Panda: a Proof Assistant in Natural Deduction for All. A Gentzen style proof assistant for undergraduate students

Olivier Gasquet, François Schwarzentruber, and Martin Strecker

IRIT (Institut de Recherche en Informatique de Toulouse)
Université de Toulouse
Toulouse, France

Abstract. We present a proof assistant in Natural Deduction for undergraduate students. The system is interactive: you can combine, delete, modify proofs with a easy-to-use graphical interface. We discuss the pedagogical benefit of this tool.

Key words: Teaching logic in computer science; Gentzen style natural deduction; interactive provers

1 Introduction

We, authors of this paper, have been teaching logic to second-year undergraduate students in computer science for several years. We classically present syntax, model theory and proof theory (natural deduction) for both propositional logic and predicate logic. Natural deduction seems to reveal some difficulties among students: rigour and abstraction.

Curiously, while being able of writing rather *rigourously* a simple C program (at least to the point that the compiler is happy with the syntax of their program), often they still cannot rigourously write down a mathematical proof of the kind needed in graph theory, formal logic, abstract data types,... We believe that this may partly be attributed to their mathematical backgrounds: at least in France, mathematics, up to high school, is mainly taught as a science of *numbers and quantities* (relations between integers in arithmetics, between reals or complex numbers in analysis, between matrices of numbers in linear algebra, but always numbers), and not as a science of *structures*, which is a crucial point of view in logic, in computer science and other areas of discrete mathematics. This latter problem seems to us linked to a lack of abstraction capabilities.

Many courses of logic include an introduction to some proof system often based on some variant of natural deduction among the following three main approaches (according to the thorough review of natural deduction systems of

[5]): Fitch's diagrams[1], Suppes' annotated proofs[2] and Gentzen's trees. Any of them of course could help students in gaining rigour in reasoning by the *use* of formal systems of deduction. So why should one choose Gentzen's trees as we did since Pelletier, in the same paper, says:

> "The Gentzen tree method did not get used much in elementary logic books with the exception of Curry [...] van Dalen [...] and Goodstein [...] and Bostock [...] In any case, this method of representing natural deduction proofs is not at all common any more."

We want here to argue in favor of Gentzen's tree. We believe that one reason of their less frequent use w.r.t. the other systems is purely technical: they are more difficult to typeset (and hence it is more difficult to implement a friendly interface for them) when compared with Fitch's and Suppes' systems. But it seems to us that Gentzen's style proofs are more readable because the tree-like structure of proofs (which is linked to their inductive construction) is more evident.

Another drawback of other systems is that on the one hand Suppes' style proofs tend to be very long and the structure of the proof tends to be lost, or at least invisible. On the other hand, Fitch's boxes for big proofs involving many subproofs tend to be inconvenient, a nice interface could be difficult to design due to the need of resizing the boxes dynamically, also, the tree-structure tends to disappear with the size of the proof. On the contrary, Gentzen's trees really do justice to the natural tree-structure of proofs, and to us, this is an underestimated feature: it is the best formalism[3] when one wants to point out the fact that *proofs are objects,* and that as such they may be combined and manipulated. It is in this sense that Gentzen's trees train for abstraction: they not only accustom students to the use of logic for proving, but also to the abstract manipulation of proofs as mathematical objects. At least for future graduate computer scientists, this is of high importance. But of course, all these theoretical considerations must be taken into account when it comes to propose an implementation. In our software Panda, students may as well build pieces of proofs, but also manipulate them by drag-and-drop actions, so they can see how easy it is to build a proof of say $A \wedge B$ once they are provided with a proof of A and a proof of B.

We initially looked for tools that could help in teaching natural deduction like *proof assistants*, or at least proof checkers, that would meet our requirements:

- Easy to install and to use;
- Allows one to check a given proof;
- Allows one to build a proof either with some help, or without help;
- Allows both backward and forward reasoning;
- Allows to easily compose proofs from subproofs;
- Uses Gentzen's proof trees style.

[1] That goes back to the "graphical method" of Jaśkowski (1934).

[2] Pelletier argues that the annotated method of the latter is different from that of Suppes, but here we will ignore this small difference.

[3] Of course, here we do not mention Sequents systems, but they are not the most natural formalism.

And we came to develop our own proof assistant PANDA. Nevertheless, we briefly compare PANDA to some existing tools in Section 4.

2 Constructing proofs in Natural Deduction

Let us first give an example of a typical proof using PANDA and then describe the great degree of freedom to construct and compose proofs.

2.1 Starting a proof

In its most basic form, the user first enters a formula to be proved by means of the "Add a Formula" button. A few exercises in various degrees of complexity ("Examples level 1 ... 5") are also available in the menu bar. After entering, say, the initial goal $(A \wedge B) \longrightarrow (B \vee A)$, the formula is displayed on the main canvas.

During proof construction, there is always a currently active formula, highlighted by a red frame, to which one of the rules of the calculus of Natural Deduction (or derived rules) can be applied. Applicable rules are displayed in the task bar on the left. The set of rules changes, depending on the syntactic form of the active formula. For example, the rule "And introduction" $(I\wedge)$ is not displayed for the initial goal, because it would not be applicable.

2.2 Forward- and backward reasoning

Rules can be applied in forward- and backward-chaining mode. In the taskbar, the formula matched against the currently active formula is indicated in boldface font. Let us proceed in pure backwards style and apply the rule $(I \longrightarrow)$. The subgoal $B \vee A$ is displayed directly above the original formula. Shaded in light green, there is also the hypothesis that has just been introduced and that can be used later on. By clicking on the appropriate $(I\vee)$ rule, we now have reduced the current goal to B. Entering the formula (in this case A, see Figure 1) produces the available hypothesis $A \wedge B$, which automatically concludes the proof.

Proving by backward chaining is implemented in some existing proof editors (see § 4). This style of proof can be cumbersome, due to the missing subformula property of Natural Deduction. Therefore, PANDA allows arbitrary mixture of forward and backward reasoning. Partial proof trees can be constructed, moved across the canvas and easily composed with some mouse gestures. Of course, there is no guarantee that these partial proofs can be eventually extended to a complete proof. However, PANDA keeps track of hypotheses introduced in branches of the proof tree and verifies variable conditions during proof composition, which ensures that finished proofs are sound.

In order to explore some of these possibilities, let us return to the situation in the above proof directly after application of the $(I \longrightarrow)$ rule. This is easily done: PANDA allows subtrees or entire regions of the canvas to be deleted, and there is a chronological undo operation that reverts the proof to a previous proof state. Activating the hypothesis $A \wedge B$ (as depicted in Figure 2 on the left) now allows to proceed in several ways:

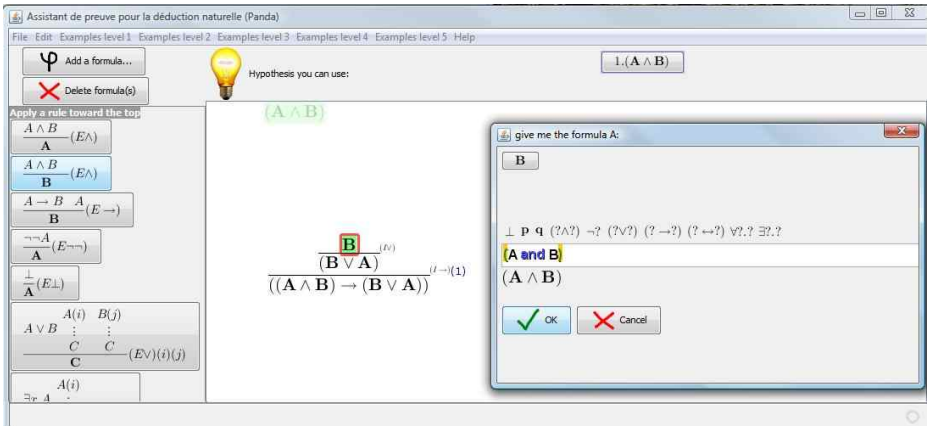

Fig. 1. Simple backward proof in Panda

- We may select one of the elimination rules $(E\wedge)$ in the task bar on the left.
- We may accept one of the formula decompositions that are already suggested by Panda, for example by clicking on the light green A below $A \wedge B$. Panda automatically chooses the appropriate rule and applies it.
- We may introduce the desired conclusion of the rule (for example A), using the formula editor, and then connect hypothesis and conclusion of the rule, as described further below.

2.3 Proof tree manipulation

In any case, the result is a partial proof tree whose contours are in light red as in the right part of Figure 2. We join the two partial prooftrees by moving the upper tree close to a leaf of the lower proof tree. By using some heuristics, Panda will recognize that these trees are indeed compatible, by application of an $(I\vee)$ rule (this is indicated by the blue line drawn between the trees). By releasing the upper tree at this position, the two trees indeed "snap" together, and the proof is finished.

Rules with several preconditions, such as $(E\vee)$ or $(E \longrightarrow)$, can be applied in forward chaining style with great ease: It is sufficient to align the conclusions of the subtrees on approximately the same height, and then draw a horizontal line below them, such as in Figure 3. Panda will recognize the appropriate rule pattern and apply the corresponding rule.

2.4 First-order reasoning

At the moment, Panda only treats standard first-order logic[4]. Concerning rules for handling quantifiers, and particularly $\exists$ we choose the so-called Gentzen's

[4] Some extensions (e.g. with equality) are under development.

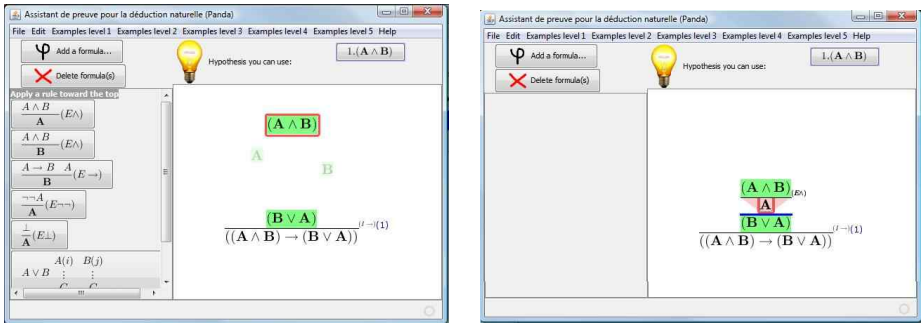

Fig. 2. Forward proofs in PANDA

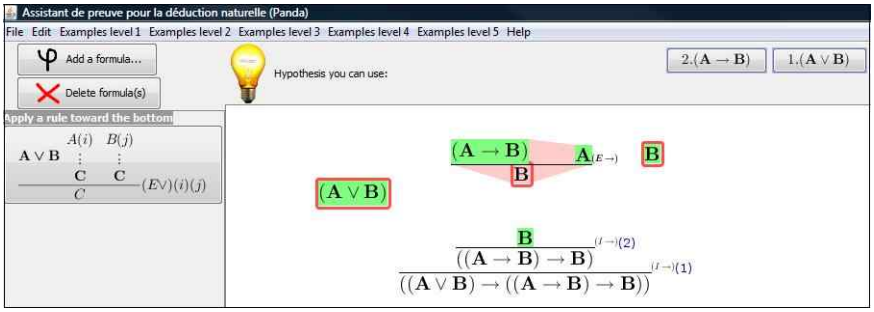

Fig. 3. Applying elimination rules

style vs. Quine's style. The former is based on "Elimination of Existential" ($E\exists$) and the second on "Existential Instanciation" (EI). It seems to us that (EI) is always less natural than ($E\exists$), since either it allows unsound steps in the proofs: from $\exists x\phi(x)$ infer $\phi(a)$, or requires that the a introduced at this step is not a term, but a *parameter*[5]. We believe that ($E\exists$) with its subproof process looks more natural: From $\exists x\phi(x)$ and a subproof $(\phi(x)\ldots\Psi)$ infer Ψ (with restriction on x and on variables of Φ of course). Thus a proof of $\exists xR(x,x) \to \exists x\exists yR(x,y)$ in PANDA looks like in Figure 4.

2.5 Saving and loading

Proofs may be saved in LaTeX format for display in other documents.They may also be saved in our own specific format which is stored as an xml file and may be reloaded for further use.

3 Behind the scenes

The piece of software Panda is an open-source project written in JAVA for two reasons. The first one is because an object-oriented language is really adapted to

[5] Also called arbitrary/quasi/pseudo/ambiguous names

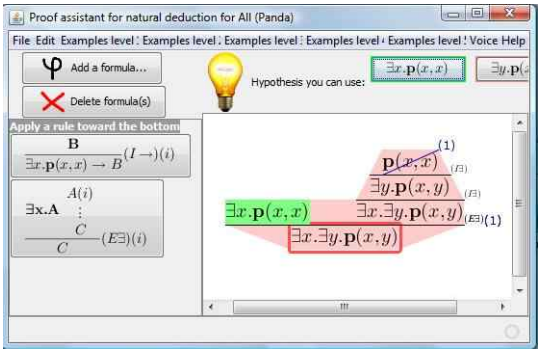

Fig. 4. Applying elimination of ∃

create graphical user interface. The second one is for cross-platform compatibility and in order to be launched from the web without any installation.

3.1 The general graphical user interface

Abstraction has been adopted in order to be able to generalize easily this interface to other proof systems. The class `ProofFormulaNode` represents an abstract node in a proof tree which can have a father and children. Then the class `ProofFormulaNodeNatDet` inherits from `ProofFormulaNode` and implements the rules of Natural Deduction. Formulas are written in a Lisp/Scheme approach: for instance `(a and (b or (not c)))` stands for $(a \wedge (b \vee \neg c))$. The main advantage is that parsing s-expressions is simple since we use the library JScheme[6]. Formulas in a proof node are then translated in LaTeX code and displayed thanks to the library jLatexMath[7] (it does not require a real LaTeX installation).

3.2 Pattern-matching to select rules

The class `InsertionRuleButton` represents an abstract button enabling to implement an application of a rule of natural deduction. It allows to implement a function `testIfRuleApplicable` for testing if the rule is applicable on the current selected nodes. If a rule is not applicable, the button is invisible. It offers also a function `ruleApply` in order to apply the rule. For instance, the class `InsertionRuleEliminationImplyBU` implements the functions `testIfRuleApplicable` and `ruleApply` for rule $(E \longrightarrow)$:

$$A \qquad A \longrightarrow B \qquad \xrightarrow{\texttt{ruleApply}} \qquad \frac{A \qquad A \longrightarrow B}{B}$$

The function `testIfRuleApplicable` tests if the selected nodes match with the left-side of the rule. The function `ruleApply` adds the node B and connect it to nodes A and $A \longrightarrow B$.

[6] `http://jscheme.sourceforge.net/`
[7] `http://forge.scilab.org/index.php/p/jlatexmath/`

3.3 Cancel and redo

The interface `Command` provides the method `execute` to execute an action and `undo` to execute the converse of the action. For instance, the class `CommandAddNode` implements the interface `Command`: `execute` adds a node to the proof and `undo` removes this node. This way of thinking makes programming the cancel/redo mechanisms easy: we just save `Command` objects in a stack.

3.4 Load and save rules

There are several file formats for saving proofs: Isabelle, Coq etc. Here our needs are different: our proofs are trees with graphical positions etc. So already-known file formats are not adapted to save proofs. That is why we created our own XML file format for saving rules in Panda.

4 Related tools

There are many logic tools (provers, proof assistants, truth table editors,...) on the internet, but most of them are either very difficult to install (and not always work), are no more maintained, are not platform independant, are only libraries, etc. Nevertheless there are also suitable tools: Pandora described in [3] and J∀P∃ [2], which are in Fitch style, and Proof Lab described in [6] which uses Suppes' style annotated proofs. They are all three very mature and valuable tools, in an advanced state of development including many advanced topics (like elements of set theory for example). Note that the last version of Proof Lab is not really a proof assistant but rather a tutor, it proposes exercises, but one cannot try to build arbitrary proofs. Their respective style of proof and that of Panda have their respective merits and drawbacks. In general, the hypotheses available locally at a particular point in the proof can be read off more easily from the nesting structure in the box style, whereas the premises contributing to a conclusion are displayed more clearly in the tree style of Panda (the premise-conclusion relation is obtained by numbering nodes in J∀P∃). Besides, the free composition of partial proofs, a distinctive feature of Panda, is barely conceivable (and indeed not supported) in J∀P∃ nor in Proof Lab and Pandora, since it would mean introducing a box in the scope of a surrounding box.

Of course, we should say a word about Barwise and Echtemendy's well-known Hyperproof ([1]), which allows students to prove statements about situations involving color, shape and size of geometrical objects. This software, which is only a part of the OpenProof project, has clearly great educational virtues. But it runs only on MacIntosh, it is not free, and is not designed to be used alone but inside the whole courseware package. Unfortunately, the Hyperproof package is out of print. We believe that despite its high qualities, this software has too many drawbacks that limit its diffusion.

A less known tool is Bonsai [4]. The notable differences with Panda are:

- Bonsai uses a minimalistic logic language, without "or" or "exists"
- Proof construction is bottom-up and does not allow forward reasoning.

– However, it allows different calculi to be selected (such as intuitionistic and classical logic), whereas PANDA has "hard-wired" classical logic.
– In BONSAI, there are facilities for proof search, for generating proof terms (lambda terms) and for proof normalization.

5 Further work

Firstly, we would like to make it possible for users of PANDA to define their own rules, which can capture commonly occurring patterns of reasoning. For this, one would need an abstract language of proof rules.

Secondly, we would like to increase the automation of PANDA. Proof search is rather easy to conceive in a system that is working in backward reasoning style, and where it is necessary to close open leaves in the proof tree. However, it is more difficult to see how proof search could fill in the gaps between partially constructed proof trees that are disposed on the main canvas of PANDA.

There are extensions that might gradually be introduced, for example support for certain theories (equality, arithmetic, geometry) or a framework for reasoning about programs, in the style of a Hoare logic.

However, there are other topics that will most likely not be integrated into PANDA. For example, proof terms and normalization presuppose notions of lambda calculus that our students have not acquired at this point of their studies.

To conclude, we think that the PANDA experience was interesting for us, and helped our students improve their understanding of logic reasoning. We will especially point out that one disabled student who cannot write with a pencil, PANDA was the only reasonable means of interaction. We invite the reader to try out PANDA: an interactive and multilingual [8] version is available from `http://www.irit.fr/panda`

Acknowledgements. We would like to thank Elsy Kaddoum who has implemented a prototype of PANDA during a student internship.

References

1. J. Barwise and J. Etchemendy. *Hyperproof.* CSLI Publications Standford, 1994.
2. Richard Bornat. *Natural Deduction Proof and Disproof in Jape*, March 2004. `http://jape.comlab.ox.ac.uk:8080/jape/`.
3. Krysia Broda, Jiefei Ma, Gabrielle Sinnadurai, and Alexander Summers. Pandora: A reasoning toolbox using natural deduction style. *Logic journal of the IGPL*, 15(4):293–304, 2007.
4. Oliver Deiser. *Bonsai Manual*, April 2004. `http://www.aleph1.info/bonsai/bonsai.html`.
5. Francis Jeffry Pelletier. A brief history of natural deduction. *History and Philosophy of Logic*, 20(1):1–31, 1999.
6. Wilfried Sieg. The AProS project: Strategic thinking & computational logic. *Logic journal of the IGPL*, 15(4):359–368, 2007.

[8] At the moment only English and French are available.