketch contains gaps in the reasoning from a formal point of view (which is why we call it a *sketch*), a mathematician probably would call such a text just a 'proof'.

1 Introduction

1.1 Problem

This paper is about formalization of mathematics: the encoding of mathematics in a formal language in sufficient detail that a computer program can verify the correctness. The systems that are currently most suitable for this are Coq, NuPRL/MetaPRL, Mizar, Isabelle/Isar, HOL and PVS. Of these systems Mizar and Isabelle/Isar are *declarative*, which means that the input language of the system is designed to be similar to the language of the informal proofs that one finds in mathematical papers and textbooks.

Two main applications of formalized mathematics are:

1. *representation*, and from this *presentation*, of the mathematics
2. verification of the mathematical *correctness* of the mathematics

However, if one looks at the current state of the art in formal mathematics, then both seem to have difficulties:

– A mathematical formalization currently almost always is a big tar file on an ftp server. Inside such a tar file one finds a number of 'proof script' files, which mostly resemble program source code. Even with the declarative systems Mizar and Isabelle/Isar these files are not readable as ordinary mathematical text. At best one can read them like one might study a computer program.

This means that the files of a formalization by themselves are useless to communicate the mathematics that is in them.

For this reason several people have developed tools that produce natural language versions of formalized proofs. However, we have never seen a convincing demo of such a system where we were able to follow a non-trivial proof by reading the generated text. The output of such a tool generally is an order of magnitude larger than the proof scripts that were the input, and also it is rather monotone, as it is automatically generated by a computer program. For these reasons, we consider current formalizations not well suited for the communication of mathematics: the proof script files are unreadable, while the generated natural language presentation is generally too verbose and too monotone to keep the reader's attention needed to understand the mathematics.

In practice, to find out what is in a formalization people almost exclusively look at the statements of the lemmas and ignore the proofs (which generally are not in the files that they look at anyway). They only use the proofs for *proof engineering*, like modifying a copy of a proof to prove a similar lemma, or changing a proof to make it check faster or extract to a better program.

– Formalization of mathematics is a very labor-intensive activity. A rough estimate of the amount of work needed for the formalization of mathematics is that with the current state of the art it takes about one work-week (five days from nine to five) to formalize one page from an undergraduate mathematics textbook.

  Eventually, we would like the mathematicians to start using our systems for routine formalization of mathematics. However, currently it will be unrealistic to expect working mathematicians to go to the trouble of doing full formalizations. Even with a proper mathematical library (which currently is not available for any of the existing systems) the cost of doing a full formalization is prohibitive in comparison to the benefit.

  At the TYPES workshop of 2002 in Nijmegen, Peter Aczel claimed that in order to get mathematicians involved with the formalization of mathematics, technology is needed for *reasoning with gaps*, where one can leave out the details of a formalization that one considers to be obvious or well-known, and one only needs to formalize the interesting parts. This vision is the focus of this paper.

1.2 Approach

We will define the notion of a *formal proof sketch* for the declarative proof language of the Mizar system [6, 12]. A formal proof sketch is an incomplete Mizar text that only has one kind of error – that justifications do not necessarily justify the steps, the famous `*4` error of the Mizar system – such that it can be completed into a correct Mizar text by adding steps, and references between the steps, to the proofs.

Although this paper uses Mizar as the declarative proof language, the notion of formal proof sketch makes sense for any declarative language. For instance one can have formal proof sketches for Markus Wenzel's Isar language for the Isabelle system [11], or for the proof language of Don Syme's Declare system [9].

1.3 Contribution

'An unfinished Mizar article that has only `*4` errors left in it' is a well-known concept to every Mizar user. However, we think it is new to consider such a text to be more important than just being an intermediate stage during formalization. In fact, we would like to claim that that kind of text might be *more* interesting and useful than a completed Mizar formalization. Also, it is new to use this kind as text as a *presentation* of the contents of a finished formalization.

New also is the observation that many informal mathematical proofs from the literature can be mimicked closely by formal proof sketches. (This is a primarily an observation on the syntax of the Mizar language.) See for examples Sections 2 and 8 below. Finally it is new to consider the incomplete justifications of a formal proof sketch to be natural targets for automated theorem proving, as discussed in Section 7.

As far as we know no system exists yet that uses the same declarative language both on the formal proof sketch level and on the formalization level. The main contribution of this paper is the proposal of building such a system.

1.4 Outline

In Section 2 we give a detailed example of a formal proof sketch. In Sections 3 and 4 we discuss the notion of formal proof sketch. In Sections 5 and 6 we compare formal proof sketches with other approaches to proof presentation. In Section 7 we point out the relation between formal proof sketches and automated theorem proving. In Section 8 we present two more examples.

2 An Example: Four Versions of the Irrationality of $\sqrt{2}$

On pages 39 and 40 of the fourth edition of Hardy and Wright's *An Introduction to the Theory of Numbers* [3], we find the following proof:

THEOREM 43 (PYTHAGORAS' THEOREM). $\sqrt{2}$ *is irrational.*

The traditional proof ascribed to Pythagoras runs as follows. If $\sqrt{2}$ is rational, then the equation

$$a^2 = 2b^2 \qquad (4.3.1)$$

is soluble in integers a, b with $(a, b) = 1$. Hence a^2 is even, and therefore a is even. If $a = 2c$, then $4c^2 = 2b^2$, $2c^2 = b^2$, and b is also even, contrary to the hypothesis that $(a, b) = 1$. □

If we mimic this text in Mizar syntax, it turns out that we get surprisingly close:

THEOREM Th43: sqrt 2 *is irrational* :: PYTHAGORAS' THEOREM

PROOF assume sqrt 2 is rational; consider a, b such that

4_3_1: $$a\,\hat{}\,2 = 2 * b\,\hat{}\,2$$

and a, b are_relative_prime; $a\,\hat{}\,2$ is even; then a is even; consider c such that $a = 2 * c$; $4 * c\,\hat{}\,2 = 2 * b\,\hat{}\,2$; then $2 * c\,\hat{}\,2 = b\,\hat{}\,2$; b is even; thus contradiction; END;

This is *almost* a correct Mizar text, because the reasoning is too high level for the system to know why the steps in this proof are allowed (to turn it into a correct Mizar formalization one needs to add intermediate steps and labels, as shown below). However, this text is *syntactically* completely correct according to the Mizar grammar. To stress this, we repeat it in conventional Mizar layout:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 consider a,b such that
4_3_1: a^2 = 2*b^2 and
  a,b are_relative_prime;                              ←1
 a^2 is even;                                          ←2
 a is even;                                            ←3
 consider c such that a = 2*c;                         ←4
 4*c^2 = 2*b^2;                                        ←5
 2*c^2 = b^2;                                          ←6
 b is even;                                            ←7
 thus contradiction;                                   ←8
end;
```

If one runs this through the Mizar system, one gets eight times the error '*this inference is not accepted*' (as indicated by the arrows in the right margin).

The following text is a completed Mizar formalization of the same proof, where the errors have been eliminated. The parts that correspond to the previous text have been underlined:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 then consider a,b such that
A1: b <> 0 and
A2: sqrt 2 = a/b and
A3: a,b are_relative_prime by Def1;
A4: b^2 <> 0 by A1,SQUARE_1:73;
 2 = (a/b)^2 by A2,SQUARE_1:def 4
```

```
  .= a^2/b^2 by SQUARE_1:69;
 then
4_3_1: a^2 = 2*b^2 by A4,XCMPLX_1:88;
 a^2 is even by 4_3_1,ABIAN:def 1;
 then
A5: a is even by PYTHTRIP:2;
 then consider c such that
A6: a = 2*c by ABIAN:def 1;
A7: 4*c^2 = (2*2)*c^2
  .= 2^2*c^2 by SQUARE_1:def 3
  .= 2*b^2 by A6,4_3_1,SQUARE_1:68;
 2*(2*c^2) = (2*2)*c^2 by XCMPLX_1:4
  .= 2*b^2 by A7;
 then 2*c^2 = b^2 by XCMPLX_1:5;
 then b^2 is even by ABIAN:def 1;
 then b is even by PYTHTRIP:2;
 then 2 divides a & 2 divides b by A5,Def2;
 then
A8: 2 divides a gcd b by INT_2:33;
 a gcd b = 1 by A3,INT_2:def 4;
 hence contradiction by A8,INT_2:17;
end;
```

Note that this formalization does not much resemble the Hardy and Wright proof anymore but looks like the kind of 'code' that is customary in proof assistants.

This example shows the three kinds of proof texts that we will consider in this paper:

– the first text fragment is an *informal English proof*
– the second and third are a *formal proof sketch* (in two different layouts)
– the fourth is a *full formalization*

We imagine a prover interface in which all these variant texts are present next to each other, connected by hyperlinks.

Note that the informal English proof and the formal proof sketch are very similar. Also note that the formal proof sketch and the full formalization are both written in the same formal language. Finally note that the formal proof sketch occurs as a 'skeleton' in the text of the full formalization.

This paper proposes to take these formal proof sketches seriously. We claim that a formal proof sketch can be used to precisely communicate mathematics (although the computer cannot check the correctness, a human can, cf. the proposition on page 8). We also claim that a formal proof sketch is the best way to present a formalization, as a 'road map' to the full text of the formalization.

For two more examples of an informal English proof together with a formal proof sketch version, see Section 8 at the end of this paper.

3 Formal Proof Sketches

We now give the informal definition of a formal proof sketch. This definition can be made rigorous but we will not do that here.

Definition. A *formal proof sketch* is a text in the syntax of a declarative proof language that was obtained from a full formalization in that language by removing some proof steps and references between proof steps. The only errors (according to the definition of the proof language) in such a stripped formalization should be *justification errors*: the errors that say that a step is not sufficiently justified by the references to previous steps.

Some people might object to the name 'formal proof *sketch*', claiming that these formal proof sketches are designed to closely follow the informal proofs of mathematics, and that mathematicians consider those to be *proofs*, and not sketches. However, we would like to take the formalizer's point of view that formal proof sketches leave parts of the mathematics implicit, and that therefore we can use the word 'sketch'.

If we specialize the notion of a formal proof sketch to the Mizar proof language, we have the following *formal proof sketch grammar*:

$$\begin{array}{rcl}
\text{statement} & = & \text{proposition justification} \\
 & | & [\,\textit{label}\ \textbf{:}\,]\ \textit{term} = \textit{term}\ \text{justification} \\
 & & \quad \{\textbf{.=}\ \textit{term}\ \text{justification}\} \\
\\
\text{proposition} & = & [\,\textit{label}\ \textbf{:}\,]\ \text{formula} \\
\\
\text{formula} & = & \textit{formula} \\
 & | & \textbf{thesis} \\
\\
\text{justification} & = & [\,\textbf{by}\ \textit{label}\ \{\textbf{,}\ \textit{label}\}\,] \\
 & | & \textbf{proof}\ \{\text{step}\ \textbf{;}\}\ [\,\text{cases}\,]\ \textbf{end} \\
\\
\text{step} & = & [\,\textbf{then}\,]\ \text{statement} \\
 & | & \textbf{assume}\ \text{proposition} \\
 & | & \textbf{let}\ \textit{variable}\ \{\textbf{,}\ \textit{variable}\} \\
 & | & (\,\textbf{thus}\ |\ \textbf{hence}\,)\ \text{statement} \\
 & | & [\,\textbf{then}\,]\ \textbf{consider}\ \textit{variable}\ \{\textbf{,}\ \textit{variable}\} \\
 & & \quad \textbf{such that}\ \text{proposition justification} \\
 & | & \textbf{take}\ \textit{term}\ \{\textbf{,}\ \textit{term}\} \\
 & | & \textbf{set}\ \textit{variable} = \textit{term} \\
\\
\text{cases} & = & \textbf{per cases}\ \text{justification}\ \textbf{;} \\
 & & \quad \big\{\textbf{suppose}\ \text{proposition}\ \textbf{;}\ \{\text{step}\ \textbf{;}\}\big\}
\end{array}$$

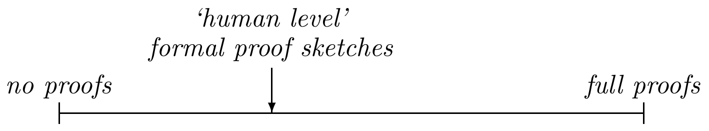
Note that we do not propose our own proof language here. We use the Mizar system to experiment with formal proof sketches and therefore we do not want to depart from the Mizar syntax. Also note that this grammar is small. Often people seem to have the impression that Mizar has a complex syntax, but the proof part of the language is really not much larger than this.

We can graphically represent the process of removing steps and references from a formalization to obtain a formal proof sketch:

$$\begin{tikzcd}
& \boxed{a} \arrow[d] & & & a \arrow[d] & & & a & \\
& \boxed{b} \arrow[dl] \arrow[d] \arrow[dr] & & & b \arrow[dr] \arrow[ddd, bend right] & & & b & \\
c \arrow[d] & d \arrow[d] & \boxed{e} \arrow[dl] \arrow[ddd] & & & e \arrow[ddl] \arrow[dddd, bend left] & & & e \\
f \arrow[dr] & g \arrow[d] & & & & & & & \\
& \boxed{h} \arrow[d] & & & h \arrow[ddr] & & & h & \\
& i \arrow[dr] & j \arrow[d] & & & & & & \\
& & \boxed{k} & & & k & & & k
\end{tikzcd}$$

The first diagram corresponds to the full formalization. There are eleven proof steps which are labeled a–k. The justifications in the proof are represented by the arrows. For instance step h is justified by references to f and g. In Mizar syntax it would be written as 'h : ... **by** f, g ;'. In the second diagram six of the steps have been removed, but the references between the steps are still present. The third diagram corresponds to the formal proof sketch. Now there are no labels and references to labels left. This is generally what happens in a formal proof sketch that mimics an informal mathematical text.

Two extreme formal proof sketches of a given formalization are those in which all proof steps have been removed,[1] and those in which no proof steps have been removed. These are the end points of a spectrum:

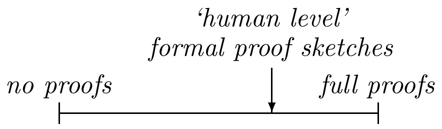


FORMAL PROOF SKETCH SPECTRUM

Most systems have files that correspond to the end points. For instance in the Coq system there are the `.g` and `.v` files, and in the Mizar system there are the `.abs` and `.miz` files. Formal proof sketches give one intermediate levels of proof representation in between these two extremes.

[1] It might seem strange to call such an empty proof a 'proof sketch'. However we think it is natural to include this case in the notion of formal proof sketch, just like 0 is generally considered to be a natural number, and the empty set is considered to be a set.

When we show formal proof sketches to people, we sometimes get the reaction ‘we should build systems that can check them!’ We think it is overtly optimistic to expect that this will be possible soon. However, as proof checking technology improves we can expect the endpoint of ‘full proofs’ to get closer to the formal proof sketches at the ‘human level’:



FORMAL PROOF SKETCH SPECTRUM IN 2048

So formal proof sketches are too high level to be checked automatically for mathematical correctness (else they would be ‘full formalizations’). However, the notion of being a formal proof sketch is *not* an informal notion. If someone presents you with a formal proof sketch, it is meaningful to say that ‘it is correct’ or ‘it is not correct’.

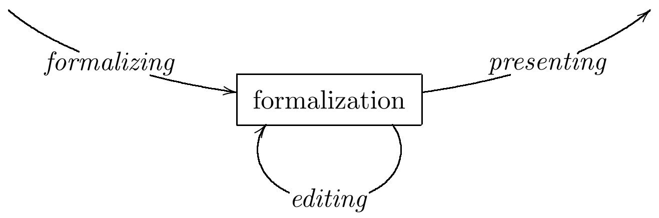
This is related to the proposition:

Proposition. *It is semi-decidable whether a text is a correct formal proof sketch.*

Proof. If a text is a correct formal proof sketch, this can be shown by showing the full formalization from which it was derived. However, as in general it is not decidable whether a given statement is provable in a given context, it can in general not be decided whether a given statement with an empty justification is a formal proof sketch. □

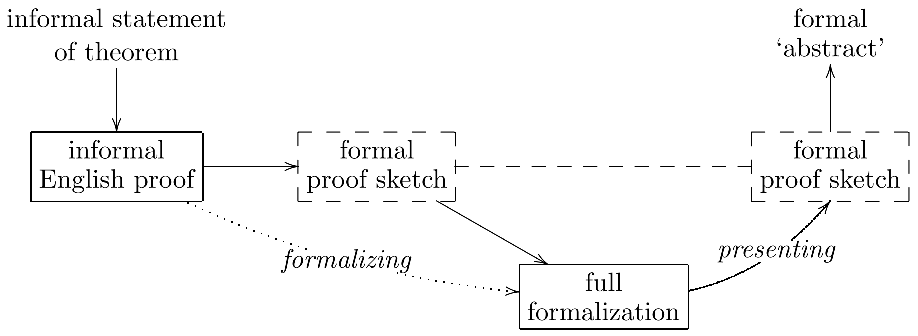
4 The Proof Development Cycle

The development of a mathematical formalization follows the following pattern:



It is important not to forget the activity of editing a formalization. Having a great presentation mode is not very helpful if one needs to deal with the underlying formalization (which often is an unclear tactic script).

If we use formal proof sketches, we can give a more detailed structure to this development diagram:



Here we present development of a formalization from an existing informal mathematical text (as in the example from Section 2). The process consists of two phases. First, one mimics the informal English proof in the formal proof sketch language. This is easy and fast. Second, one 'fleshes out' this formal proof sketch to a full formalization. This generally takes much longer. During this second phase one often discovers that the original formal proof sketch was not correct, and so it will change. At the end one has a matched pair of a formal proof sketch and a full formalization. The full formalization then guarantees the correctness of the mathematics, while the formal proof sketch presents it in an understandable way.

Note that the three levels in this second diagram correspond to the three positions on the formal proof sketch spectrum: the top level corresponds to no proofs, the middle level to 'human level' formal proof sketches, and the bottom level to full proofs.

5 Generated Natural Language Proof

There are many systems to generate proofs in natural language from formalized mathematics. An example of such a system is the MoWGLI system from the University of Bologna [1]. It generates web pages with proofs in natural language from Coq formalizations using the following pipeline:

$$\textbf{Coq script} \xrightarrow{\textsf{Coq}} \text{Coq XML} \xrightarrow{\textsf{XSLT}} \text{OMDoc XML} \xrightarrow{\textsf{XSLT}} \textbf{English}\ \text{HTML}$$

If one looks at the syntax of the output of such a system, it looks very similar to the text that one finds in a formal proof sketch. However, although superficially the two ways of presenting a formalization seem very similar, there are important differences:

– The systems that generate proofs in natural language *automatically* convert a formalization to a presentation. We do not have a method to generate a (reasonable) formal proof sketch from a full declarative formalization without human help. However, one can reduce a full formalization *manually* to a good formal proof sketch, in two stages:

1. The first stage is automatic: one just removes all labels and references to labels.[2] This is the most important phase. It is surprising how much more readable a Mizar text becomes without this 'visual noise'.[3]
2. The second stage consists of removing intermediate steps, just keeping the 'interesting steps' of the proof. The problem is how to decide which steps to drop and which to leave in. We do not know of a good heuristic to decide this from the structure of the proof.[4] Currently it will need manual markup of the full formalization to mark these 'interesting steps'. Inserting that kind of markup will not be much work.

– The size of generated proofs in natural language is an order of magnitude *larger* than the size of the original formalizations, while the size of formal proof sketches is an order of magnitude *smaller* than that of the corresponding full formalizations.
  The accepted opinion seems to be that to deal with this one should *fold* subproofs until one has something of reasonable size. However, we doubt that this will work well. In a procedural prover most of the steps are backward steps, while in a declarative prover most of the steps are forward. In both cases there will be no convenient subproofs to fold. Of course one can write the formalizations to combine forward and backward steps in a way that leads to proofs with a nice folding structure, but that is not very practical. And even if one would do that: our experience with Lamport-style proof (as described in the next section) is that a subproof structure obfuscates the 'narrative' of the proof. If one compares formal proof sketches that mimic existing informal proofs to formal proof sketches that mimic proofs with conveniently foldable subproofs (like Lamport-style proofs), then the former kind has far less subproofs than the latter kind.[5]

6 Lamport's 'How to Write a Proof'

Fleshing out a formal proof sketch along the formal proof sketch spectrum toward a full formalization relates to a proof format described in Leslie Lamport's 'How

[2] Removing *all* labels might be a bit too strong (in long proofs references sometimes can be helpful to understand the proof) but it is a very good first order approximation. The number of references in informal mathematics is a tiny fraction of the number of references in formalized mathematics.

[3] Removing all linkage from a Mizar text means removing the '`then`' keywords as well. But of course this first stage does not need to do so too. In informal mathematics words like *then* are often used to link steps together. We illustrated both styles in the example on page 4: in the informal layout of the formal proof sketch we left two 'then' keywords in, but in the Mizar layout we removed them.

[4] Possibly compiler technology like *basic block analysis* is relevant for this.

[5] When discussing this with Laurent Théry at the TPHOLs 2003 conference, he summarized this opinion by saying 'so you like iteration but you do not like recursion'.

to Write a Proof' [5]. In that paper he describes a hierarchical proof style where each step of the proof recursively is 'clarified' by a subproof until the steps are considered to be obvious. These 'recursive proofs' apparently are very natural and seductive.[6] Note that Lamport's proof format is *not* formal.

An implementation of Lamport-style proof on the web was made in the IMP project of Paul Cairns and Jeremy Gow [2]. They have course notes in topology as a set of web pages. On each of these pages is both an informal proof as well as a Lamport-style proof in which subproofs can be collapsed by clicking on buttons. We compared one of these web pages to our approach: we created formal proof sketches both for the informal proof as well as for the Lamport-style proof. Our experience was that the first formal proof sketch was the more understandable of the two. We had the impression that having many subproofs does obscure the structure of a proof rather than clarify it.

It should be clear that we do not claim that it is bad to have lemmas to isolate important parts of proofs. Subproofs certainly have their place on the lemma level. However, having them on the micro-level of the proofs steps seems not to be very helpful.

Interestingly, when doing this experiment we found that without adding steps the Lamport-style proof was not detailed enough to be accepted by the Mizar system, and when we added that detail it turned out that that it contained some mistakes.

7 Proof Obligations for Automated Theorem Provers

The justifications in a formal proof sketch cannot be checked by the computer because the steps are too large and because there are not enough references. If one does not remove the references (i.e., one takes the formal proof sketch that corresponds to the middle graph on page 7), then one might consider the inferences that need to be justified. For the example from Section 2 all but one inference turn out to be a simple algebraic problem:

$$\begin{array}{lr}
1\rightarrow & b \neq 0 \ \wedge\ \sqrt{2} = a/b \ \vdash\ a^2 = 2b^2 \\
2\rightarrow & b \in \mathbb{Z} \ \wedge\ a^2 = 2b^2 \ \vdash\ 2 \,|\, a^2 \\
3\rightarrow & a \in \mathbb{Z} \ \wedge\ 2 \,|\, a^2 \ \vdash\ 2 \,|\, a \\
5\rightarrow & a^2 = 2b^2 \ \wedge\ a = 2c \ \vdash\ 4c^2 = 2b^2 \\
6\rightarrow & 4c^2 = 2b^2 \ \vdash\ 2c^2 = b^2 \\
7\rightarrow & b \in \mathbb{Z} \ \wedge\ c \in \mathbb{Z} \ \wedge\ 2c^2 = b^2 \ \vdash\ 2 \,|\, b \\
8\rightarrow & (a,b) = 1 \ \wedge\ 2 \,|\, a \ \wedge\ 2 \,|\, b \ \vdash\ \perp
\end{array}$$

The Mizar notion of inference (called 'obviousness') is rather weak, and therefore it is interesting to give these problems – together with the relevant lemmas – to a complete first order prover (if it runs long enough and with enough memory it

[6] When discussing formalization of mathematics with Hendrik Lenstra, he reinvented exactly this proof format on the spot.

eventually will find a justification, if it exists). As an experiment we gave these problems[7] to the Meson prover of the HOL system [4]. The results were rather disappointing (the table also shows the number of steps that the Meson prover needed to solve the problems that it could solve):

	Mizar	*Meson*		
1→	−	−		
2→	+	+	61	1.49 *s*
3→	+	+	20	0.14 *s*
4→	+	+	142	1.02 *s*
5→	−	−		
6→	−	−		
7→	−	+	18839	4.71 *s*
8→	−	−		

We claim that justification problems taken from formal proof sketches of existing mathematical proofs are a good target for the automated theorem proving community. In contrast, current problem sets for automated theorem provers like TPTP [8] are mathematically mostly not very interesting. Of course we would like our automated theorem provers to be able to solve non-trivial problems, but first they should be able to solve the problems that humans consider to be sufficiently trivial that they do not need any words in an informal proof.

8 Two More Examples

In Section 2 we showed that a formal proof sketch of an informal mathematical proof can closely mimic the original. We think that this is generally the case:

Claim. *Most existing informal proofs can be faithfully mimicked as a Mizar formal proof sketch.*

We have experimented with formal proof sketches of various existing proofs and in our experience this claim seems justified.

In this section we will show two more examples. The first is a proof that was on a slide of the talk 'Formalizing an intuitionistic proof of the Fundamental Theorem of Algebra' by Herman Geuvers at the 7th Dutch Proof Tools Day in Utrecht. The slide called it a 'romantic proof', in contrast to the 'cool proofs' of formalized mathematics. (To highlight the correspondence to the formal proof sketch, we underlined two random phrases of the proof.)

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

PROOF [after Euclid]
Given n. Consider $k = n! + 1$, where $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.

[7] Conversion from Mizar justifications to first order problems in TPTP format was kindly done for us by Josef Urban of Charles University in Prague.

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

And here is the corresponding Mizar formal proof sketch:

THEOREM $\{n : n \text{ is prime}\}$ is infinite PROOF
for n ex p st p is prime & $p > n$

PROOF :: [after Euclid]
let n; set $k = n! + 1$;
consider p such that p is prime & p divides k;
take p; thus p is prime; thus $p > n$ PROOF assume $p <= n$;
then p divides $n!$;
not p divides $n! + 1$;
thus contradiction; END; END; thus thesis; END;

The other example is from Rob Nederpelt's report on Weak Type Theory [7]:

THEOREM. Let G be a set with a binary operation $\cdot$ and left unit element e. Let H be a set with binary operation $*$ and assume that ϕ is a homomorphism of G onto H. Then H has a left unit element as well.

PROOF. Take $e' = \phi(e)$. Let $h \in H$. There is $g \in G$ such that $\phi(g) = h$. Then

$$e' * h = \phi(e) * \phi(g) = \phi(e \cdot g) = \phi(g) = h,$$

hence e' is left unit element of H. □

And here is the corresponding formal proof sketch of this simple proof:

---

let G, H be non empty HGrStr; let e be Element of G such that e is_left_unit_of G; let phi be map of G, H such that phi is_homomorphism G, H and phi is onto; thus ex e' being Element of H st e' is_left_unit_of H

PROOF take $e' = phi.e$; now let h be Element of H; consider g being Element of G such that $phi.g = h$; thus

$$e' * h = phi.e * phi.g \;.= phi.(e * g) \;.= phi.g \;.= h;$$

end; hence e' is_left_unit_of H; END;

---

9 Conclusion

9.1 Discussion

In Section 1.1 we listed two problems with formalization of mathematics:

The first problem was that formalized mathematics does not look much like ordinary mathematical text. We think that the notion of formal proof sketch is a step toward a solution of this problem.

The second problem was that formalization of mathematics is too much work. We think that by not fully formalizing a proof, but just using formal proof sketches for the interesting parts, we have an approach that is between no check at all and full verification of the correctness. In the formal proof sketch we will be able to write the interesting parts of the proof (like in Lamport's approach) and the system will check those parts, while we do not have to spend time on the trivial details that take most of the time.

9.2 Related Work

Rob Nederpelt has a language called *weak type theory* or WTT [7], for writing formal mathematics in a way that is closer to informal mathematics than currently is the case in formal systems. It is basically a formalization language where steps in the proofs are just stated but not checked. WTT texts are similar to formal proof sketches (they might be considered the formal proof sketches of the Automath language), but there are several big differences. Although it is structurally similar to informal mathematics, a WTT text does not at all resemble the way informal mathematics looks. Also, there is no notion of mathematical correctness for a WTT text (there is only a notion of 'weak type correctness'). Finally there is currently no way to relate a WTT text to a full formalization of the same mathematics.

Laurent Théry describes in [10] a proof representation (which he there calls 'proof format') that is similar to what we call a formal proof sketch here.[8] Nevertheless, his approach differs in some important respects from ours:

- In [10] there are three very different proofs languages: the XML file that contains the 'formal proof sketch' inside the machine, the natural language presentation with which this file is shown in the interface, and the Coq file that holds the proof obligations that have to be proved. In our approach we use the Mizar language for both the formal proof sketch and for the full formalization, and therefore in our approach all these three languages coincide, giving a much more integrated whole.
- In [10] there are *two* levels: the XML level and the Coq level. In our approach there is a *spectrum* of related formal proof sketches (see page 7 below).

[8] We both seemed to be unaware of each other's work when we were doing it (a preliminary paper about formal proof sketches [13] was only available to the attendees of the 7th Dutch Proof Tools Day in Utrecht), and we only found out about the similarities in each other's work at the UITP workshop in Rome.

- In our approach it is possible to take a full Mizar formalization and systematically reduce it to a formal proof sketch. In the approach from [10] there is no similar way to turn a Coq formalization into anything similar to a formal proof sketch.
- We 'reuse' the language of a full-fledged system for our formal proof sketches. Therefore we have a much richer language than the system from [10]. Especially we get the full Mizar type system, which is very mathematical. Also we get features in our language like definitions, proof by cases and iterated equalities.
- Finally, the correctness of the natural deduction 'skeleton' steps in our formal proof sketches are checked by Mizar with respect to the structure of the current goal, in contrast with [10] where even that kind of checking is considered a proof obligation to be handled later. (An example of this kind of checking occurs when the goal is of the form $A \to B$ and the step says '*assume* A'.)

9.3 Future Work

Formal proof sketches should be integrated into a proof development environment. The same proof language should be used for the formal proof sketches and for the full formalizations. In that system one should then try to mimic the proofs of a non-trivial existing mathematical text (like for instance a chapter from a textbook) as a sequence of formal proof sketches. This will be a good test for the formal proof sketch approach in practice.

Acknowledgments. Thanks to Henk Barendregt, Herman Geuvers, John Harrison, Joe Hurd, Paul Jackson, Hendrik Lenstra, Rob Nederpelt, Claudio Sacerdoti Coen, Bart de Smit, Bas Spitters, Dan Synek and Laurent Théry for stimulating discussions on the subject of this paper. Thanks also to the anonymous referees for helpful comments.

References

1. A. Asperti and B. Wegner. MOWGLI – A New Approach for the Content Description in Digital Documents. In *Proc. of the 9th Intl. Conference on Electronic Resources and the Social Role of Libraries in the Future*, volume 1, Autonomous Republic of Crimea, Ukraine, 2002.
2. Paul Cairns and Jeremy Gow. A Theoretical Analysis of Hierarchical Proofs. In Andrea Asperti, Bruno Buchberger, and James Davenport, editors, *Mathematical Knowledge Management, Proceedings of MKM 2003*, volume 2594 of *LNCS*, pages 175–187. Springer, 2003.
3. G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers.* Clarendon Press, Oxford, fourth edition, 1960.
4. John Harrison. Optimizing Proof Search in Model Elimination. In M. A. McRobbie and J. K. Slaney, editors, *13th International Conference on Automated Deduction*, volume 1104 of *LNCS*, pages 313–327, New Brunswick, NJ, 1996. Springer-Verlag.

5. Leslie Lamport. How to Write a Proof. *American Mathematical Monthly*, 102(7):600–608, 1995.
6. M. Muzalewski. *An Outline of PC Mizar.* Fondation Philippe le Hodey, Brussels, 1993. `<http://www.cs.kun.nl/~freek/mizar/mizarmanual.ps.gz>`.
7. Rob Nederpelt. Weak Type Theory, a formal language for mathematics. Technical Report 02-05, Eindhoven University of Technology, Department of Math. and Comp. Sc., May 2002.
8. Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Proc. 12th Conference on Automated Deduction CADE, Nancy/France*, pages 252–266. Springer-Verlag, 1994.
9. Don Syme. Three Tactic Theorem Proving. In *Theorem Proving in Higher Order Logics, TPHOLs '99, Nice, France*, volume 1690 of *LNCS*, pages 203–220. Springer, 1999.
10. Laurent Théry. Formal Proof Authoring: an Experiment. In Christoph Lüth and David Aspinall, editors, *UITP 2003, Rome, Technical Report No. 189, Institut für Informatik, Albert-Ludwigs Universität*, pages 143–159, Freiburg, September 2003.
11. M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents.* PhD thesis, Institut für Informatik, Technische Universität München, 2002. `<http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html>`.
12. F. Wiedijk. Mizar: An Impression. `<http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz>`, 1999.
13. F. Wiedijk. Formal proof sketches. In Wan Fokkink and Jaco van de Pol, editors, *7th Dutch Proof Tools Day, Program + Proceedings*, Amsterdam, 2003. CWI. `<http://www.cs.kun.nl/~freek/notes/sketches.ps.gz>`.