The Mathematical Vernacular

Freek Wiedijk
Nijmegen University
<freek@cs.kun.nl>

Abstract

A 'mathematical vernacular' is a formal language for writing mathematical proofs which resembles the natural language from mathematical texts. Several systems (Hyperproof, Mizar, Isabelle/Isar) all basically have the same proof language. It consists of the combination of natural deduction with first order inference steps. In this note we compare these three languages and present a simplified common version.

1 Mathematical Vernaculars

The term 'mathematical vernacular' ('wiskundige omgangstaal' or WOT in Dutch) was introduced by de Bruijn [2] in the eighties. With this term he didn't mean the language that mathematicians actually use to communicate their work in practice (which is a somewhat stylized variant of natural language interspersed with formulas.) Instead he meant a formal language: a system that he had developed to represent mathematics. Supposedly, it was close in style to the actual way that mathematicians communicate, hence the name.

'Vernacular' doesn't seem a word that readily admits a plural: each natural language (English, Dutch, etc.) only has *one* vernacular. However, because many people turned out to have different ideas about the 'best' way to have a formal language resemble ordinary mathematics, one started talking about 'mathematical vernaculars' in the plural. So because many had their own thoughts about what the 'mathematical vernacular' should look like when formalized, the term became the name of a species of formal language.

This paper doesn't want to promote an 'own' variant of the mathematical vernacular concept. Instead, it just makes an observation about already existing formal languages. It turns out that in a significant number of systems ('proof assistants') one encounters languages that look almost the same. Apparently there is a canonical style of presenting mathematics that people discover independently: something like a *natural* mathematical vernacular. Because this language apparently is something that people arrive at independently, we might call it *the* mathematical vernacular.

Now this 'natural' way of representing mathematics basically is the combination of just a few ideas:

- Natural deduction, written down in 'Fitch style': this kind of proof has a natural block structure (just like procedural programming languages from the Algol tradition), corresponding to the 'flag' concept of de Bruijn.
- Big inference steps: a way to state that some forward deduction of the shape:

  $$A_1, \ldots, A_n \vdash A$$

  is valid in first order logic (and that a checker of the language should ask a first order prover to fill in the proof.)

- A choice for an input language instead of a presentation language: the language is not written like natural language, but instead resembles something like a programming language (so for instance it has a '`thus`' statement, instead of saying something like '*therefore we have proven that...*'.)

Abstract mathematics consists of various different activities: defining concepts, stating propositions, giving proofs. In this paper we exclusively focus on the last activity: proving. However, when discussing the mathematical vernacular concept, it turns out that many think that the proof fragment of such a language is not the interesting one. Instead they think that the interesting questions are about concept representation. This is mysterious for various reasons:

- The word 'vernacular' suggest *language*, and hence is about form. But the way concepts are modelled seems to be a question of *content.* (For instance, the question of the best way to formalize the real numbers is not a matter of how to write it down, but of how to shape the mathematics itself.)

- When one looks at the bulk of a mathematical text, most of it is proof. The definitions and statements of propositions are a fraction of the text compared to the proofs (which run on for pages.) Similarly, when looking at formal mathematics, the majority of the 'code' is proof steps.

- There clearly is *not* a consensus about the best way to represent a proof in a formal system. For instance, both the input language (a sequence of tactic invocations) and output language (a proof term in some typed lambda calculus) of the popular type theoretical proof assistants don't resemble the language that is discussed in this paper at all.

This note started by the observation that the Mizar [3, 5] and Isar [4] systems have languages that looked very much the same, but that the *words* that they use for the various constructions are completely different. So this note contains a *translation table* between the two languages, in section 3. That section therefore is present in this paper to show that while the mathematical vernacular discussed here apparently is canonical, its vocabulary certainly is not.

2 Three Systems

Let's consider a very simple mathematical proof. If A, B, C, D are logical formulas and we are given the lemmas:

$$A \to C$$

$$B \to D$$

then from those lemmas we can prove that:

$$\forall x\, (A \land B \to C \land D)$$

(We chose here not to have x occur inside A, B, C and D. This is unnatural, but makes the examples simpler.)

Consider the following three formal versions of a proof. First, in the Hyperproof system of Barwise and Etchemendy [1], we get:

	1	$A \to C$	Given
	2	$B \to D$	Given
	3	$\boxed{a}$	Assume
	4	$A \land B$	Assume
$1, 4 \Rightarrow$	5	C	Log Con
$2, 4 \Rightarrow$	6	D	Log Con
$5, 6 \Rightarrow$	7	$C \land D$	$\land$ Intro
$4 \ldots 7 \Rightarrow$	8	$(A \land B) \to (C \land D)$	$\to$ Intro
$3 \ldots 8 \Rightarrow$	9	$\forall x\, ((A \land B) \to (C \land D))$	$\forall$ Intro

Second, in the Mizar System of Trybulec [3, 5] with:

```
L1: A implies C
L2: B implies D
```

we have:

```
theorem
  for x holds (A & B implies C & D)
proof
  let a;
  assume L4: A & B;
  thus C by L1,L4;
  thus D by L2,L4;
end;
```

(Generally in Mizar a combination of `for` and `implies` like this is written '`for x st A & B holds C & D`', but we use the more explicit syntax here, to make it easier to compare the systems.) Finally, in the Isabelle/Isar system of Wenzel [4], with:

```
L1: "A --> C"
L2: "B --> D"
```

this proof is written like[1]:

```
theorem
  "ALL x. A & B --> C & D";
proof (intro);
  fix a;
  assume L4: "A & B";
  from L1 L4; show "C"; by blast;
  from L2 L4; show "D"; by blast;
qed;
```

(Note that all three systems are powerful enough to prove this statement in one single step: these example proofs are just to show the mechanisms available for proving.)

Clearly, this is three times the same proof, with just the syntax differing. The structure of these proofs has a number of notable features:

[1]The `fix` step really needn't be present (Isar is not *that* similar to Mizar). It's only there to make this proof structurally equal to the other two. Also, this is not the only way to write down this proof with Isar: for instance the `(intro)` method is not necessary if one structures the proof differently.

- A proof consist of a number of lines, each giving a *step* in the proof. Essentially, this proof consists of four steps:
  - introduce the variable a
  - introduce the assumption $A \wedge B$
  - deduce the conclusion C
  - deduce the conclusion D
- There are the basic *natural deduction* ways to reduce statements that have to be proved. For instance the assume step is used to prove an implication. It corresponds to →-introduction, reducing a proof obligation of an implication to that of its conclusion.
- There is a general purpose *first order prover* to prove steps from earlier statements automatically. It is called respectively Log Con, by and blast. Actually, the Hyperproof system has a number of variants of various strengths (Taut Con, Ana Con) and likewise the Isabelle/Isar system has a whole spectrum of automated provers (simp, fast, force, auto.) Also, while Log Con and blast give full first order derivability, Mizar's by is a weaker version of inference (this makes checking decidable and much faster; in the other two systems, when the inference does not hold, the check of correctness may not terminate.)

3 Isar versus Mizar

Both Mizar and Isabelle/Isar make use of 'natural language' keywords like show, thus, hence, then, etc. However, these words apparently don't have a 'natural' meaning, because they mean quite different things in the two systems ($\texttt{show}_{\mathit{Isar}} = \texttt{thus}_{\mathit{Mizar}}$, $\texttt{thus}_{\mathit{Isar}} = \texttt{hence}_{\mathit{Mizar}}$, $\texttt{hence}_{\mathit{Isar}} = \texttt{then}_{\mathit{Mizar}}$, etc.)

Here is a translation table between some of the keywords of the two systems:

Isar	*Mizar*	
.		1
..		1
...		2
	.=	2
	@proof	3
also		2
assume	assume	
by		1
def		4
	deffunc	4
	defpred	4
finally		2
fix	let	
from	by	
	from	1
	given	5
have	*no prefix*	

Isar	*Mizar*	
hence	then	
	hereby	5
let	set	
next + assume	suppose	
note		1
obtain	consider	
presume		5
proof	proof	
qed	end	
show	thus	
sorry		3
then		1
?thesis	thesis	
thus	hence	
with	then + by	
{{	now	
}}	end	

Quite a number of keywords don't have an exact match on the other side. The various categories of this (indicated in the third column of the table) are:

1. *Forward inference.* Isar's from, which corresponds to Mizar's by, is a combination of Isar's note (reference to previous statements) and then ('forward

chaining'). In Mizar these two things can't be separated. Also, in Mizar there's only one inference engine, so Mizar doesn't need keywords (like Isar's `by`, `.` and `..`) to indicate which prover should handle the inference.

2. *Equational reasoning.* Both Isar and Mizar have support for chaining equations together, but their mechanisms work a bit differently and don't map cleanly to each other. Mizar only does this for equalities, and has something special at the start of the chain, while Isar does this for all kinds of relations and has something special at the end (`finally`).

3. *Unchecked proofs.* These constructions are for getting around the obligation of giving a full proof. Isar's `sorry` is used to get rid of a subgoal without proving it, while Mizar's `@proof` is for speeding up the reprocessing of a file by skipping over already correct proofs.

4. *Local definitions.* It's very useful to be able to locally define a name to mean some expression. The mechanisms in Isar and Mizar for this are slightly different and don't map exactly to each other (Isar's `let` is like Mizar's `set`, but it is more powerful because it also can take expressions apart; Isar's `def` doesn't do automatic expansion, so isn't like the local definitions of Mizar.)

5. *Goal shuffling.* When proving some step in the course of a proof, there is a certain amount of freedom of moving the location of the statement around in the proof. The keywords in this category are related to that.

4 A Simplified Vernacular

We now will present a bare bones version of the 'vernacular' that we find in the three systems Hyperproof, Mizar and Isabelle/Isar. We will use Mizar syntax with a slight modification: instead of using `then` we will have the special label `-` for referring back to the most recent *proposition* in scope. Also the meaning of the '`by`' construction will be different from that in Mizar: here we have it mean *full* first-order derivability.

Because we focus here on the deductive style of the language, we won't fix the syntax for first order formulas, but just use the ordinary mathematical notation for that. So the language will be a bit 'fuzzy', because it mixes mathematical formulas and computer text. Hopefully that will make it more easy to see what is part of the formulas and what is part of the deductive structure.

So suppose we are already have syntactic categories *label, variable, term* and *formula.* Then we can give the following grammar[2]:

statement = *proposition justification* `;`

proposition = [*label* `:`] *formula*

justification =
  empty
| `by` *reference* {`,` *reference*}
| `proof` {*step*} [*cases*] `end`

reference =
  label
| `-`

step =
  statement
| `thus` *statement*

[2]This is of course an extremely simple grammar: the *yacc* version has only 55 states.

```
| let variable {, variable} ;
| assume proposition ;
| consider variable {, variable} such that proposition justification ;
| take term {, term} ;
cases = per cases justification ; {suppose proposition ; {step}}
empty =
```

Using the language given by this grammar, the example from the first section becomes:

```
$\forall x\,(A \wedge B \to C \wedge D)$
proof
  let $a$;
  assume L4: $A \wedge B$;
  thus $C$ by L1,L4;
  thus $D$ by L2,L4;
end;
```

For a more complex example, here is an *extremely* explicit proof of the Drinker's principle:

```
$\exists x\,(P(x) \to \forall y\,P(y))$
proof
  $\neg\neg\exists x\,(P(x) \to \forall y\,P(y))$
  proof
    assume H1: $\neg\exists x\,(P(x) \to \forall y\,P(y))$;
    H2: $\forall x\,P(x)$
    proof
      let $a$;
      $\neg\neg P(a)$
      proof
        assume H3: $\neg P(a)$;
        $\exists x\,(P(x) \to \forall y\,P(y))$
        proof
          take $a$;
          assume $P(a)$;
          $\bot$ by -,H3;
          thus $\forall y\,P(y)$ by -;
        end;
        thus $\bot$ by -,H1;
      end;
      thus $P(a)$ by -;
    end;
    $\exists x\,(P(x) \to \forall y\,P(y))$
    proof
      consider $a$ such that $\top$;
      take $a$;
      assume $P(a)$;
      thus $\forall y\,P(y)$ by H2;
    end;
    thus $\bot$ by -,H1;
  end;
  thus $\exists x\,(P(x) \to \forall y\,P(y))$ by -;
end;
```

And here is a more ‘human’ proof:

```
$\exists x\,(P(x) \to \forall y\, P(y))$
proof
  per cases;
  suppose H1: $\forall x\, P(x)$;
    consider $a$ such that $\top$;
    take $a$;
    thus $P(a) \to \forall x\, P(x)$ by H1;
  suppose $\exists x\, \neg P(x)$;
    consider $a$ such that H2: $\neg P(a)$ by -;
    take $a$;
    assume $P(a)$;
    $\bot$ by -,H2;
    thus $\forall y\, P(y)$ by -;
end;
```

Of course, because the statement is provable, all that is really needed is the empty justification:

```
$\exists x\,(P(x) \to \forall y\, P(y))$;
```

5 Proof Steps

We will now describe the language constructions that implement various natural deduction rules:

- `let` or $\forall$-*introduction*

  When the statement that has to be proved is of the form $\forall x\, P(x)$, then after the step ‘`let` y`;`’ the statement to be proved will be $P(y)$. So we have that:

  ```
  ⟨proof of $\forall x\, P(x)$⟩ ≡
      ⟨preliminary steps⟩
      let $y$;
      ⟨proof of $P(y)$⟩
  ```

  Of course the name of the variable in the `let` step usually will be the same as the name of the variable under the $\forall$ quantifier.

  This step corresponds to the natural deduction rule of $\forall$-introduction:

  $$\frac{\Gamma,\, y \vdash P(y)}{\Gamma \vdash \forall x\, P(x)}\ \texttt{let}\ y$$

  (The reason that a rule for *introduction* causes the relevant quantifier to be *omitted* from the goal, is because in deduction the natural way of reasoning is backwards.)

- `assume` or $\to$-*introduction,* $\neg$-*introduction* and *reductio ad absurdum*

  The `assume` step can be used in three ways:

  – First, it is for implication what the `let` step is for universal quantification. So, when the statement to be proved is of the form $A \to B$, then after the step ‘`assume` A`;`’ the statement to be proved will be B. This means that:

$\langle$*proof of* $A \to B\rangle \equiv$
 $\langle$*preliminary steps*$\rangle$
 `assume` A;
 $\langle$*proof of* $B\rangle$

In natural deduction style this is $\to$-introduction:

$$\frac{\Gamma,\, A \vdash B}{\Gamma \vdash A \to B}\ \texttt{assume}\ A$$

– Second, because $\neg A$ is equivalent to $A \to \bot$, the `assume` step also can be used to prove negation:

 $\langle$*proof of* $\neg A\rangle \equiv$
  $\langle$*preliminary steps*$\rangle$
  `assume` A;
  $\langle$*proof of* $\bot\rangle$

 This is $\neg$-introduction:

$$\frac{\Gamma,\, A \vdash \bot}{\Gamma \vdash \neg A}\ \texttt{assume}\ A$$

– Finally, if deducing a contradiction from A proves $\neg A$, deducing a contradiction from $\neg A$ proves A:

 $\langle$*proof of* $A\rangle \equiv$
  $\langle$*preliminary steps*$\rangle$
  `assume` $\neg A$;
  $\langle$*proof of* $\bot\rangle$

 This is the principle of ‘reductio ad absurdum’, reasoning from a contradiction:

$$\frac{\Gamma,\, \neg A \vdash \bot}{\Gamma \vdash A}\ \texttt{assume}\ \neg A$$

- `thus` or $\wedge$-*introduction*

 The converse to the `assume` step (which eliminates an hypothesis from the statement to be proved) is the `thus` step (which eliminates a conclusion from the statement to be proved.) If the statement to be proved is of the form $A \wedge B$, then after a ‘`thus` A `by` ...`;`’ (where the `by`-justification proves A) the statement left to be proved will be B. So:

 $\langle$*proof of* $A \wedge B\rangle \equiv$
  $\langle$*preliminary steps*$\rangle$
  `thus` A `by` ...;
  $\langle$*proof of* $B\rangle$

 This step works modulo associativity of $\wedge$: both A and B can have multiple conjuncts.

 In natural deduction style this is the $\wedge$-introduction rule:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \texttt{thus}\ A$$

 Note that there are two proof obligations above the line in this rule. The justification of the step corresponds to the left-most one.

- `per cases` or $\vee$-*elimination*

  This construction is more complicated than the previous one. Suppose we have to prove B and are able to justify the disjunction $A_1 \vee \ldots \vee A_n$. Then we can prove by cases:

  ```
  ⟨proof of B⟩ ≡
      ⟨preliminary steps⟩
      per cases by ...;
      suppose A_1;
        ⟨proof of B from A_1⟩
      ...
      suppose A_n;
        ⟨proof of B from A_n⟩
  ```

  where the '`by` ...' references justify $A_1 \vee \ldots \vee A_n$. Often the disjunction is provable without conditions (for instance because it has the shape $A \vee \neg A$) and there just is a '`per cases;`'.

  In natural deduction this is $\vee$-elimination:

  $$\frac{\Gamma \vdash A_1 \vee \ldots \vee A_n \quad \Gamma,\, A_1 \vdash B \quad \ldots \quad \Gamma,\, A_n \vdash B}{\Gamma \vdash B}\ \texttt{per cases}$$

  Again, the left-most of the top subgoals is the justification of the construction.

- `consider` or $\exists$-*elimination*

  The `consider` step is used to apply an existential fact. If one is able to justify $\exists x\, P(x)$ one may reason about such an x:

  ```
  ⟨proof of A⟩ ≡
      ⟨preliminary steps⟩
      consider x such that P(x) by ...;
      ⟨proof of A⟩
  ```

  So the '`by` ...' justifies $\exists x\, P(x)$.

  In natural deduction, this is $\exists$-elimination:

  $$\frac{\Gamma \vdash \exists x\, P(x) \quad \Gamma,\, x,\, P(x) \vdash A}{\Gamma \vdash A}\ \texttt{consider}\ x\ \texttt{such that}\ P(x)$$

  Once more, the top-left subgoal is the justification needed for the step.

- `take` or $\exists$-*introduction*

  Finally, to prove an existential statement there's the `take` step. If the statement to be proved has the form $\exists x\, P(x)$, then after a step '`take` a`;`' (for some term a), the statement to be proved will be $P(a)$:

  ```
  ⟨proof of ∃x P(x)⟩ ≡
      ⟨preliminary steps⟩
      take a;
      ⟨proof of P(a)⟩
  ```

  In natural deduction this is $\exists$-introduction:

  $$\frac{\Gamma \vdash P(a)}{\Gamma \vdash \exists x\, P(x)}\ \texttt{take}\ a$$

All the other natural deduction rules don't need a special step of their own, but just are instances of the `by` construction.

If we use the correspondences that we just gave to build the natural deduction tree corresponding to the example proof, we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\ldots}{a,\ A\wedge B\vdash C}\,\texttt{by L1,L4}\quad \dfrac{\dfrac{\ldots}{a,\ A\wedge B\vdash D}\,\texttt{by L2,L4}\quad\dfrac{}{a,\ A\wedge B\vdash}\,\texttt{end}}{a,\ A\wedge B\vdash D}\,\texttt{thus}\ D}{a,\ A\wedge B\vdash C\wedge D}\,\texttt{thus}\ C}{a\vdash A\wedge B\to C\wedge D}\,\texttt{assume L4:}\ A\wedge B}{\vdash \forall x\,(A\wedge B\to C\wedge D)}\,\texttt{let}\ a$$

6 Loose Deduction

The language that we described in the previous sections follows Mizar in the requirement that steps like `assume` and `thus` have to follow the structure of the statement to be proved exactly: so to prove $A_1 \to \ldots \to A_m \to B_1 \wedge \ldots \wedge B_n$ one has to have the lines:

`assume` A_1;
$\ldots$
`assume` A_m;
`thus` B_1 `by` $\ldots$;
$\ldots$
`thus` B_n `by` $\ldots$;

in exactly that order.

This can be loosened considerably, in the following way:

- In order to determine whether it is allowed to do an `assume` step, bring the statement to be proved in negative disjunctive normal form. Then a step:

  $$\texttt{assume}\ A_{i_1} \wedge \ldots \wedge A_{i_m}\,;$$

  is allowed if the normal form is $\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n$. After the step, of course, the new statement is the disjunction of the 'remaining' $\neg A_i$.

- In order to determine whether it is allowed to do a `thus` step, bring the statement in conjunctive normal form. Then a step:

  $$\texttt{thus}\ B_{i_1} \wedge \ldots \wedge B_{i_m}\ \texttt{by}\ \ldots;$$

  is allowed if the conjuctive normal form is $B_1 \wedge B_2 \wedge \ldots \wedge B_n$. Again, after the step the new statement is the conjuction of the remaing B_i.

When using this more loose style of natural deduction, it doesn't matter for the proof whether one phrases a proposition in the common mathematical style like:

$$A_1 \wedge \ldots \wedge A_n \to B$$

or whether one uses the Curried implication that's common in type theory:

$$A_1 \to \ldots \to A_n \to B$$

Both propositions will then behave in the same way.

Acknowledgements Thanks to Gertrud Bauer and Marcus Wenzel for explanations about Isar and to Herman Geuvers and Jan Zwanenburg for valuable discussion.

References

[1] J. Barwise and J. Etchemendy. *Hyperproof.* Number 42 in CSLI Lecture Notes. CSLI Publications, Stanford, 1995.

[2] N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In P. Dybjer et al., editors, *Proceedings of the Workshop on Programming Languages*, Marstrand, Sweden, 1987.

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

[4] M. Wenzel. *The Isabelle/Isar Reference Manual.* TU München, München, 1999. URL: `<http://isabelle.in.tum.de/dist/Isabelle99/doc/isar-ref.pdf>`.

[5] F. Wiedijk. Mizar: An impression. Unpublished,
URL: `<http://www.cs.kun.nl/~freek/mizar/mizarintro.ps.gz>`, 1999.