Formal proof sketches

Freek Wiedijk

University of Nijmegen

Abstract. We define the notion of *formal proof sketch* for the mathematical language Mizar. We show by examples that formal proof sketches are very close to informal mathematical proofs. We discuss some ways in which formal proof sketches might be used to improve mathematical proof assistants.

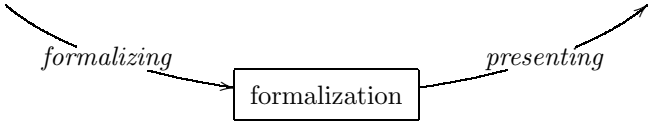
1 Introduction

1.1 Problem

Large mathematical formalizations are often difficult to understand. They generally consist of several files, each containing a long series of definitions and lemmas, with for each lemma a proof which is a maze of steps. As such, they are like large computer programs, which consist of several files, each containing a long series of declarations and functions, with for each function an implementation which is a maze of statements. However the analogy is not perfect. It is much easier to follow the steps through the implementation of a program than to follow the steps through the formalization of a theorem.

There are two levels on which it is difficult to understand a large mathematical formalization. It is difficult to understand the overall structure of the formalization, i.e., to know what are the important definitions and lemmas and in which files to find them. And it is difficult to find one's way through the formal proof of a lemma, i.e., to understand what the structure of that proof is. In this note we address this second difficulty.

Something that makes a formalization easier to understand, does not necessarily make it easier to create. The approach presented in this note makes it easier to read a mathematical formalization, but it will not make it easier to write one. This means that the representation of proofs that this note proposes are *not* meant to be a 'better way' for writing a formal proof. A representation that can be checked by machine is much more involved than the representation that we study here, and it will be as difficult to write as ever. In other words, our work is on right hand side of the following diagram:



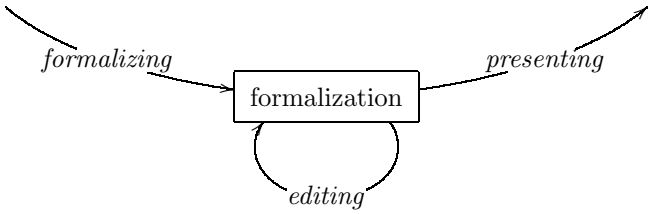
and it is not especially relevant for the left hand side.

Here are some of the main ways to present a formalization in such a way that it becomes easier to understand for a human:

- *Removing the proofs.*
  For instance in the Coq system one can generate a so-called '`.g`' file, which is just the Coq input script with the proofs removed. Likewise, in the Mizar system one generates so-called 'abstracts', again just the formalizations with all proofs removed.
- *Rendering the formulas with mathematical symbols.*
  For instance in the prover interface Proof General one can use the X-Symbols feature of emacs, showing mathematical symbols instead of ASCII strings. In a similar approach, the Theorema system uses the Mathematica front-end to present its formulas in mathematical style. Some systems (like Agda and NuPRL) even have a full-fledged structure editor that displays mathematical formulas in a structural way.
- *Generating natural language text from the formalization.*
- *Merging natural language text with the formalization in the spirit of 'literate programming'.*

Generally the problem with a formal proof is that it is big (because the steps are small), and that one looses grip on the whole because of this. The last two approaches in this list tend to make a formalization even bigger. In contrast with this we believe that to make a formalization accessible, one has to make it smaller.

Many people think that in order to make a formalization easier to understand, one has to transform it. However this means that one creates distance between the formalization and its presentation. This can be a problem if we add a third arrow to the diagram:



To be able to successfully edit a formalization, one has to look at the presentation (in order to be able to understand it), but one also has to work in the underlying formalization itself (because that is where things really happen). If those two versions of the proof differ too much, then the presentation does not help. In that case, for a user who just understands the presentation, the formalization will be a static thing (because it cannot easily be modified) instead of a 'living proof'. Therefore, we claim that the presentation should be as close to the original formalization as possible.

In this paper we are looking for a presentation of a formalization which is:

– *Shorter than the full formalization*, in the sense that the most trivial details of the proof are not shown.
– *Close to the full formalization*, in the sense that it is easy to go back and forth between the two and to find matching locations in both versions.

1.2 Approach

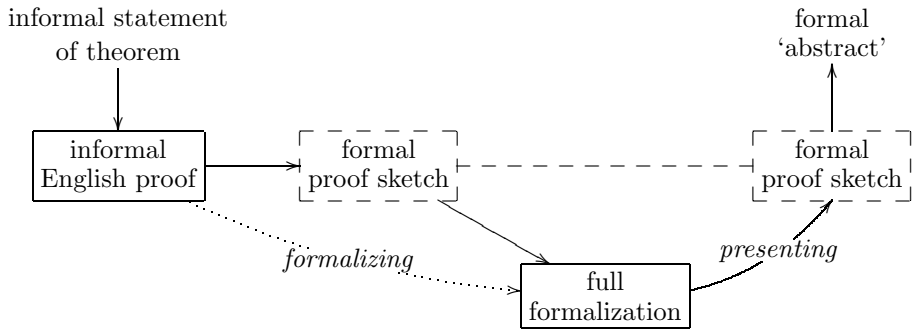
We introduce the notion of *formal proof sketch*. A formal proof sketch is a notion related to the mathematical language Mizar [5, 7]. A formal proof sketch is a completely correct Mizar text, apart from errors `*4` and `*1`. These errors say that reasoning steps are not justified. (Therefore, in a formal proof sketch one can leave out all labels and justifications to make it more readable.)

A formal proof sketch falls between a formalization with full proofs and an abstract with all proofs removed. In a formal proof sketch some of the steps, as well as the connections between the steps, have been removed.

A formal proof sketch is called *correct*, if one can add labels, justifications and steps to it, in such a way that one gets a correct Mizar formalization. We call such a formalization the *completion* of the sketch. Being correct is a semi-decidable notion. If a sketch is correct one can show so by exhibiting a completion, but if it is not, there is no guaranteed way to find out.

The observation of this note is that, given an informal natural language proof, it is possible to write a correct formal proof sketch which is linguistically very close to it. In this note we will show two examples of this, one in Section 2, and one in Section 9.

The process of formalizing an informal proof, and then presenting it through a formal proof sketch, is represented in the following diagram:



For our first example the informal English proof and the formal proof sketch are in Section 2, while the full formalization is in Section 4. For our second example, all three representations are next to each other in Section 9.

When completing a formal proof sketch into a full formalization, sometimes one modifies the structure of the proof a bit. This makes the formal proof sketch change too: hence the two boxes for the formal proof sketch in the diagram. We show an example of such a change in Section 9.

1.3 Related Work

Rob Nederpelt has a language called *weak type theory* or WTT [6], that is meant to formalize mathematics in a way that is closer to natural language than currently is the case in formal systems. It is basically a formalization language like the others, but without any proofs. We are not convinced that having a separate language for this purpose is desirable or even necessary. To us Mizar abstracts (which also are formalizations without proofs) are at least as readable as Nederpelt's WTT formalizations.

Paul Jackson told us that in the NuPRL community it is customary to present proofs by showing the statements from the goals interleaved with the tactics from the proof script. To improve the presentation one then 'groups' the tactics and gets a much shorter presentation. This is very similar to what we describe in this note. The main difference is that in the case of NuPRL it is not easy to write the proof sketch *before* working out the full formalization. In the case of Mizar it is. Another difference is that in the case of Mizar the result is much closer to that informal English version of the proof than in NuPRL.

1.4 Contribution

Our contribution is twofold:

- We note that it is possible to get very close to informal mathematical English using Mizar syntax, as long as one does not mind omitting justifications for the steps.
- We note that one can give a formal definition of a notion called *formal proof sketch* for this. (Note that in this note we do not give this formal definition, but it does exist.)

1.5 Outline

The plan of the paper: in Section 2 we present an informal English proof with its formal proof sketch approximation. In Section 3 we discuss how the typographical differences obfuscate the similarity of those two versions of the proof. In Section 4 we give the full Mizar formalization of the proof. In Section 5 and Section 6 we study whether we can make the full formalization be closer to the formal proof sketch by using a complete first order prover (instead of the restricted inference engine of Mizar) and by using computer algebra. In Sections 7 and 8 we look at possibilities of using formal proof sketches in the interface of a proof assistant. Finally, in Section 9 we present a second example of a formal proof sketch.

2 A formal proof sketch

In Hardy and Wright's *An Introduction to the Theory of Numbers* [3], the irrationality of $\sqrt{2}$ is proved on pp. 39–40 in the following way:

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

$$a^2 = 2b^2 \tag{4.3.1}$$

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 one writes this text in Mizar syntax, it turns out to be almost identical:

```
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;
```

With an appropriate `environ` header, this is correct Mizar, except for eight 'reasoning errors', as indicated by the numbered arrows. We call such a Mizar text that only has reasoning errors a formal proof sketch.

3 Comparing typography

The informal proof and the corresponding formal proof sketch are more similar than might seem at a first glance, because one of them is printed as natural language in a text font, while the other is printed with one step per line in a typewriter font. To stress the similarity of the texts, here they are next to each other:

THEOREM 43. $\sqrt{2}$ *is irrational.*	`theorem Th43:` `sqrt 2 is irrational`
	`proof`
If $\sqrt{2}$ is rational,	`assume sqrt 2 is rational;`
then the equation	`consider a,b such that`
(4.3.1) $a^2 = 2b^2$ is soluble in	`4_3_1: a^2 = 2*b^2 and`
integers a, b with $(a, b) = 1$.	`A1: a,b are_relative_prime;`
Hence a^2 is even,	`a^2 is even;`
and therefore a is even.	`a is even;`
If $a = 2c$,	`consider c such that a = 2*c;`
then $4c^2 = 2b^2$,	`4*c^2 = 2*b^2;`
$2c^2 = b^2$,	`2*c^2 = b^2;`
and b is also even,	`b is even;`
contrary to the hypothesis	`thus contradiction`
that $(a, b) = 1$.	`by A1;`
□	`end;`

On the left there is the Hardy & Wright original, on the right is the formal proof sketch translation. We also can write the 'Mizar' of the formal proof sketch in 'natural language typography' like this:

THEOREM $\sqrt{2}$ *is irrational.*

PROOF Assume $\sqrt{2}$ is rational. Then consider a, b such that

$$a^2 = 2b^2$$

and a, b are relative prime. a^2 is even. Then a is even. Then consider c such that $a = 2c$. $4c^2 = 2b^2$. Then $2c^2 = b^2$. Then b is even. Hence contradiction. □

This text exactly follows the Mizar syntax. The only changes are that the formulas have been written with symbols and that the capitalization and punctuation has been modified slightly.

4 The full proof

If we 'fill out' the formal proof sketch from Section 2 to a full Mizar article, we get:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 then consider a,b such that
B1: b <> 0 and
B2: sqrt 2 = a/b and
A1: a,b are_relative_prime by Def1;
B3: b^2 <> 0 by B1,SQUARE_1:73;
```

```
 2 = (a/b)^2 by B2,SQUARE_1:def 4
  .= a^2/b^2 by SQUARE_1:69;
 then
4_3_1: a^2 = 2*b^2 by B3,REAL_1:43;
 a^2 is even by 4_3_1,ABIAN:def 1;
 then
A2: a is even by PYTHTRIP:2;
 then consider c such that
A3: a = 2*c by ABIAN:def 1;
B4: 4*c^2 = (2*2)*c^2
  .= 2^2*c^2 by SQUARE_1:def 3
  .= 2*b^2 by A3,4_3_1,SQUARE_1:68;
 2*(2*c^2) = (2*2)*c^2 by AXIOMS:16
  .= 2*b^2 by B4;
 then 2*c^2 = b^2 by REAL_1:9;
 then b^2 is even by ABIAN:def 1;
 then b is even by PYTHTRIP:2;
 then 2 divides a & 2 divides b by A2,Def2;
 then
B5: 2 divides a gcd b by INT_2:33;
 a gcd b = 1 by A1,INT_2:def 4;
 hence contradiction by B5,INT_2:17;
end;
```

In this, the formal proof sketch parts have been underlined. This proof needs two 'lemmas' (in the form of redefinitions) that are not present in the Mizar library:

```
 redefine attr x is rational means
:Def1: ex a,b st b <> 0 & x = a/b & a,b are_relative_prime;
```

and:

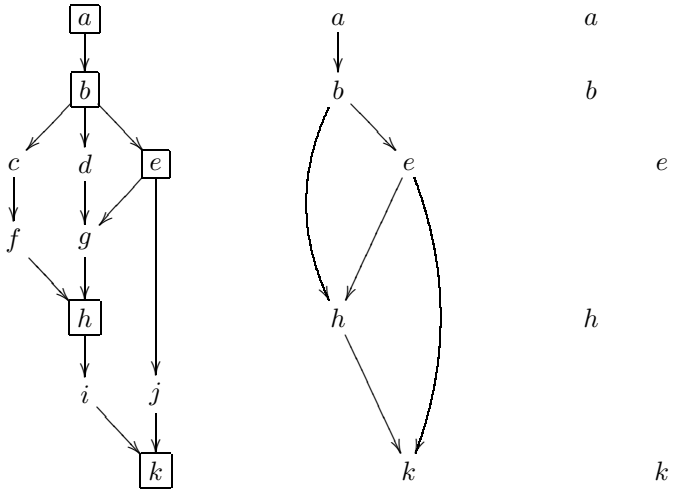
```
 redefine attr a is even means
:Def2: 2 divides a;
```

5 Bigger steps from a stronger first order prover

One can show the relation between the full proof and the formal proof sketch by studying relations between graphs. We draw graphs in which the steps of a proof are represented by the vertices, and where justifications are represented by the edges. The graph that we give here is not the graph of the example (that one would be too big), but a simplified version of the upper part of it.

The graph on the left corresponds to the full article. The steps that are in the formal proof sketch as well, have been put in boxes. The graph on the right corresponds to the formal proof sketch. The graph in the middle is an intermediate version: here the appropriate connections have been left in. These justifications are too 'big' for Mizar to be verified. If Mizar inferences would be

allowed to be arbitrarily difficult first order inferences, the article corresponding to this graph would be correct Mizar.



The Mizar text for the example that corresponds to the middle graph is:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 then consider a,b such that
4_3_1: a^2 = 2*b^2 and
A1: a,b are_relative_prime by Def1,SQUARE_1:73,
   SQUARE_1:def 4,SQUARE_1:69,REAL_1:43;                    ←1
 a^2 is even by 4_3_1,ABIAN:def 1;                          ←2
 then
A2: a is even by PYTHTRIP:2;                                ←3
 then consider c such that
A3: a = 2*c by ABIAN:def 1;                                 ←4
 4*c^2 = 2*b^2 by A3,4_3_1,SQUARE_1:def 3,SQUARE_1:68;      ←5
 then 2*c^2 = b^2 by AXIOMS:16,REAL_1:9;                    ←6
 then b is even by ABIAN:def 1,PYTHTRIP:2;                  ←7
 hence contradiction by A1,A2,ABIAN:def 1,INT_1:def 9,
  INT_2:33,INT_2:def 4,INT_2:17;                            ←8
end;
```

The lists of references to theorems in the justifications get quite long, because the justifications of several steps have been 'put together'. In practice determining those lists is as much work as doing the intermediate steps one-selves. Therefore no-one would write a proof like this, even if Mizar could justify arbitrarily big first order steps.

Still it is interesting to see how well a full first order prover can do on these steps. The HOL Light system by John Harrison [4] implements a full first order

prover called `MESON_TAC`. We gave the first order problems corresponding to the eight inference steps to this prover. The steps in TPTP format were provided to us by Josef Urban. The result of this test was:

	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→	−	−		

The last two columns are number of inferences for the step and the time it took to find those inferences. For the steps for which HOL Light could not find a justification, we let `MESON_TAC` run for at least an hour on a 1 GHz machine, and we let it try at least ten million inferences.

Surprisingly, although the first order prover in HOL Light can *in theory* do arbitrarily complicated first order problems, *in practice* it does about as well as Mizar.

6 Bigger steps from computer algebra

Most of the eight 'problems' from the previous section do not involve logical quantifiers, but are basically algebraic in character:

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

The omitted fourth step (and the omitted part of the first step) involves an existential quantifier.

It is not practical to use a first order reasoning tool to find the steps to get from $a^2 = 2b^2$ and $a = 2c$ to $4c^2 = 2b^2$. This is a problem to be solved algorithmically, and not by searching.

To be able to be allowed steps like this in a formalization, one would like to have an algebraic decision procedure that knows about the notions in these inferences. If one had such a decision procedure, one would be able to get closer to the formal proof sketch.

7 Hyperlinks between formal proof sketch and full formalization

We will now turn to the question of how to integrate formal proof sketches into an interface to make a formalization more accessible.

The simplest is to put two windows next to each other in the interface: one with the full formalization and one with the formal proof sketch. In such an interface it would be nice if those two windows were 'synchronized' in such a way that when scrolling one, the corresponding part of the other always appear next to it. Alternatively one could have 'hyperlinks' between the two representations to easily go from a position in the one to the corresponding position in the other. In such an interface the underlined parts of the proof from Section 4 would be links to the corresponding place in the abstract.

One would like to have a tool to *extract* a formal proof sketch from a Mizar formalization. For that one would like to have some syntax in the Mizar language to 'mark' the steps that should appear in the formal proof sketch. Such a tool has to be a bit subtle, because it should also extract all constructions like '`let`', '`consider`' and '`set`' which introduce variables that are used in the extracted steps in the formal proof sketch.

8 Folding in the interface

Alternatively one could have only one window, with a 'folding interface'. On request of the user some parts of the full formalization might be 'hidden', to make the text look more like the formal proof sketch, and therefore easier to understand.

The naive way to do this, is to complete the formal proof sketch with *subproofs* and then hide the subproofs on request. That way the proof will look like:

```
theorem Th43:
 sqrt 2 is irrational
proof
 assume
C1: sqrt 2 is rational;
 ...; then consider a,b such that
4_3_1: a^2 = 2*b^2 and
A1: a,b are_relative_prime;
 a^2 is even by 4_3_1,ABIAN:def 1;
 then
A2: a is even by PYTHTRIP:2;
 then consider c such that
A3: a = 2*c by ABIAN:def 1;
B4: 4*c^2 = 2*b^2 ...;
C3: 2*c^2 = b^2 ...;
C4: b is even ...;
```

```
 thus contradiction ...;
end;
```

Here the ellipsis symbols '...' represent hidden subproofs of on average six lines each. The disadvantage of this approach is that the full formalization will not be 'ordinary' Mizar. In a normal Mizar article subproofs occur much less often than here.

A more natural way for Mizar to fold parts of a proof is to fold away steps. That way one gets:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 then consider a,b such that
  ... and ... and
A1: a,b are_relative_prime by Def1;
 ...; ... ...; then
4_3_1: a^2 = 2*b^2 by B3,REAL_1:43;
 a^2 is even by 4_3_1,ABIAN:def 1;
 then
A2: a is even by PYTHTRIP:2;
 then consider c such that
A3: a = 2*c by ABIAN:def 1;
B4: 4*c^2 ... ... .= 2*b^2 by A3,4_3_1,SQUARE_1:68;
 ... ...; then 2*c^2 = b^2 by REAL_1:9;
 ...; then b is even by PYTHTRIP:2;
 ...; ...; ...; hence contradiction by B5,INT_2:17;
end;
```

Note that this still is basically the same structure as the formal proof sketch, although it looks much 'messier'. In this version the folded parts are all the same size, one line long:

```
... 1 ≡ B1: b <> 0
... 2 ≡ B2: sqrt 2 = a/b
... 3 ≡ B3: b^2 <> 0 by B1,SQUARE_1:73
... 4 ≡ 2 = (a/b)^2 by B2,SQUARE_1:def 4
... 5 ≡  .= a^2/b^2 by SQUARE_1:69
... 6 ≡   = (2*2)*c^2
... 7 ≡  .= 2^2*c^2 by SQUARE_1:def 3
... 8 ≡ 2*(2*c^2) = (2*2)*c^2 by AXIOMS:16
... 9 ≡  .= 2*b^2 by B4
...10 ≡ then b^2 is even by ABIAN:def 1
...11 ≡ then 2 divides a & 2 divides b by A2,Def2
...12 ≡ then B5: 2 divides a gcd b by INT_2:33
```

```
...13 ≡ a gcd b = 1 by A1,INT_2:def 4
```

A third way of folding is to keep the proof the way it is, but to hide all labels and justifications:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 consider a,b such that
  b <> 0 and
  sqrt 2 = a/b and
  a,b are_relative_prime;
 b^2 <> 0;
 2 = (a/b)^2
  .= a^2/b^2;
 a^2 = 2*b^2;
 a^2 is even;
 a is even;
 consider c such that
  a = 2*c;
 4*c^2 = (2*2)*c^2
  .= 2^2*c^2
  .= 2*b^2;
 2*(2*c^2) = (2*2)*c^2
  .= 2*b^2;
 2*c^2 = b^2;
 b^2 is even;
 b is even;
 2 divides a & 2 divides b;
 2 divides a gcd b;
 a gcd b = 1;
 thus contradiction;
end;
```

This is rather long, but it is much more readable than the formalization with the labels and justifications shown.

9 Another example

We will now present a second example, taken from the linear algebra part of a textbook [2]. This textbook currently has not been published on paper yet, but it is available on the WWW.

On p. 16, we find the lemma:

> **Lemma 2.1.** *Given a linearly independent family $(u_i)_{i\in I}$ of elements of a vector space E, if $v \in E$ is not a linear combination of $(u_i)_{i\in I}$, then*

the family $(u_i)_{i\in I} \cup_k (v)$ *obtained by adding* v *to the family* $(u_i)_{i\in I}$ *is linearly independent (where* $k \notin I$*).*

Proof. Assume that $\mu v + \sum_{i\in I} \lambda_i u_i = 0$, for any family $(\lambda_i)_{i\in I}$ of scalars in K. If $\mu \neq 0$, then μ has an inverse (because K is a field), and thus we have $v = -\sum_{i\in I}(\mu^{-1}\lambda_i)u_i$, showing that v is a linear combination of $(u_i)_{i\in I}$ and contradicting the hypothesis. Thus, $\mu = 0$. But then, we have $\sum_{i\in I} \lambda_i u_i = 0$, and since the family $(u_i)_{i\in I}$ is linearly independent, we have $\lambda_i = 0$ for all $i \in I$. □

The formal proof sketch that corresponds to this is:

```
theorem Lem21:
 u is linearly-independent & not v in Lin(u) implies
  u \/ {v} is linearly-independent
proof
 assume u is linearly-independent & not v in Lin(u);
 assume u \/ {v} is linearly-dependent;
 consider m being Element of K,
  l being Linear_Combination of u such that
   m*v + Sum(l) = 0.E;
 now
  assume m <> 0.K;
  v = -m"*Sum(l);
  v in Lin(u);
  thus contradiction;
 end;
 m = 0.K;
 Sum(l) = 0.E;
 Carrier(l) = {};
 thus contradiction;
end;
```

To show the similarity between the two, here is the informal English version next to the formulas from the formal proof sketch:

Assume that $\mu v + \sum_{i\in I} \lambda_i u_i = 0$,	`m*v + Sum(l) = 0.E`
for any family $(\lambda_i)_{i\in I}$ of scalars in K.	
If $\mu \neq 0$,	`m <> 0.K`
then μ has an inverse (because K is a field),	
and thus we have $v = -\sum_{i\in I}(\mu^{-1}\lambda_i)u_i$,	`v = -m"*Sum(l)`
showing that v is a linear combination of $(u_i)_{i\in I}$	`v in Lin(u)`
and contradicting the hypothesis.	`contradiction`
Thus, $\mu = 0$	`m = 0.K`
But then, we have $\sum_{i\in I} \lambda_i u_i = 0$,	`Sum(l) = 0.E`
and since the family $(u_i)_{i\in I}$ is linearly independent,	
we have $\lambda_i = 0$ for all $i \in I$.	`Carrier(l) = {}`

Here is the full formalization that one gets by completing the formal proof sketch:

```
theorem
 u is linearly-independent & not v in Lin(u) implies
  u \/ {v} is linearly-independent
proof
 assume
A1: u is linearly-independent & not v in Lin(u);
 assume u \/ {v} is linearly-dependent;
 then consider l' being Linear_Combination of u \/ {v}
 such that
A2: Sum(l') = 0.E & Carrier(l') <> {} by VECTSP_7:def 1;
 consider m' being Linear_Combination of {v},
  l being Linear_Combination of u such that
A3: l' = m' + l by Th2;
 set m = m'.v;
A4: m*v + Sum(l) = Sum(m') + Sum(l) by VECTSP_6:43
  .= 0.E by A2,A3,VECTSP_6:77;
 now
  assume
A5: m <> 0.K;
  m*v = -Sum(l) by A4,RLVECT_1:def 10;
  then v = m"*(-Sum(l)) by A5,VECTSP_1:67
   .= -m"*Sum(l) by VECTSP_1:69;
  then
A6: v = (-m")*Sum(l) by VECTSP_1:68;
  Sum(l) in Lin(u) by VECTSP_7:12;
  then v in Lin(u) by A6,VECTSP_4:29;
  hence contradiction by A1;
 end;
 then
A7: m = 0.K;
 Sum(l) = 0.E + Sum(l) by VECTSP_1:7
  .= 0.E by A4,A7,VECTSP_1:59;
 then
A8: Carrier(l) = {} by A1,VECTSP_7:def 1;
 now
  let x be set;
A9: Carrier(m') c= {v} by VECTSP_6:def 7;
  not v in Carrier(m') by A7,VECTSP_6:20;
  hence not x in Carrier(m') by A9,TARSKI:def 1;
 end;
 then Carrier(m') = {} by BOOLE:def 1;
 then Carrier(l) \/ Carrier(m') = {} by A8;
 then Carrier(l') c= {} by A3,VECTSP_6:51;
 hence contradiction by A2,BOOLE:30;
end;
```

The steps that have been underlined in this proof can be omitted (as pointed out by the `relinfer` tool). When one omits them, and then again extracts the formal proof sketch, one gets:

```
theorem Lem21:
 u is linearly-independent & not v in Lin(u) implies
  u \/ {v} is linearly-independent
proof
 assume u is linearly-independent & not v in Lin(u);
 given l' being Linear_Combination of u \/ {v} such that
  Sum(l') = 0.E & Carrier(l') <> {};
 consider m' being Linear_Combination of {v},
  l being Linear_Combination of u such that
   l' = m' + l;
 set m = m'.v;
 m*v + Sum(l) = 0.E;
 now
  assume m <> 0.K;
  v = -m"*Sum(l);
  thus contradiction;
 end;
 Sum(l) = 0.E;
 Carrier(l) = {};
 thus contradiction;
end;
```

This is clearly less readable than the first version of the formal proof sketch.

When one does the experiment with HOL Light's `MESON_TAC` for this example, one discovers that HOL Light is not even able to do the steps that Mizar can do. The reason for this is that the typing of the objects in this proof is quite involved, and it is too complicated to discover the necessary deductions to get the necessary typing judgments by first order proof search.

The two steps in this example which are of the 'algebraic problem' type, are:

$$\mu v + \sum \lambda = 0 \ \wedge \ \mu \neq 0 \ \vdash \ v = -\mu^{-1} \sum \lambda$$

$$\mu v + \sum \lambda = 0 \ \wedge \ \mu = 0 \ \vdash \ \sum \lambda = 0$$

These may seem trivial, but Mizar is not powerful enough to do either one of them in one step. Also, note that these are not algebraic problems about numbers, but about abstract entities like elements of an arbitrary field, vectors in an arbitrary vector space, and 'linear combinations of vectors'.

10 Conclusion

10.1 Discussion

We presented the notion of *formal proof sketch* and looked at some possibilities to use this to make a better interface for presentations of formalizations. The main things that one should consider are:

- The dual window approach from Section 7.
- The third variant of the folding interface from Section 8, where the labels and justifications can be hidden.

To get a 'better' proof checker one should:

- Study algebraic decision procedures as discussed in Section 6.

10.2 Future work

We want to investigate the existing algebraic decision procedures (like for instance in the ICS system [1]) to find out how well they can do the algebraic problems from Section 6.

10.3 Acknowledgments

Thanks to Josef Urban for his translation of the proof obligations in a Mizar article to TPTP format. Thanks to Dan Synek for the example from Section 9. Thanks to Henk Barendregt for encouraging remarks.

References

1. Jean-Christophe Filliâtre, Sam Owre, Harald Rueß, and N. Shankar. ICS: integrated canonizer and solver. To be presented at CAV'2001, 2001.
2. Jean Gallier. *Basics of Algebra and Analysis For Computer Science*. University of Pennsylvania, 2001. Published at:
URL: `<http://www.cis.upenn.edu/~jean/gbook.html>`.
3. G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, fourth edition, 1960.
4. John Harrison. *The HOL Light manual (1.1)*, 2000.
URL: `<http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.ps.gz>`.
5. M. Muzalewski. *An Outline of PC Mizar*. Fondation Philippe le Hodey, Brussels, 1993.
URL: `<http://www.cs.kun.nl/~freek/mizar/mizarmanual.ps.gz>`.
6. Rob Nederpelt. Weak Type Theory, a formal language for mathematics. Draft version, July 2001.
7. F. Wiedijk. Mizar: An Impression.
URL: `<http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz>`, 1999.

A The smallest first order problem from Section 5

We will now show an example of the problems that we gave to `MESON_TAC` in Section 5. It is the smallest of these first order problems, which is step number 3 in the formal proof sketch. That problem is (where the variable `a` has type `Integer`):

$$\frac{\texttt{a\^{}2 is even} \qquad \texttt{for i being Integer holds i is even iff i\^{}2 is even}}{\texttt{a is even}}$$

The corresponding problem that we gave to `MESON_TAC` looks much more complicated:

```
v1_abian (k5_square_1 c1) /\
(!U:V. v1_int_1 U
     ==> ~(v1_abian U /\ ~v1_abian (k5_square_1 U)) /\
         ~(v1_abian (k5_square_1 U) /\ ~v1_abian U)) /\
v1_int_1 c1 /\
v1_int_1 c2 /\
(!U. m1_subset_1 U k1_arytm ==> m1_subset_1 (k5_square_1 U) k1_arytm) /\
(!U. m1_subset_1 U k5_ordinal2 ==> m1_subset_1 (k5_square_1 U) k5_ordinal2) /\
(!U. v1_int_1 U ==> m1_subset_1 (k5_square_1 U) k5_ordinal2) /\
(!U. v1_int_1 U ==> v1_arytm U) /\
(!U V W.
     (~v1_subset_1 U /\ ~v1_subset_1 V /\ m1_subset_1 V (k1_zfmisc_1 U)) /\
     m1_subset_1 W V
     ==> m1_subset_1 W U) /\
m1_subset_1 k5_ordinal2 (k1_zfmisc_1 k1_arytm) /\
(!U. m1_subset_1 U k5_ordinal2
     ==> v1_ordinal1 U /\
         v2_ordinal1 U /\
         v3_ordinal1 U /\
         v4_ordinal2 U /\
         v1_arytm U /\
         v1_int_1 U) /\
(!U. m1_subset_1 U k1_arytm ==> v1_arytm U) /\
(!U. v1_arytm U ==> v1_arytm (k5_square_1 U)) /\
~v1_subset_1 k1_arytm /\
~v1_subset_1 k5_ordinal2 /\
(!U. v4_ordinal2 U ==> v1_int_1 U) /\
(!U. v4_ordinal2 U ==> v1_arytm U) /\
(!U V. r2_hidden U V = m1_subset_1 U V /\ ~v1_subset_1 V) /\
(!U V. r1_tarski U V = m1_subset_1 U (k1_zfmisc_1 V)) /\
(!U V. v1_subset_1 U /\ ~(U = V) ==> ~v1_subset_1 V) /\
(!U. r1_tarski U U) /\
(!U. v3_ordinal1 U ==> r1_tarski U U) /\
(!U V. v3_ordinal1 U /\ v3_ordinal1 V ==> r1_tarski U V \/ r1_tarski V U) /\
(!U V. r2_hidden U V ==> ~r2_hidden V U) /\
```

```
(!U V W.
     r2_hidden U V /\ m1_subset_1 V (k1_zfmisc_1 W)
     ==> m1_subset_1 U W /\ ~v1_subset_1 W)
==> v1_abian c1
```

The reason for this is that we use `MESON_TAC` as an untyped prover (the HOL type system is not powerful enough to accommodate the Mizar type system), so all relevant typing judgments have to be given as first order sentences.

The syntax that is used here does not use Mizar notation, but gives each 'constructor' with a code. These codes translate back to the original Mizar notation according to the following table:

k1_zfmisc_1	bool
k5_ordinal2	NAT
k1_arytm	REAL
k5_square_1	^2
r2_hidden	in
r1_tarski	c=
m1_subset_1	Element
v1_subset_1	empty
v1_ordinal1	epsilon-transitive
v2_ordinal1	epsilon-connected
v3_ordinal1	ordinal
v4_ordinal2	natural
v1_int_1	integer
v1_arytm	real
v1_abian	even

We now transform the HOL problem back to Mizar using this translation. Note the following Mizar type equivalences:

$$\begin{aligned} \texttt{Nat} &\equiv \texttt{Element of NAT} \\ \texttt{Integer} &\equiv \texttt{integer set} \\ \texttt{Real} &\equiv \texttt{Element of REAL} \\ \texttt{Subset of X} &\equiv \texttt{Element of bool X} \end{aligned}$$

The problem that we gave to `MESON_TAC` is in Mizar notation:

```
a^2 is even

for i being integer set holds i is even iff i^2 is even

a is integer

b is integer

for x being Element of REAL holds x^2 is Element of REAL

for n being Element of NAT holds n^2 is Element of NAT

for a being integer set holds a^2 is Element of NAT

for IT being integer set holds IT is real

for D being non empty set, X being non empty Element of bool D,
      IT being Element of X holds IT is Element of D

NAT is Element of bool REAL

for IT being Element of NAT holds IT is epsilon-transitive
      epsilon-connected ordinal natural real integer

for IT being Element of REAL holds IT is real

for x being real set holds x^2 is real

REAL is non empty

NAT is non empty

for IT being natural set holds IT is integer

for IT being natural set holds IT is real

for U,V being set holds U in V iff (U is Element of V & V is non empty)

for U,V being set holds U c= V iff U is Element of bool V

for U,V being set st U is empty & U <> V holds V is non empty

for U being set holds U c= U

for U being ordinal set holds U c= U

for U being ordinal set, V being ordinal set holds U c= V or V c= U

for U being set, V being set st U in V holds not V in U

for U,V,W being set  st U in V & V is Element of bool W holds
      U is Element of W & W is non empty
```

```
a is even
```