Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 140 (2005) 55–66
www.elsevier.com/locate/entcs

Validation of Proofs Using PhoX

Patrick Thévenon[1,2]

LAMA
Université de Savoie
Campus Scientifique - 73376 Le Bourget-du-Lac Cedex, FRANCE

Abstract

In this paper we present the DemoNat project, its purposes and the ideas developed so far. DemoNat is a French project whose aim is to make a program able to analyze and validate proofs made in a natural language. It will be used by students in order to improve the way they understand and make mathematical proofs.

Keywords: Proof assistant, linguistics, resolution method

1 Introduction

As there are two parts in the project (the analysis and the validation), three laboratories of linguistics and/or mathematics are working on it:

- Lattice / TaLaNa (Paris 7)
- Calligramme (Nancy)
- LAMA (Chambéry)

The program, not yet implemented, will be based on the proof assistant PhoX which is developed by Christophe Raffalli at the 'Université de Savoie'. This program is already used by students but since it requires the knowledge of commands, it is not so easy to use it.

[1] Thanks to René David for his attentive reading and corrections. Thanks to Christophe Raffalli for his suggestions of presentation.
[2] Email: patrick.thevenon@univ-savoie.fr

doi:10.1016/j.entcs.2005.06.024

With the DemoNat project students will be able to write their proof in a natural language (say French) and the machine will have to analyze it in term of linguistic and to validate it in term of logic. See below (Fig 1) the diagram of the system.

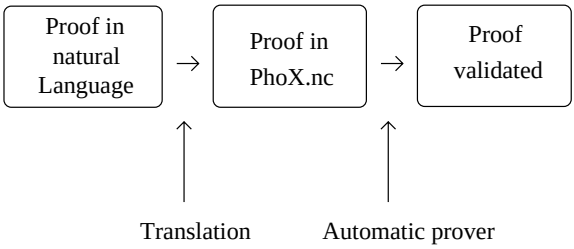


Fig. 1. Diagram of the system

Let us explain it: the student first writes a proof of a theorem in a natural language. The program then translates it into a proof made in an intermediate language called PhoX.nc. It includes the standard commands of PhoX and some others that we will call new_commands.

Standard commands are the usually defined commands in proof assistants: **intro**, **left**, **elim**, **apply**, **prove**, **trivial**, ... All except **trivial** correspond to combinations of proof rules and so are validated for syntactic reasons. New_commands are different because the automatic tactic must prove a formula before validating them.

In section 2 we introduce the new_commands and explain the purpose of this language. In section 3 we give a simple example of a proof that could be made by a student and we explain how the program will have to analyze it. The section 4 is devoted to the validation, the mathematical part of the project, which needs a theorem prover dedicated to this project.

2 New_command: a meta-rule

Before giving the grammar of the new_commands we expose their general purpose. A step of the proof is a list of sequents $\mathcal{H} \vdash G$ where $\mathcal{H}$ are lists of hypotheses and G are goals to prove. At each step we consider one of these sequents.

When a new_command is given it generates a list of sequents $(\mathcal{H}, \mathcal{H}_i \vdash G_i)$. The automatic tactic must then be able to show that the current sequent can be deduced from the sequents generated. Intuitively the theorem prover must prove the following sequent:

$$\mathcal{H} \vdash (\mathcal{H}_1 \to G_1) \to \ldots \to (\mathcal{H}_s \to G_s) \to G$$

which will justify the meta-rule:

$$\frac{\mathcal{H}, \mathcal{H}_1 \vdash G_1 \quad \dots \quad \mathcal{H}, \mathcal{H}_s \vdash G_s}{\mathcal{H} \vdash G}$$

In the next state of the proof $\mathcal{H} \vdash G$ is deleted from the list of sequents and the sequents $\mathcal{H}, \mathcal{H}_i \vdash G_i$ are added.

As the definition of new_commands is not yet fixed, we give here a very general and partial (but an important part of the) grammar for the commands in PhoX.nc. An important thing to mention is that these commands are designed to be close to a natural language but with a formal grammar. Thus by this principle some of the standard commands are kept:

let .. cmd	(naming)
| **assume** .. { **and** ..} cmd	
| **deduce** .. { **and** ..} cmd	
| **by** .. { **with** ..} cmd	(hints)
| **show** ..	(replace goal)
| **prove** ..	(cut rule)
| **trivial**	
| $\emptyset$	
| cmd **then** cmd	(cases)
| **begin** cmd **end**	(brackets)

Some of the commands are obvious but some of them need to be explained.

- **let** allows to introduce new names, that is to introduce new constants (or variables). For instance if you have to prove that $\forall x \in \mathbb{R} \quad P(x)$, you can write **let** $t \in \mathbb{R}$ **show** $P(t)$. Then the new name can be used in the current branch of the proof.
- **assume** allows to give more information on what the introduced constant must verify. For instance if $\forall x \quad M(x) \to N(x)$ is the goal we could write **let** t **assume** $M(t)$ **show** $N(t)$. More generally it is used to add hypotheses.
- **deduce** is just for deducing things from the current hypotheses which are added to the hypotheses. For instance if $A \to B$ and A are hypotheses we can write **deduce** B.
- **by** and **with** are really important commands for the prover as they are used to give some clues on how the proof of the 'meta-rule' can be done. After **by** and **with** can be given either an hypothesis (fact) or a variable name. These commands have no influence on the property associated to the new_command, they only help the prover. Examples are given in the next sections.

- **show** is quite obvious, its purpose is to change the current goal. It can be read as 'it suffices to prove that'.
- **prove** is very different from **show** and from the others given as it is standard and so does not need to be validated by an automatic prover: it is simply the use of a cut rule. Here the user wants to introduce a lemma that he will prove next.
- **then** adds sub-goals in the meta-rule. It allows to prove by cases: for example, if we have $A \vee B$ as an hypothesis, we can write **assume** A **then assume** B. This command is also used when the goal is $A \wedge B$: we can then write **show** A **then show** B.
- **begin** ... **end** is just to put new_commands into parentheses. This is necessary because it is possible to put commands into others.

3 The translation

We now introduce an example of a proof for a proposition that can be made by a student with the system. But first let us introduce some constants and definitions.

Let X and Y be two topological spaces. Let $f : X \longrightarrow Y$. We can define two notions of continuity:

• f continuous = the inverse image by f of an open set is an open set.

• f continuous at x = the inverse image by f of a neighbourhood of $f(x)$ is a neighbourhood of x. For PhoX the notation will be "continuous_at".

Proposition 3.1 *f continuous $\rightarrow \forall x \in X$ f continuous at x.*

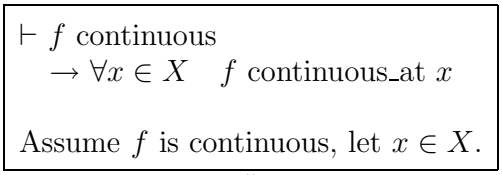
Proof. Assume f is continuous, let $x \in X$. Let V be a neighbourhood of $f(x)$, we must prove that its inverse image is a neighbourhood of x. By definition of a neighbourhood let O be an open set included in V and containing $f(x)$. As f is continuous, $f^{-1}(O)$ is open and $x \in f^{-1}(O)$. As $f^{-1}(O) \subset f^{-1}(V)$, the proof is finished. □

As you can see the proof is really easy. Now let us study in detail how the translation from the natural language to PhoX.nc could be made. We must warn about the fact that this translation is only one possible and that there may be many other translations. The aim is just to point out some of the difficulties that may occur while translating proofs into PhoX.nc. The translation here will be done step by step, that is one sentence after the other. At each step we give the current hypotheses and goal together with the sentence written by the student and from them will follow the translation.

Note that the syntax we use in this article for the formulae is not exactly

the syntax found in PhoX. For example, we will always write "$x \in X$" when in PhoX it could be written "$X\ x$". Another examples are the use of "x_n" instead of "$x\ n$" and "$f(x)$" instead of "$f\ x$". The aim is just to have usual notations in order to be more readable by anyone.

• First the user adds assumptions and so changes the goal.

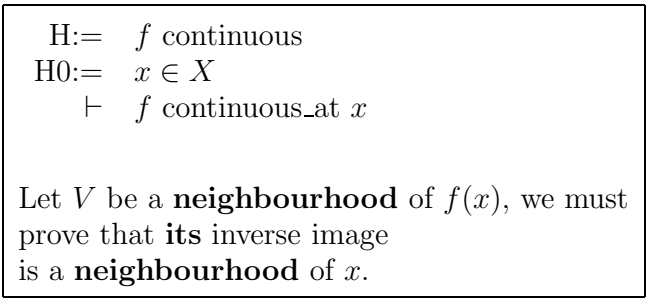


⇓

assume f continuous let $x \in X$
show f continuous_at x.

It is necessary, in the translation, to give the new goal (if it changes) even if the user does not give it explicitly, in order to obtain a valid meta-rule. It is the first difficulty encountered but this one is not very hard as the translator knows the current state of the proof and so can infer that we are dealing with a goal of the form $M \to N$ and that M is assumed. Thus what is to be proved is N.

• Now the user shows that he knows his definition of local continuity.



⇓

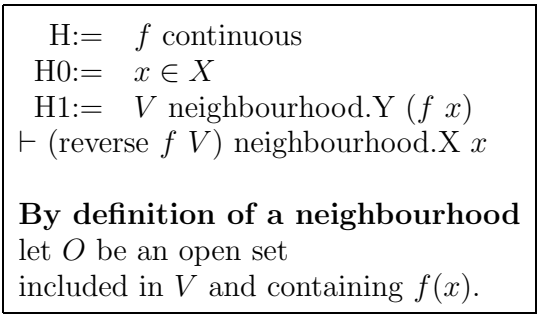
let V assume V **neighbourhood.Y** $f(x)$
show (**reverse** f **V**) **neighbourhood.X** x.

Here there are two difficulties. The first one is that the student knows implicitly that there are two kinds of neighbourhood but he does not make this difference explicitly in his sentence. So the translator has to be able to deal with that: find the good notions of neighbourhood for each word. Here it is one more time helped by reading the definition of the goal. The sentence of the student is no more than a reading of a definition.

The second difficulty comes from the use of the word 'its'. To what does it refer ? Such kind of words, called anaphoras, can often appear in proofs

and the translator will have to deal with them. Here it clearly refers to the neighbourhood V which is the last constant defined by the student. Moreover it is not given by which function we take the inverse image of 'it'. Here the context implies that it can only be the function f.

• Now the student uses an hypothesis:

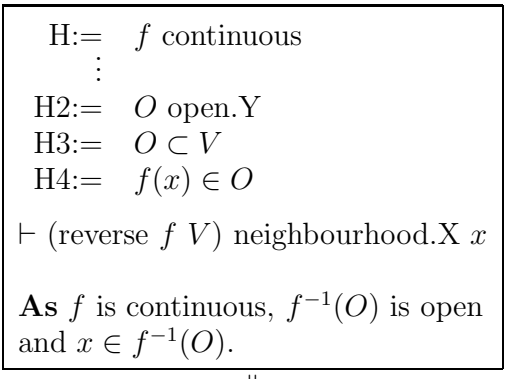


⇓

by H1 let O
assume O open.Y and $O \subset V$ and $f(x) \in O$.

Most of the hypotheses will never be explicitly named by the user (even if it is possible to do this in the system) because usually they are not. Hypotheses are only named in difficult proofs or when they can be used many times, etc... Here the student just wrote "by definition of a neighbourhood" so he could either work on the goal or on the hypothesis H1 as both 'speak about' neighbourhood. In fact if the translator does not manage to know that we are working on H1, it could try the different possibilities in order to find the good one which is the one that will be validated by the automatic prover (otherwise the user can come back to the last goal and changes its sentence). But it could here find that the user talks about the hypothesis H1 as it talks about $f(x)$, which only appear in this hypothesis.

• Here is another use of assumption:



⇓

by f continuous deduce
(reverse f O) open.X and $x \in$(reverse f O).

We just here note that the translator may use directly the hypothesis instead of its name

• Here is the end of the proof:

$$\begin{array}{|l|}\hline \\ \quad\vdots \\ \text{H3:=}\quad O \subset V \\ \quad\vdots \\ \text{H5:=}\quad \text{(reverse } f\ O\text{) open.X} \\ \qquad\quad \wedge\ x \in \text{(reverse } f\ O\text{)} \\ \\ \vdash \text{(reverse } f\ V\text{) neighbourhood.X } x \\ \\ \textbf{As } f^{-1}(O) \subset f^{-1}(V), \text{ the proof is finished.} \\ \\ \hline \end{array}$$

$$\Downarrow$$

deduce (reverse f O) $\subset$ (reverse f V) trivial.

The difficulty here is that the word 'as', unlike in the previous step, does not refer to an hypothesis already proven but to a fact that seems obvious to the user. So the new_command here must be **deduce** and not **by**. Finally, when the proof is finished (explicitly said as here or not) the last command must be **trivial**, that is the result must be proven in the last state of the proof by the automatic prover.

4 The validation of meta-rules

In this part we explain how the meta-rules coming from the new_command could be validated by an automatic prover. As the current automatic tactic of PhoX is not powerful enough for the new_commands, we need another one, dedicated to the problem. The idea is in fact to implement an inverse resolution method in a lazy way. The purpose of this paper is not to define it, we will just give some examples and hope that they are clear enough. We assume anyway that the reader knows what is the resolution method. Otherwise, see some of the references given, for instance [2] and [5].

We forget the previous example and take another ones, better for purposes of readability. In fact we will give two examples. For each we give the current step of the proof then the sequent given to the automatic prover, to finish with the proof.

A simple example

• Consider the following formula:

$$\forall i \in I \quad A \cap h(i) = f(i) \Rightarrow A \cap \bigcap_{i \in I} h(i) = \bigcap_{i \in I} f(i)$$

It is obvious that it is an implication $P \rightarrow Q$. Thus to prove it we take P as an hypothesis and Q as the goal. We obtain the following sequent to prove which is solved by the given new_command:

$$\begin{array}{|l} \text{E0:=} \quad \forall i \in I \quad A \cap h\ i = f\ i \\ \quad \vdash \quad A \cap (\text{Inter } h\ I) \subset (\text{Inter } f\ I) \\ \\ \text{let } x \in A \cap (\text{Inter } h\ I) \text{ show } x \in (\text{Inter } f\ I). \end{array}$$

Where Inter $f\ I$ is the notation for the intersection of the $f(i)$ for $i \in I$.

• To validate the new_command, that is to validate the meta-rule associated to this situation, the automatic prover has to be able to prove the sequent below (which should never appear to the user):

$$\begin{array}{|l} \text{E0:=} \quad \forall i \in I \quad A \cap h\ i = f\ i \\ \quad \vdash \quad \forall x\ [x \in A \cap (\text{Inter } h\ I) \rightarrow x \in (\text{Inter } f\ I)] \\ \qquad \rightarrow \\ \qquad A \cap (\text{Inter } h\ I) \subset (\text{Inter } f\ I) \end{array}$$

Simplifying the formulae by using generic names and thanks to the definition of a subset ($X \subset Y := \forall x \quad x \in X \rightarrow x \in Y$) it is equivalent to the following sequent:

$$\begin{array}{|l} \text{E0} \\ \quad \vdash \quad \text{K} \rightarrow \text{K} \end{array}$$

• This is very easy to prove by any prover. We see here that in some cases the formula given to the automatic prover is just a propositional tautology. Note however that here K represents a formula which is not propositional. Thus the automatic tactic must not see in detail what is the formula. It decomposes only the formula $K \rightarrow K$ and ends the proof when it unifies K and K. Formulae can be seen as 'black boxes' until a decomposition is needed.

Note that after this proof, the new_command is validated, and that the current sequent (printed for the user) becomes:

$$\begin{array}{ll} \text{E0:=} & \forall i \in I \quad A \cap h\ i = f\ i \\ \text{E1:=} & x \in A \cap (\text{Inter}\ h\ I) \\ \vdash & x \in (\text{Inter}\ f\ I) \end{array}$$

Two things have been done between the two states of the proof: an introduction of a universal quantifier and an introduction of an arrow. An important thing to notice is that the automatic tactic didn't have to make these introductions, as they are implied by the syntax of the new_command. The automatic tactic just has to validate the meta-rule. And in this case it is possible by proving a propositionnal formula.

A more difficult one

• Let F be a closed set and $(x_n)_n \subset F$. While proving that

$$(x_n)_n \text{ converging to } a \to a \in F$$

we take a sequence $(x_n)_n$ of F converging to a. As F is closed it suffices to take a neighbourhood V of a and to show that $F \cap V$ is not empty. So we get the following sequent (the dots replace some hypotheses like F closed that we will not need) and new_command:

$$\begin{array}{ll} & \vdots \\ \text{G:=} & \forall V \quad (V \text{ neighbourhood } a \to \exists n \in \mathbb{N} \quad x_n \in V) \\ \text{H:=} & V \text{ neighbourhood } a \\ \vdash & \exists y \in F \cap V \\ & \\ \text{by [G] with [H] let } n \in \mathbb{N} & \quad \text{assume } x_n \in V. \end{array}$$

• To this is associated the following sequent given to the automatic prover (the hypotheses are the same, they are our environment):

$$\begin{array}{ll} & \vdots \\ \text{G:=} & \forall V \quad (V \text{ neighbourhood } a \to \exists n \in \mathbb{N} \quad x_n \in V) \\ \text{H:=} & V \text{ neighbourhood } a \\ \vdash & \forall n \in \mathbb{N}\ (x_n \in V \to \exists y \in F \cap V) \\ & \to \exists y \in F \cap V \end{array}$$

Remember that the commands **by** and **with** have no influence on the formula, they just give some information to the automatic prover. And here the information is very good, as only the two hypotheses named will be used to prove the property.

We can again simplify by giving names to some sub-formulae and deleting the useless hypotheses:

$$\begin{array}{ll} \text{G:=} & \forall\ V\ (H[V] \rightarrow \exists n\ (n \in \mathbb{N} \wedge x_n \in V\ )) \\ \text{H:=} & H[V] \\ \vdash & \forall n\ (n \in \mathbb{N} \rightarrow x_n \in V \rightarrow K) \rightarrow K \end{array}$$

• And now how can the automatic tactic prove this sequent ? As we have already said we will use a resolution method. So we start with a first set of clauses. We add one clause for each hypothesis and we negate the goal to obtain the last one:

$$\begin{array}{l} C_1{:=}\{\forall V\ (H[V] \rightarrow \exists n\ (n \in \mathbb{N} \wedge x_n \in V))\} \\ C_2{:=}\{H[V]\} \\ G{:=}\{\neg\ [\forall n\ (n \in \mathbb{N} \rightarrow x_n \in V \rightarrow K) \rightarrow K]\} \end{array}$$

We follow now what should be the better way to find a contradiction. First we decompose the goal G ($\neg(A \rightarrow B)$ is seen as $A \wedge \neg B$) to obtain two new clauses C_3 and G':

$$\begin{array}{l} C_1{:=}\{\forall V\ (H[V] \rightarrow \exists n\ (n \in \mathbb{N} \wedge x_n \in V))\} \\ C_2{:=}\{H[V]\} \\ C_3{:=}\{\forall n\ (n \in \mathbb{N} \rightarrow x_n \in V \rightarrow K)\} \\ G'{:=}\{\neg K\} \end{array}$$

Then we decompose the clause C_1, obtaining a variable of unification $V?$, giving C'_1 and the clause C_3, obtaining a variable of unification $n?$ and we rewrite the clause obtained C'_3:

$$\begin{array}{l} C'_1{:=}\{(H[V?] \rightarrow \exists n\ (n \in \mathbb{N} \wedge V?\ (x\ n)))\} \\ C_2{:=}\{H[V]\} \\ C'_3{:=}\{\neg n? \in \mathbb{N} \vee \neg x_{n?} \in V \vee K)\} \\ G'{:=}\{\neg K\} \end{array}$$

Now considering that $A \rightarrow B$ is equivalent to $\neg A \vee B$, we can apply the resolution rule between C'_1 and C_2 to obtain C_4:

$$\begin{array}{l} C_4{:=}\{\exists n\ (n \in \mathbb{N} \wedge x_n \in V)\} \\ C'_3{:=}\{\neg n? \in \mathbb{N} \vee \neg x_{n?} \in V \vee K)\} \\ G'{:=}\{\neg K\} \end{array}$$

Now we decompose the clause C_4, obtaining a constant of unification n in a clause C'_4:

$$\begin{array}{l} C'_4{:=}\{n \in \mathbb{N} \wedge x_n \in V\} \\ C'_3{:=}\{\neg n? \in \mathbb{N} \vee \neg x_{n?} \in V \vee K)\} \\ G'{:=}\{\neg K\} \end{array}$$

Decomposing the clause C'_4 we get two new clauses C_5 and C_6:

$$\begin{array}{|l}\hline C_5{:=}\{n \in \mathbb{N}\}\\ C_6{:=}\{x_n \in V\}\\ C'_3{:=}\{\neg n? \in \mathbb{N} \vee \neg x_{n?} \in V \vee K)\}\\ G'{:=}\{\neg K\}\\ \hline\end{array}$$

We see now that we can apply some resolution rules between C_5, C_6, G' and C'_3 to obtain the empty clause $\Box$, which ends the proof.

What can we conclude about what we have just seen ? That our way to apply a resolution strategy is not standard. In fact it is in some sense an inverse resolution strategy: the formulae are decomposed only when needed, we are not dealing with conjunctive normal forms of atomic formulae as it is done in the classical resolution method.

We can often see that most of the sub-formulae are useless for the automatic tactic. For example it does not need to know what is the formula K when it proves $K \rightarrow K$. So formulae are kept as they are and decomposed only when needed. This is why we call our method an 'inverse resolution in a lazy way'. So The commands **by** and **with** are very useful to indicate which hypotheses should be more quickly decomposed than the others.

5 Conclusion

What we have seen here is an overview of the DemoNat project which is not yet implemented (except PhoX and PhoX.nc). Some of the difficulties that can appear in the linguistic part of the project have been pointed out. In the basic example given they seem not to be strong: we have only done a translation sentence by sentence.

In fact there are two ways of thinking the system. The first one is to validate a whole proof and so to read the complete text of the proof. In this case the system has a great freedom on the parts translated into PhoX.nc. The second one is to build a proof, i.e. the user interacts with the system. He writes a proof and when he needs to know what is the current state of the proof he tells it to the analyzer. Then the system analyses the part asked by the user.

In both cases the context must be kept by the system after each step. This is needed to be able to solve the problems of anaphoras. Moreover the translation must be able to give the "hints" to the automatic tactic in order to help it in proving the formulae implied by the new_command. The user may not give them each time and so a study of the context may help to find the important hypotheses. This part is really an essential one and makes a

difference with usual theorem provers.

As we have seen, a good way to handle with the formulae implied by the commands of PhoX.nc is to implement an inverse resolution in a lazy way. Indeed the formulae given to the automatic prover seem to be particular: most of them are propositional or need the use of few quantifiers. We could think about a prover becoming stronger when the user is a more advanced student. This means that the system may reject steps of the proof if it sees that they are too difficult for a beginner for example.

References

[1] C.-L. Chang and R. C.-T. Lee, *Symbolic Logic and mechanical Theorem proving*, Academic Press 1973.

[2] R. David, K. Nour and C. Raffalli, *Introduction la logique*, Dunod 2001.

[3] D. A. Duffy, *Principles of automated theorem proving*, Wiley 1991.

[4] M. Fitting, *First-order Logic and automated Theorem proving*, Springer-Verlag 1990.

[5] A. Leitsch, *The Resolution Calculus*, Springer-Verlag 1997.

[6] T. Tammet, *Resolution Theorem Prover for Intuitionistic Logic*, in CADE-13, LNCS 1104, Springer-Verlag 1996.

[7] T. Tammet, *Resolution, inverse method and the sequent calculus*, University of Göteborg and Chalmers University of Technology.