Naming and scoping in a mathematical vernacular

Gilles Dowek

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

N° 1283

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

NAMING AND SCOPING IN A MATHEMATICAL VERNACULAR

Gilles DOWEK

Août 1990

RR_1283

Naming and Scoping in a Mathematical Vernacular

Nom et Portée des Objets dans un Vernaculaire Mathématique

Preleminary Version

Gilles Dowek

INRIA*

Rocquencourt, France

dowek@margaux.inria.fr

Abstract

A Mathematical Vernacular is a language in which mathematics can be written, both usable for publishing and mathematically fully described. We present a first step toward such a language that uses the Calculus of Constructions as a foundation.

We first design a very simple language that uses four kinds of symbols: variables, constants, axioms and theorems. In this language proofs can be written in a natural deduction style, using the Curry-Howard isomorphism. We then show how the rules $\rightarrow$-intro and $\forall$-intro can be replaced by a more intuitive mechanism in restricting the scope of some symbols and we design a section mechanism that allows to express these scopes. At last we show how the rules $\exists$-elim and $\vee$-elim can be replaced by more intuitive mechanisms in adding new kinds of symbols to the language.

Résumé

Un Vernaculaire Mathématique est un langage destiné à l'expression des Mathématiques, à la fois utilisable comme langage de publication et formellement défini. Nous présentons ici une première ébauche d'un tel langage s'appuyant sur le Calcul des Constructions.

Nous décrivons tout d'abord un langage très simple basé sur quatre sortes de symboles: les variables, les constantes, les axiomes et les théorèmes. Dans ce langage les preuves sont exprimées en déduction naturelle par l'intermédiaire de l'isomorphisme de Curry-Howard. Nous montrons ensuite que les règles $\rightarrow$-intro et $\forall$-intro peuvent être remplacées par un mécanisme plus intuitif consistant à associer une portée à chaque symbole et nous décrivons un mécanisme de structuration du texte qui permet d'exprimer ces portées. Enfin nous montrons que les règles $\exists$-elim et $\vee$-elim peuvent aussi être remplacées par des mécanismes plus intuitifs en ajoutant de nouvelles sortes de symboles au langage.

*This research was partly supported by ESPRIT Basic Research Action "Logical Frameworks".

Introduction

The languages proposed so far to express mathematics may be broadly divided into two kinds: those, like the one in which Bourbaki's *éléments* are written, that are designed to be used by mathematicians and those, like natural deduction, that are used only by logicians for meta-theory, and by machines.

The languages of the second kind have to be as simple and as regular as possible, and they have to be precisely defined. The languages of the first kind are never fully defined and are more lists of writing conventions than real formal languages.

So, at present, there is no mathematical language that is both usable for publishing and mathematically fully described. Thus the situation is quite similar to that of the languages proposed forty years ago to express algorithms: we had natural languages and assembly languages but not yet high level programming languages.

A *Mathematical Vernacular*[1] aims at providing a usable and precisely described mathematical language. Many problems that appear in designing such a Vernacular have already been solved in formal languages, and a Vernacular has to rely on such a formalism. The Vernacular we describe here relies on the Calculus of Constructions [12] and can be seen as a front-end for its proof checker.

Uses of a Mathematical Vernacular

The main difference between natural languages and computers languages is that the latter are very poor. Each of them can be used for only one task. Programming languages can only be used to express algorithms, languages of data bases to express requests, operating systems to manage the resources of a computer, etc. To make the use of machines easier, more general languages may be needed. Natural languages may be too general for this task. A Vernacular might be a good compromise between too specialized and too general languages. As shown in Paulin [15], the mathematical discourse may be used to express programs, specifications and proofs of programs in a uniform fashion.

A second field of applications is Mathematics itself. Since the Vernacular is an interface language for the Calculus of Constructions, it simplifies the use of this system as a proof checker and interactive theorem prover. A Vernacular also could be used as a uniform language for a mathematical data base or an automatic translation system for mathematical papers as proposed in [18].

Designing a Vernacular is answering five questions

The formalization of mathematics can be divided into two main tasks.

In the first, we formalize the mathematical objects. Set theory, for instance, can be considered as a good formalization of the nature of mathematical objects, but other theories such as type theory or category theory may be good challengers. In the second, we formalize the discourse written about these objects. First we formalize the grammar of the propositions. Frege's predicate calculus, which defines the concepts of proposition, predicate, logical connectives and quantifiers is such a grammar. Then we formalize the way these propositions are justified, i.e. the way

[1] The expression *Mathematical Vernacular* has been created by N.G. de Bruijn, he defines it as *the very precise mixture of words and formulas used by mathematicians in their better moments, whereas the "official" mathematical language is taken to be some formal system that uses formulas only.*

proofs are written. Frege-Hilbert proof systems, natural deduction, sequent calculus, type theories, logical frameworks are examples of such formalizations. Type theories are systems that formalize objects, propositions and proofs in a uniform fashion. When designing a formal language it is more convenient to achieve the second task before the first one, since the way the objects can be defined depends on the way that the discourse can be written.

Here we focus on the second of these tasks i.e. we shall only try to formalize the way the discourse is written. This task requires the solution to four problems:

The **first problem** is to make an inventory of all the elementary steps that are allowed in a text. Natural deduction gives a very short and elegant list of elementary steps. Texts are proofs, and new proofs are written with old proofs and about a dozen of rules. This gives a naïve grammar for proofs, and the Curry-Howard isomorphism provides a much more concise and informative one. In a Vernacular we want to write not only theorems but also definitions, declarations and axioms. And because some of the rules of natural deduction are not so natural ($\rightarrow$-intro, $\forall$-intro, $\exists$-elim, $\vee$-elim) we want to replace them by intuitive operations.

Then we want to study the way each kind of elementary step is written beginning with the way proofs are written. For instance when we have two theorems $th1 : P \rightarrow Q$ and $th2 : P$ and we want to give a proof of Q, in type theory, using the Curry-Howard isomorphism we write this proof $(th1\ th2)$. But nobody writes this in a paper. We rather write a sentence such as "We know $P \rightarrow Q$ (th1) and P (th2) and so Q". The reader has to look up the propositions $P \rightarrow Q$ and P and to find by himself the proof of Q. The writer indicates the premises and the conclusion of the proof-step, and the reader has to find the rule or the rules used in this proof-step[2]. So the reader is doing proof synthesis while he reads. Of course the proofs synthesized by the reader must be simple enough for him to find them by himself. We want actually to decompose demonstrations into little lemmas in such a way that a proof of each lemma is "obvious" when previous lemmas are known. For each of these lemmas we want to provide only one piece of information: the list of the previous lemmas that may be used in the synthesis of a proof of the present lemma. So proof synthesis is not only a way to avoid writing obvious and cumbersome details, it also is a way to avoid having an explicit grammar for proofs.

When such a demonstration is read by a machine, a theorem prover must check that a proof of each lemma is indeed obvious when we know the previous lemmas. So the **second problem** is to design a theorem-prover, and the **third** is to describe how the writer indicates to the reader (i.e. the theorem-prover) which among the previous lemmas may be used, for instance with the phrases: "so", "using the two previous propositions", "using theorem (4)", "since we know $P \rightarrow Q$", etc.

The **fourth problem** is to describe how the other sentences (definitions, axioms, etc.) are written. One important part of this problem is to describe the way propositions are written. It is a problem of natural language, we have to look in mathematical textbooks how they are written. For instance, one never writes "$fixpoint(m, f)$" but "m is a fixpoint of f". Here the main question is which grammar formalism is to be used to describe this language.

In this note we describe an elementary Vernacular that attempts to answer only the first problem. In another paper [10] we design a Vernacular with proof synthesis that attempts to answer the second one.

[2]Remark that here proof synthesis is used in a quite different way from the way it is used in an interactive theorem prover with tactics where the conclusion (goal) and the rules (tactics) are given and the premises (subgoals) are synthesized.

1 Variables, axioms, constants and theorems

1.1 Variables and axioms

In the Calculus of Constructions, one can express a proposition P and a proof-term t such that $\vdash t : P$. A proof checker can mechanically check that t is indeed a proof of P. If free variables occur in P they must be declared before the judgment $t : P$ is checked. If axioms are used in t they must be declared before $t : P$ is checked. These declarations of variables and axioms are called *the context* in which the judgment $t : P$ is checked.

Thus, in the formalism, we need three kinds of sentences: declarations of variables, declarations of axioms and judgements that a term t is a proof of a proposition P. In a declaration of a variable, we give the name and the type of the variable. In an axiom, we give the name of the axiom and the assumed proposition. In a judgment we give the proof and the proposition.

1.2 Constants and theorems

To write propositions and proofs it is convenient to define abbreviations. A symbolic name that abbreviates an object is called a constant. A symbolic name that abbreviates a proof is called a theorem. The names defined this way can be used after their definitions, and everything happens as if the symbolic names had been replaced by the terms they abbreviate. In the formalism, two new kinds of sentences allow these definitions: constant definitions and theorems. In a constant definition, we give the name of the constant and the term it abbreviates (and optionally its type). In a theorem we give the name of the theorem, its statement and its proof. A proof checker must verify that the proof is a proof of the statement.

Now that we have theorems, a special kind of sentences for judgments is no longer necessary, these sentences are merely theorems that define a symbolic name that may or may not be used. So there are four basic kinds of sentences in Vernacular: Variable declarations, Axioms, Definitions and Theorems. These sentences are written with four kinds of elements: types, propositions, objects and proofs.

2 Locality

2.1 Proofs written in Natural Deduction

In order to express a proof in this simple minded Vernacular, a mathematician would have to translate his proof in a proof-term via the Curry-Howard isomorphism. This translation is quite cumbersome and should be done automatically.

One way to avoid entire translation of the proofs is to use the abbreviation facility. Some proofs can be written that way nearly in natural deduction. For instance, we have three axioms $h1 : P \to Q \to R$, $h2 : P$ and $h3 : Q$, and want to prove a theorem th whose statement is R.

The natural deduction proof is:

$$\dfrac{\dfrac{\Gamma \vdash P \to Q \to R \quad \Gamma \vdash P}{\Gamma \vdash Q \to R} \qquad \Gamma \vdash Q}{\Gamma \vdash R}$$

This proof can be translated into the term $(h1\ h2\ h3)$, therefore we can write:

```
Theorem th R Proof (h1 h2 h3).
```

But we also may notice that the rule $\rightarrow$-elim is applied twice and cut the proof in two steps:

```
Theorem th1 (Q -> R) Proof (h1 h2).
Theorem th R Proof (th1 h3).
```

Each of these steps is an $\rightarrow$-elim rule and the term is always of the same kind: the application of a symbol to another. We could change the grammar very superficially and write, for instance:

```
Theorem th1 (Q -> R) Using h1, h2.
Theorem th R Using th1, h3.
```

The sentence `Theorem <name> <statement> Using <name1>, <name2>.` means that a proof of the proposition `<statement>` can be found by $\rightarrow$-elim from the propositions `<name1>` and `<name2>`. A proof checker would check that the proof-term (`<name1> <name2>`) is well typed and a proof of the proposition `<statement>`.

So we can write proofs without explicitly using the Curry-Howard isomorphism.

2.2 The $\rightarrow$-intro and $\forall$-intro rules

The problem is that this trick cannot be applied with all rules, because it assumes that the context is the same for all the steps of the proof, and this is false when we use the $\rightarrow$-intro and $\forall$-intro rules. The reason for this problem is that in natural deduction there are three symbols to express consequence, one in the language: the arrow ($\rightarrow$) and two in the meta-language: the turnstile ($\vdash$) and the line used to write rules (—). In other words, in natural deduction we deduce sequents from sequents, and these sequents are themselves deductions. In Vernacular, we do not want this extra level. We want to deduce directly propositions from propositions.

In natural deduction, because of the extra level contexts are considered as objects and one can compute with them. This notion of context is quite far from the ordinary one. In a demonstration in a book or a paper, there is a total ordering among sentences. A sentence is written before or after another, and the context of a sentence is the list of sentences written before it.

A way of understanding the $\rightarrow$-intro rule is the following. Let Γ be the current context. If in another book, where the context is $\Gamma@[h : P]$, it would have been possible to prove Q, then we can deduce $P \rightarrow Q$ in the current context. In order to justify this proposition we copy the proof we could have written in the other book. In this proof we can use the axiom P. The axiom P has a restricted scope in the text. To use this rule we must:

- 1. assume a new axiom P,
- 2. prove Q,
- 3. announce that we stop assuming P.

The effect of this magical third step is to erase the axiom P and modify the theorem Q in $P \rightarrow Q$.

A solution could be to write this:

```
Axiom h:P.
Theorem th Q Proof <proof>.
Discharge h.
```

In a proof written in a book the two first steps are quite explicit. The third one is always implicit. For a mathematician, there is little difference between proving Q under the assumption P and proving $P \rightarrow Q$. Hence there may be only one symbol to express consequence. This is related to the fact that implication is the internalization in the language of intuitionistic deduction.

Since this third step is implicit in mathematical books it must also be in Vernacular. We write this:

```
Theorem th.
     Hypothesis h:P.
Statement Q.
Proof <proof>.
```

The theorem definition has been cut in three lines and the axiom has become an hypothesis, i.e. a local axiom. The hypothesis is local to the theorem.

For instance, the following defines the theorem $I = [h : P]h : P \rightarrow P$:

```
Theorem I.
      Hypothesis h:P.
Statement P.
Proof h.
```

It is also possible to insert a hypothesis between the statement and the proof. In this case the statement must be the statement of the whole theorem. For instance:

```
Theorem I.
Statement P -> P.
      Hypothesis h:P.
Proof h.
```

When the proof checker meets the instruction `Proof t.` in a context Γ, it computes the type of t in Γ, eliminates the local hypotheses declared since the last instruction `Statement` (i.e. erases these hypotheses of the context, λ-abstracts them in t and π-abstracts them in the type of t), checks that the type obtained that way is the same as the one given in the last instruction `Statement` (otherwise it fails) then eliminates the local hypotheses declared since the last instruction `Theorem` and adds this new theorem to the context.

Local variables can also be defined in order to use the rule $\forall$-intro.

A more general mechanism of locality can be defined, where any sentence can be local to any sentence, out of their scopes local objects and theorems are replaced by their values and local variables and axioms are λ-abstracted in objects and proofs and π-abstracted in types and propositions.

2.3 Completeness of the Vernacular

Now that we have defined this locality mechanism we can remove the $\rightarrow$-intro and the $\forall$-intro rules. A way to do this is to restrict the proof-terms to be of the form u where u is a symbol referring

to a proof of a known proposition, $(u\ v)$ where u and v are symbols referring to proofs of known propositions and $(u\ t)$ where u is a symbol referring a proof of a known proposition and t is an object i.e. a term which type is of type $Type$.

Let us prove that this Vernacular is complete, i.e. that for each context Γ and proposition P and proof t such that $\Gamma \vdash t : P$, there exists a text in vernacular that defines a theorem $th = t : P$.

We build such a text by induction on t: since the type of the type of t is $Prop$, t is different of the symbols $Prop$ and $Type$ and it is not a product. So t is a variable, an application or a abstraction.

- If t is a variable of Γ then the following text defines a symbol $th = t : P$:

```
Remark th P Proof t.
```

- If t is an application $t = (t_1\ t_2)$ we let $(x : T)T'$ be the type of t_1 and T be the type of t_2. $T'[x \leftarrow t_2] = P$.

  If the type of T is $Prop$ then by induction hypothesis there exists two texts `<text1>` and `<text2>` which define symbols $th1$ and $th2$ which are proofs of $(x : T)T'$ and T. The following text defines a symbol $th = t : P$:

```
Remark th.
Statement P.
   <text1>
   <text2>
Proof (th1 th2).
```

  If the type of T is $Type$ then by induction hypothesis, there exist a text `<text>` that defines a symbol $th1$ proof of $(x : T)T'$. The following text defines a symbol $th = t : P$:

```
Remark th.
Statement P.
   <text>
Proof (th1 t2).
```

- If t is an abstraction $t = [x : T]t_1$ we let T' be the type of t_1 in $\Gamma@[x : T]$. $P = (x : T)T'$.

  By induction hypothesis, there exists a text that defines a symbol $th1$ proof of T' in the context $\Gamma@[x : T]$. Let this text be:

```
Remark th1.
   <text1>
Statement <stat>.
   <text2>
Proof <proof>.
```

  We transform this text in the following one which defines a symbol $th = t : P$:

```
Remark th.
   Variable/Hypothesis x:T.
   <text1>
Statement <stat>.
   <text2>
Proof <proof>.
```

2.4 Sections

A common style in writing books is to prove in one section theorems which share an hypothesis. For instance in a book of group theory there may be a chapter dedicated to abelian groups. All the theorems of this chapter are of the form $((Abelian\ \ G) \rightarrow P)$. In order to prove such a theorem we can assume locally that G is abelian, and then prove P. But what we want to do is to write at the beginning of the chapter "In this chapter we assume G abelian", and then write the statements of the theorems: P. Inside the chapter, the theorem is considered as a proof of P, and outside as a proof of $((Abelian\ \ G) \rightarrow P)$. We write this:

```
Section Abelian_groups.
   Hypothesis h:(Abelian G).
   Theorem th1 P Proof <proof1>.
   Theorem th2 Q Proof <proof2>.
   ...
End Abelian_groups.
```

The hypothesis h is local to the whole section and $th1$ and $th2$ are global. If a sentence is local to a section it is local to all the global sentences defined after it. So outside the section, if this sentence defines a local object or a remark (i.e. a local theorem), the symbol is replaced by its value in global sentences, and if it is a local variable or an hypothesis (i.e. a local axiom) it is λ-abstracted in objects and proofs and π-abstracted in types and propositions.

2.5 Interpretation of the locality mechanism as sections

When we have a sentence S local to another sentence T, we can consider T as a little section. For instance:

```
Theorem th1.
      Hypothesis h:<hyp>.
Statement <statement>.
Proof <Proof>.
```

can be seen as an abbreviation for:

```
Section th1.
   Hypothesis h:<hyp>.
   Theorem th1 <statement> Proof <Proof>.
End th1.
```

A problem occurs when we define a sentence local to another local sentence, for instance if the theorem *th1* was a remark (i.e. a local theorem), it would be local to the section *th1*. And everything would be eliminated when the section is closed.

So the remark *th1* must survive to the end of the section *th1* but must die the next time a section is closed, the elementary information local/global is not enough any more and a proof-checker must keep for each sentence a integer which indicates the scope of this sentence.

So we have seen that natural deduction deals with context and locality in a non natural way and we have replaced in the Vernacular the rules $\rightarrow$-intro and $\forall$-intro by this mechanism of locality and sections. The power of the two systems is exactly the same, it is only a question of convenience that makes us prefer the section mechanism to these two rules.

3 Virtual Witnesses

The rule $\exists$-elim is also quite unnatural in natural deduction. Let us give an example:

Let T be a type and A and B two unary predicates on this type. We assume the two following axioms:

$u : \exists x : T\ (A\ x)$

$v : \forall x : T\ ((A\ x) \rightarrow (B\ x))$

```
Axiom u:(ex T [x:T](A x)).
Axiom v:(x:T) ((A x) -> (B x)).
```

We want to deduce $\exists x : T\ (B\ x)$. In a book we would write: "*Let y be an element of T such that $(A\ y)$. From v we deduce $((A\ y) \rightarrow (B\ y))$. We know $(A\ y)$ so $(B\ y)$. So $\exists x : T\ (B\ x)$.*"

3.1 $\exists$ in Natural Deduction

This mechanism is translated in natural deduction as the rule $\exists$-elim:

$$\dfrac{\dfrac{\pi_1}{\exists x : T\ (P\ x)} \qquad \dfrac{[y : T \quad y\_prop : (P\ y)]}{\dfrac{\pi_2}{Q}}}{Q}$$

where y is not free in Q.

There are a few differences between this rule and the natural mechanism. In Vernacular, it is the fact that we have proved π_1 which allows the declaration of the variable y and the hypothesis *y_prop*, these declarations are written with a special syntax that refers to the proof π_1, the proof π_2 is always written after π_1 and we do not mention the rule $\exists$-elim: when we have proved Q, the proof is over. It means that the special syntax used for the declaration of the variable y and the hypothesis *y_prop* indicates that when we will have proved a proposition Q in which y and *y_prop* are not free we will be able and we will have to compute another proof π such that y and *y_prop* are not free in π.

3.2 $\exists$ in the Calculus of Constructions

In the Calculus of Constructions we modify a little the rule $\exists$-elim in order to reduce the abstraction on y and y_prop to the rules $\forall$-elim and $\rightarrow$-elim. This gives the rule:

$$\frac{\dfrac{\pi_1}{\exists x : T\ (P\ x)} \qquad \dfrac{\pi_3}{\forall y : T\ ((P\ y) \rightarrow Q)}}{Q}$$

where y is not free in Q. So the quantifier $\exists$ can be defined in the formalism as an abbreviation:

$$\exists = [T : Type][P : T \rightarrow Prop](Q : Prop)((y : T)((P\ y) \rightarrow Q)) \rightarrow Q$$

If we have a proof π_1 of $\exists x : T\ (P\ x)$ and a proof π_3 of $\forall y : T\ (P\ y) \rightarrow Q$ then the term $\pi = (\pi_1\ Q\ \pi_3)$ is a proof of Q. So the previous example must be written:

```
Theorem th.
Statement (ex T [x:T](B x)).
   Remark rem1.
   Statement (y:T) ((A y) -> (ex T [x:T](B x))).
         Variable y:T.
         Hypothesis y_prop:(A y).
         Remark rem2 (B y) Proof (v y y_prop).
   Proof (ex_intro T [x:T](B x) y rem2).
Proof (u (ex T [x:T](B x)) rem1).  (* Elimination of existential quantifier *)
```

3.3 Virtual Witnesses

We want to automate all the work of writing that we want to prove:

$$\forall x : T\ ((A\ x) \rightarrow (\exists y : T\ (B\ y)))$$

declaring a local variable y and an hypothesis y_prop and applying the rule $\exists$-elim. This gives the proof:

```
Theorem th.
Statement (ex T [x:T] (B x)).
   Call y from u.
   Remark rem2:(B y) Proof (v y y_prop).
Proof (ex_intro T [x:T](B x) y rem2).
```

The special variable y is called a *virtual witness*. A virtual witness and its characteristic property may be global or local to the section where the *Call* is written. For instance here y and y_prop are local to the theorem *th*.

When we have a proof u of an existential proposition:

```
Theorem u (ex T [x:T](P x)) Proof <proof>.
```

we can write the sentence:

`Call y from u.`

which implicitly defines two new variables, $y : T$ and $y\_prop : (P\ y)$, and then prove the theorem:

`Theorem th Q Proof <proof>.`

The symbols y and y_prop may be free in the proof of th but not in its statement. When y is out of scope, the statement of the theorem th is unchanged but its proof is replaced by another one in which y and y_prop are not free any more.

3.4 Equivalence between Virtual Witnesses and ∃-elim

The propositions provable in a system without virtual witnesses and with ∃-elim are the same as the ones provable in a system with virtual witnesses and without ∃-elim[3].

- Translation of a proof written with virtual witnesses in natural deduction:

  If a proposition Q has a proof p such that y and y_prop may be free in p but not in Q and u is a proof of $\exists x : T\ (P\ x)$, then let $v = [y : T][y\_prop : (P\ y)]p : \forall y : T\ ((P\ y) \rightarrow Q)$, the proposition Q can be proved with the rule ∃-elim for u and v, the proof is $(u\ Q\ v)$.

- Translation of a proof written with ∃-elim in a system with virtual witnesses:

  If we have a proof u of $\exists x : T(P\ x)$ and a proof v of $\forall x : T((P\ x) \rightarrow Q)$, then we define a virtual witness y from u and $(v\ y\ y\_prop)$ is a proof in Vernacular of Q.

3.5 Proof by cases

In the same way the rule ∨-elim could be replaced by a mechanism of proofs by cases. In this mechanism an axiom or a theorem $A \vee B$ provides two hypotheses $h1 : A$ and $h2 : B$. We have to give two proofs of the same proposition C using first the first hypothesis and then the second. When the two proofs are given a new proof of C in which the symbols $h1$ and $h2$ are not free any more is built. The syntax for this mechanism is not as nice as the one for virtual witnesses because we need to give two proofs instead of one and is not definitive yet.

In a future version of the Vernacular, we will study a syntax for the eliminations over and inductive type [8] which might subsume the virtual witnesses and proof by cases mechanisms.

Conclusion

In this note we have seen how the usual sentences written in mathematics (axioms, theorems, definitions, hypothesis, definition of objects which existence has been proved, section headings, etc.) could be formalized in an Elementary Vernacular. In this Vernacular some natural deduction rules (→-intro, ∀-intro, ∃-elim, ∨-elim) are replaced by intuitive operations.

The sections mechanism is implemented in the distributed version 4.10 of the Calculus of Constructions, the virtual witnesses mechanism is implemented in a prototype version, the proof by cases mechanism is still under progress.

[3] Since ∃ is defined in the formalism itself as an abbreviation, a way to remove the ∃-elim rule is to forbid the application of u to any term if u is of type ∃...

The main difficulty in the use of this Vernacular is to express the proof-terms, these terms can be avoided if we cut the proof in very small pieces (u, (u v) or (u t)) but this leads to proofs with too many lemmas, for instance we need five lemmas to use a transitivity property of a relation and we woud like this operation to be atomic.

Another drawback of this Vernacular is that since some logical symbols (conjunction, disjunction, existential quantifier, contradiction and negation) are defined inside the formalism as abbreviations we need several lemmas to use a rule like $\wedge$-elim and we want also this operation to be atomic.

This Vernacular can be improved by the use of proof synthesis in a language in which these operation are atomic. Such a Vernacular is described in [10].

References

[1] N.G. de Bruijn. "The Mathematical Vernacular, A Language For Mathematics With Typed Sets." Proceedings of the Workshop on Programming Logic, Marstrand, Sweden, 1987.

[2] N.G. de Bruijn. "The Mathematical Vernacular: Examples." Unpublished manuscript.

[3] Th. Coquand. "Une Théorie des Constructions." Thèse de troisième cycle, Université Paris VII, 1985.

[4] Th. Coquand. "An analysis of Girard's paradox." First IEEE Symposium on Logic in Computer Science, Boston, June 1986, 227–236.

[5] Th. Coquand. "Metamathematical investigations of a Calculus of Constructions." To appear, proceedings of the Talin Conference, 1990. Also in [12].

[6] Th. Coquand, G. Huet. "Constructions: A Higher Order Proof System for Mechanizing Mathematics." EUROCAL85, Linz, Springer-Verlag Lecture Notes in Computer Science n°203, 1985.

[7] T. Coquand, G. Huet. "The Calculus of Constructions." Information and Computation, Volume 76, 1988.

[8] T. Coquand, Ch. Paulin-Mohring. "Inductively defined types." "Proceedings of the Workshop on Programming Logic", Göteborg University and Chalmers University of Technology, Båstad, May 89.

[9] G. Dowek. "A Vernacular Syllabus." in [12]

[10] G. Dowek. "A Proof Synthesis Algorithm for a Mathematical Vernacular" To appear in "Proceedings of the First Workshop on Logical Frameworks", Sophia-Antipolis, France, 1990.

[11] J.Y. Girard. "Types and Proofs." translated and with appendices by P. Taylor and Y. Lafont. Cambrige University Press, 1989.

[12] G. Huet Ed. "The Calculus of Constructions, Documentation and user's guide." INRIA, Rapport Technique n°110, August 1989.

[13] G. Huet. "The Constructive Engine." in "A Perspective in Theoretical Computer Science, Commemorative Volume for Gift Siromoney." ed. R. Narasimhan, World Scientific Publishing, 89. also in [12].

[14] G. Huet. "A Uniform Approach to Type Theory." Rapport de recherche INRIA n°795 , Février 88, also in "Logical Foundation of Functional Programming." Addison-Wesley, 1990.

[15] Ch. Paulin-Mohring. "Extraction de programmes dans le Calcul des Constructions." Thèse, Université Paris VII, 1989.

[16] Ch. Paulin-Mohring. "Extracting Fω programs from proofs in the Calculus of Construction." Proceedings of POPL 1989.

[17] D. Simon. "Checking Natural Language Proofs." 9th International Conference on Automated Deduction. Argonne, Illinois, USA, May 1988, Lecture Notes in Computer Science n°310

[18] R.S. Strichartz. "An International Language for Mathematics." The Mathematical Inteligencer, Vol 11, n°1, 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399