Gentzen-Prawitz Natural Deduction as a Teaching Tool

Jean-François Monin, Cristian Ene, Michaël Périn

Gentzen-Prawitz Natural Deduction as a Teaching Tool

Jean-François Monin, Cristian Ene, and Michaël Périn

Université de Grenoble 1

Abstract. We report a four-years experiment in teaching reasoning to undergraduate students, ranging from weak to gifted, using Gentzen-Prawitz's style natural deduction. We argue that this pedagogical approach is a good alternative to the use of Boolean algebra for teaching reasoning, especially for computer scientists and formal methods practioners.

1 Introduction

Logic is one of the uppermost basic ingredients of formal methods. The most common approach for teaching logic takes its root in the model theoretical view: logical connectors are seen as Boolean functions (truth tables), and then generalized to quantifiers: $\forall$ is like an infinite conjunction, $\exists$ is like an infinite disjunction. Teaching logic along these lines is a well-established tradition. Boolean algebra proved efficient for solving enigmas and, much more seriously, for designing digital circuit or automatizing the resolution of large combinatory problems (e.g. by reduction to SAT). In the area of hardware and of programming, they provide Boolean expressions and play a key role in control structures such as the if and while constructs, not to speak about bit-level programming.

However, the Boolean approach is not so clearly related to *usual* reasoning. The case of implication is especially questionnable. In every day life, as well as in mathematics textbooks, nobody proves an implication $A \Rightarrow B$ by computing a truth table: one assumes A and then proves B under this hypothesis.

It is even argued that logic is essentially about *proofs*, before being about *truth.* First we can observe that in some logics, including temporal logics and modal logics which have many applications in formal methods, the semantics of a proposition is rather more complex than an truth value – typically, it is described by a Kripke semantics. But even in the case of usual logics, logicians following Dummet, Prawitz and Schroeder-Heister worked on a proof-theoretic semantics of logic (see [11] for a recent presentation).

Proofs can be formalized using syntactic objects described by *deduction systems.* Such systems were introduced in the last century in order to study the meta-theory of logic. Four years ago, we decided to experiment the use of a particular deduction system, namely *Gentzen-Prawitz Natural Deduction* (GPND, for short) for teaching purposes at an introductory undergraduate level. The choice of GPND is discussed below in section 2.

The main thesis of this paper is that GPND has a strong pedagogical interest independent from meta-theoretical considerations. First, it provides a much better explanation of the meaning (and even: essence) of logical connectors and quantifiers. A formal framework for proof writing is necessary to point out their mistakes in reasoning which, due to ambiguity, are always arguable in proofs written in natural language. Manual proof checking becomes perfectly rigourous – indeed it can be automatized, but this is another issue – and moreover it provides precious hints for proof search. More technical advantages are discussed below in section 4.1. Though this material is especially relevant as an introduction to formal methods, we claim that, more generally, it illustrates several key notions of computer science:

- Case analysis
- Tree-like data structures
- Modularity
- Divide and conquer problem solving
- Variables and scopes
- Rule-based formalisms (preparation to more advanced courses)
- Good support for discussing the relation between syntax and semantics
- Introduction to proof-assistants

Let us add that it also provides a good help for writing rigourous and accurate proofs by induction. However, some pitfalls have to be avoided. The way some notions are introduced is sensitive, and some notations have to be carefully designed in order to keep manageable size of interesting proofs without loss of precision.

Our thesis is supported by our experience with the use of GPND in an introductory course on logic, given to first year undergraduate students from 2005 to 2009. The rest of the paper is organized as follows. In Section 2, we present the scientific background, i.e. a short account of natural deduction. In section 3, we outline the contents of the course we gave since 2005. In section 4, we discuss some issues related to the previous experiment as well as possible extensions. We conclude in section 5.

2 Background: Natural Deduction

Natural Deduction was invented by Gerhard Gentzen [6] and further studied by Dag Prawitz [10] for the meta-theoretical study of first-order logic. In contrast with Hilbert's style deduction systems, characterized by few inference rules and many axioms, Gentzen's systems have only one axiom and many inference rules. A strong point of his approach is that each connector is considered separately, providing a intrisic meaning for it: intuitively, each connector $*$ is defined by the canonical way to prove a formula having $*$ as its principal connector (introduction rules), or to exploit a formula having $*$ as its principal connector (elimination rules). The rules are recalled in figure 1. All hypotheses have a name such as h_n. Discharged hypotheses are distinguished by square brackets around their name (e.g. $[h_n]$), and the place where they are discharged appears in the label of the rule (then we know that this hypothesis is no longer available below this rule).

The main meta-theoretical property of Gentzen's systems is the cut-elimination theorem saying that, basically, proofs can be normalized is a way such that the last rule is an introduction rule for the principal connector of the conclusion [10,7]. Important corollaries are the subformula property (delimiting the proof search space) and the consistency of logic (without reference to model-theoretic semantics).

Here, we are more interested in the pedagogical value of GPND proof-trees. We think that it comes from their intrisic features: they are concrete, intuitive, and according to some logician philosophers, they reflect exactly proof objects (again, see [11]). The latter fact is particularly evident if one considers GPND proof-trees as another syntax for typed lambda-calculus, through the Curry-Howard-De Bruijn isomorphism [3,8,2] (it is not the case in another popular representation of natural deduction using sequents, see 4.3 for details).

If we compare with the Boolean approach to teaching logic, proof-trees are certainly more complex than Boolean values but, to some respect, they look concrete and may be perceived as less abstract than Boolean functions. We just may regret that shortcuts using Boolean algebraic laws are not for free in natural deduction. In Boolean algebra, equivalence is the same as equality, whereas here, it is a congruence: we can show that if $A \Leftrightarrow B$, then for any context $\mathcal{C}[\,.\,]$, we have $\mathcal{C}[A] \Leftrightarrow \mathcal{C}[B]$. The proof is by induction on the structure of contexts. It is not very difficult and can be understood by good students, at least for propositional logic, and is a good introduction to the metatheorical study of logic but we could not afford to present it at the level considered in our pedagogical experience.

Fortunately, it turns out that algebraic laws are mainly useful when handling with large propositionnal formulas, which is not the case in our exercises. In places where, say, commutativity, associativity or replacement of $\neg A \Rightarrow \neg B$ with $B \Rightarrow A$, could be used, they can easily be bypassed.

3 Course Outline

The course was designed to introduce logical reasoning to students without previous systematic exposition to logic, in order to prepare them to further courses on computational models, automata and languages, program specification and verification, formal methods, etc. Despite some basic practice in mathematics, many of them have gaps in dealing with proofs and even in capturing the meaning of implication and quantifiers.

Our aim is then to provide an intuition of logical connectors and proofs using 1) a systematic approach based on the structure of the formula to prove, 2) a careful and explicit treatment of quantifiers and 3) a computational data-structure able to implement these requirements, namely proof-trees.

$$\frac{A \qquad B}{A \wedge B}\wedge_{\mathrm{I}} \qquad\qquad \frac{A \wedge B}{A}\wedge_{\mathrm{E1}} \qquad\qquad \frac{A \wedge B}{B}\wedge_{\mathrm{E2}}$$

$$\frac{\begin{array}{c}\overbrace{A}^{[h_n]}\\ \vdots \\ B\end{array}}{A \Rightarrow B}\Rightarrow_{\mathrm{I}[h_n]} \qquad\qquad \frac{A \Rightarrow B \qquad A}{B}\Rightarrow_{\mathrm{E}}$$

$$\frac{A}{A \vee B}\vee_{\mathrm{I1}} \qquad \frac{B}{A \vee B}\vee_{\mathrm{I2}} \qquad \frac{A \vee B \qquad \begin{array}{c}\overbrace{A}^{[h_n]}\\ \vdots \\ C\end{array} \qquad \begin{array}{c}\overbrace{B}^{[h_m]}\\ \vdots \\ C\end{array}}{C}\vee_{\mathrm{E}[h_n,h_m]}$$

$$\frac{\bot}{A}\bot_{\mathrm{E}} \qquad\qquad \neg A \stackrel{\mathrm{def}}{=} A \Rightarrow \bot$$

$$\frac{}{A \vee \neg A}\,PEM \qquad\qquad \frac{\neg\neg A}{A}\neg\neg_{\mathrm{E}}$$

$$\frac{\begin{array}{c}\overbrace{H_1(\_)}^{h_1}\ldots\overbrace{H_n(\_)}^{h_n}\\ \vdots \\ P(x_0)\end{array}}{\forall x\ P(x)}\forall_{\mathrm{I}} \qquad\qquad \frac{\forall x\ P(x)}{P(t)}\forall_{\mathrm{E}(\frac{x}{t})}$$

Side conditions for $\forall$I : x_0 must not be free in any available hypothesis $h_1 \ldots h_n$.

$$\frac{P(t)}{\exists\, x\ P(x)}\exists_{\mathrm{I}} \qquad\qquad \frac{\exists\, x\ P(x) \qquad \begin{array}{c}\overbrace{P(x_0)}^{[h_n]}\\ \vdots \\ C\end{array}}{C}\exists_{\mathrm{E}[h_n]}$$

Side conditions for $\exists_{\mathrm{E}}$:

- in the proof of C from $P(x_0)$, x_0 must not be free in any available hypothesis but h_n;
- x_0 must not be free in C.

Fig. 1. Gentzen-Prawitz Natural Deduction Rules

3.1 Proof Trees

Just to start with, we assume an intuitive and rough knowledge of $\wedge$, $\vee$ and $\Rightarrow$. The first new idea to become familiar with is the notion of proof-tree. A difficulty with GPND is that deductions, in general, depend on hypotheses and that the stock of hypotheses vary when one progresses in the reading of a proof. We then chose to postpone this issue and to use, in the first lesson, only deductions having no effect on available hypotheses. To this effect we could start with GPND rules such that $\wedge_{\text{I}}$, $\wedge_{\text{E1}}$, $\wedge_{\text{E2}}$, $\Rightarrow_{\text{E}}$, $\vee_{\text{I1}}$, $\vee_{\text{I2}}$. However this would break a systematic exposition of the rules. We therefore slightly cheat in a first stage: we introduce ad-hoc inference rules, relevant to the formalization of a toy reasoning, such as TRI (transitivity of implication) and DLI (disjunction on the left of an implication).

$$\frac{A \Rightarrow B \qquad B \Rightarrow C}{A \Rightarrow C}\,\text{TRI} \qquad\qquad \frac{A \Rightarrow C \qquad B \Rightarrow C}{(A \vee B) \Rightarrow C}\,\text{DLI}$$

The specific rules are not important at this stage. We aim at teaching a new game, where the key ideas are:

- The notion of inference rule, with premises, conclusion, and justification (a name used as a label for an inference rule).
- Checking that an inference rule is correctly applied is an easy mechanical task.
- A rule is actually a schema (propositional variables can be replaced with any proposition).
- A proof-tree relates hypotheses (on the top) to one conclusion (at the bottom).
- A proof-tree is built from inference rules.
- Generalization and modularity: one can build a proof tree from subtrees.

The last items are illustrated in a very intuitive way, using examples following the diagrams of figures 2 and 3, where numbers represent propositions[1]. Figure 3 also illustrates a situation where the same hypothesis can be used several times[2]. The rules of the game change very quickly (from the second lesson), but not its shape. Actually, playing with somewhat complicated rules (involving 3 or 4 occurrences of connectors) drives us to a quest for convincing elementary rules.

Before going further, let us mention that we can *name* proof-trees and use such names as justifications for non-elementary proof steps. We introduce in this way derived inference rules, in advanced chapters.

[1] Technically, natural deduction distinguishes the name (here: a number) of a proposition and what the proposition stands, e.g. $A \wedge B \Rightarrow C \vee D$. It may even happen that two different names stand for the same proposition. Of course such details are beyond the scope of the lesson.

[2] In general, a hypothesis can be used 0, 1 ore several times.

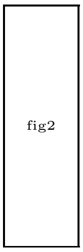


Fig. 2. Branching proof-trees together

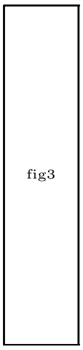


Fig. 3. Abstraction of a proof-tree

3.2 Propositional Connectors

We keep the presentation of Gentzen, where each rule deals with only one connector. In some sense, GPND rules provide a semantics to the corresponding connector, by explaining the canonical ways to prove and to exploit a formula governed by this connector. We start with $\wedge$, which has the simplest rules, and illustrate them on a proof of $B \wedge A$ assuming $A \wedge B$.

The next connector ($\Rightarrow$) is the most important for several reasons:

- any theorem has at least one occurence of $\Rightarrow$ (unless PEM – Principle of Excluded Middle – is used) : a theorem is the conclusion of a proof-tree where all hypotheses are discharged;
- it is often misunderstood by students, and
- it is the fundamental place for discussing hypotheses management.

That said, $\Rightarrow_{\mathrm{E}}$ is well-known (just another name for *modus-ponens*) the statement of $\Rightarrow_{\mathrm{I}}$ is very natural, as it sticks to the common practice for proving $A \Rightarrow B$: suppose A, then prove B.

The hypothesis h_n mentionned in $\Rightarrow_{\mathrm{I}[h_n]}$ is said to be *available* in the sub-proof-tree concluding to B. This rule has a special interest for computer scientists, as it illustrates the notion of scope, which is here applied on hypotheses : the scope of an hypothesis is its availibility domain. Note that the scope is here at the level of proof-trees. In particular, we insist on maintaining a clear separation, using boxes, between different sub-proof-trees equipped with their hypotheses, to represent scopes.

Interesting exercises, ranging from easy to difficult, can be proposed using only $\wedge$ and $\Rightarrow$, or even just $\Rightarrow$. Here are some examples – note that $\Rightarrow$ associates to the right:

- $A \wedge B \Rightarrow B \wedge A$ (very easy);
- $[(A \wedge B) \Rightarrow C] \Rightarrow (A \Rightarrow B \Rightarrow C)$, and conversely: intuitively, there are two equivalent ways to express the idea of "and if";
- $(A \Rightarrow B \Rightarrow C) \;\Rightarrow\; ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$;
- $A \Rightarrow B \Rightarrow A$ (somewhat troubling);
- $(A \Leftrightarrow\; (A \Rightarrow B)) \Rightarrow B$.

In the last one, $P \Leftrightarrow\; Q$ is an abbreviation for $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$. It is basically the essence of diagonal arguments: replacing B with absurdity $\perp$ and the definition of $\neg$ (see below), it means that A cannot be equivalent to $\neg A$. Here, the argument is developed constructively, without case analysis on $A \vee \neg A$. An interesting challenge is to find a solution without repeating sub-proof-trees: for more advanced students, it illustrates the notion of *cut*, similar to a lemma in informal practice.

We finish this part with disjunction. The two introduction rules are obvious. The elimination rules corresponds to case analysis and requires hypotheses management – hence another opportunity to discuss on scopes. Note that students are tempted to invent a $\vee_{\mathrm{E}}$ rule with 2 conclusions, something like

$$\frac{A \vee B}{A \qquad B}\ \textsc{wrong-}\vee_{\mathrm{E}}$$

Here we have to explain that a proof-tree with 2 (and then, in general, many) conclusions is a complicated beast. In some sense those conclusions should be handled separately (otherwise, we could deduce $A \wedge B$ from the previous deduction). However, separate deductions starting from A and from B need eventually to be synchronized, once the same conclusion is reached. But further complications will happen, typically if A (or B) has to be used several times. So we keep things simple, sticking to the shape of a tree.

Let us mention here another challenging exercise, which requires a good understanding of $\Rightarrow_{\mathrm{I}}$

– $((A \vee A \Rightarrow C) \Rightarrow C) \Rightarrow C$

It can related to the elimination of double negations, to come later.

3.3 Quantifiers

Before explaining the rules, a number of notions are needed on what is usually called a first-order language. Hoowever, a perfectly formal and rigourous presentation would be counter-productive at this level. Students have an operationnel knowledge of terms made of function symbols, constants and variables. So we insist only on sensitive concepts: free and bound variables, their scope (at the level of the syntax of formulas, here), substitution of free variables. Formulas which differ only on the name of free variables are considered identical. The rule for $\forall$ elimination is obvious. We choose to provide the substitution in the label: $\forall_{\mathrm{E}}(\frac{x}{t})$ means that the (free occurences of) variable x will be substituted with t. This level of precision is useful (even needed!) for students, we go back to this issue in section 4.

Rule $\forall_{\mathrm{I}}$ raises the sensitive question of fresh variables. In the premise $P(x_0)$, x_0 stands for an *arbitrary* variable. What does it mean? It is easy to explain that “arbitrary” means “not subject to an hypothesis” or, more accurately, “not subject to an available hypothesis”, hence “not free in any available hypothesis”. We could say as well that the premise $P(x_0)$ must not be in the scope (at the level of proofs) of an hypothesis on x_0. Mastering hypotheses handling is then crucial at this stage. We require students to write explicitly this side condition as s $x_0 \notin FV(h_1, \dots h_n)$ and to check it during the proof process.

The explanations about $\exists_{\mathrm{E}}[h_n]$ are along the same lines. We stress that $\exists$ behaves like an infinite version of $\vee$. Hence it is not surprising that the structure of $\exists_{\mathrm{E}}$ is similar to $\vee_{\mathrm{E}}$. But similarly as well, students tend to formalize “we know that $\exists x P(x)$; let x_0 be the witness such that $P(x_0)$” by

$$\frac{\exists x P(x)}{P(x_0)}\ \textsc{wrong-}\exists_{\mathrm{E}}$$

Such a rule leads very quickly to undesired consequences, as it behaves as a $\forall_{\mathrm{E}}(\frac{x}{x_0})$. Indeed, it yields

$$\cfrac{\cfrac{\exists x P(x)}{P(x_0)}\,\text{WRONG-}\exists_{\mathrm{E}}}{\forall x\ P(x)}\,\forall_{\mathrm{I}}$$

hence $(\exists x P(x)) \Rightarrow (\forall x\ P(x))$. This usual error is an opportunity to discuss on the effect and the consequences of WRONG-$\exists_{\mathrm{E}}$. The right $\exists_{\mathrm{E}}$ rule takes a situation where one has a proof of $\exists\, x\ P(x)$ and, from *any witness* x_0 such that $P(x_0)$, one can build a proof-tree ∇ having some conclusion C, possibly using other premises. Then, one can infer C. Note that $P(x_0)$ may be used several times in ∇, while it is not feasible in WRONG-$\exists_{\mathrm{E}}$. Students agree that each use of $\exists_{\mathrm{E}}$ would produce a different x_i. Another important intuition is that, in $\forall_{\mathrm{E}}(\frac{x}{x_0})$, x_0 does not come from the premise $\forall x\ P(x)$ it is typically given by a $\forall_{\mathrm{I}}$ below in the proof-tree – the proof is often built in a bottom-up manner, initially. This is to be contrasted with in $\exists_{\mathrm{E}}$, where x_0 is a witness contained in the proof of the premise $\exists x P(x)$.

At this point students are able to find proof-trees for formulas such as $(\exists x \forall y\ R(x, y)) \Rightarrow (\forall y\ \exists x R(x, y))$ and to become aware that the converse is not a theorem.

3.4 Absurd and Negation

The absurd proposition $(\bot)$ has no introduction rule. We mention that $\bot$ cannot be proved in the empty context, as a corollary of the cut-elimination theorem (the latter is only stated, its proof is beyond the scope of our course).

Negation is defined by $\neg A \overset{\text{def}}{=} A \Rightarrow \bot$.

It is interesting to note that $\bot$ and $\neg$ can be introduced very late, after first-order notions. Many interesting exercises can be done without $\neg$. In fact, we could delay further these connectors, after equalities and induction. A large amount of logic material can be developed without reference to False. For instance, it is the case for algebraic properties of $+$ and $\times$ on natural numbers.

However we decided to talk about $\neg$ at this stage in order to introduce classical reasoning, using either the Principle of Excluded Middle (PEM) or $\neg\neg_{\mathrm{E}}$. This is also the place for discussing about constructive reasoning – something which is certainly more sensitive in the framework of computer science than mathematics.

Among the exercises which can be proposed at this stage, let us mention some puzzles such as

- $\neg\neg A \wedge \neg\neg B \Leftrightarrow \neg\neg(A \wedge B)$
- $\neg\neg(\neg\neg A \vee \neg\neg B \Leftrightarrow \neg\neg(A \vee B))$.

They can be proved with $\neg\neg_{\mathrm{E}}$, but finding a solution without this rule and without PEM) requires a good understanding of implication.

3.5 Equational reasonning

If o_1 and o_2 are “identical”, it is clear that everything we prove about o_1, holds for o_2 as well. This common kind of equational reasoning, often called Leibniz’s law, is embodied in our framework as equality elimination. We recall it in Figure 4, together with equality introduction, which is the only general axiom about =, that is, equality is reflexive. It is easy to derive symmetry and transitivity of = from $=_{\mathrm{I}}$ and $=_{\mathrm{E}}$. In principle, it is possible to present any equationnal reasoning as proof-trees using only $=_{\mathrm{I}}$ and $=_{\mathrm{E}}$. However, it turns out quite tedious and lengthy. Therefore we prefer to present an equational reasoning as illustrated on the right hand of Figure 5. It can be shown (by induction on number of rewriting steps) that such a proof can be put in the proof-tree format. Implementing that transformation on proof-trees could be a programming exercise in a companion course on functional programming. This proof $\mathcal{E}_i$ is then abstracted under the form of a multiple inference step $\dfrac{\overbrace{}^{h_1}\dots\overbrace{}^{h_n}}{U = Z}\,\mathcal{E}_i$. Note that justifications used in $\mathcal{E}_i$ can themselves refer to separate proof-trees.

$$\dfrac{}{t = t}\,=_{\mathrm{I}} \qquad\qquad \dfrac{a = b \qquad P(a)}{P(b)}\,=_{\mathrm{E}}$$

Fig. 4. Rules for equality

$$\dfrac{\overbrace{}^{h_1}\dots\overbrace{}^{h_n}}{U = Z}\,\mathcal{E}_i$$

where $\mathcal{E}_i$ is

$$\mathcal{E}_i\begin{cases} \quad U & \\ = & \{\text{justification that } U = V \text{ provided } h_1 \dots h_n\} \\ \quad V & \\ \vdots & \\ \quad Y & \\ = & \{\text{justification that } Y = Z \text{ provided } h_1 \dots h_n\} \\ \quad Z & \end{cases}$$

Fig. 5. Equational reasoning

3.6 Definitions

A very useful device for keeping proofs manageable is to use *definitions.* For instance a sub-formula such as $\forall x\ P(x) \Rightarrow Q(x, y)$ can be abbreviated as $R(y)$, provided we define $R(u) \ \overset{\mathrm{def}}{=\!=}\ \forall x\ P(x) \Rightarrow Q(x, u)$. Free variables have to be properly taken into account. The definiendum can be freely replaced with the corresponding definiens and conversely. Technically speaking, in proof-theory, such steps are not considered as deductions, but follow from the conversion rule, which means that a proof-tree having A as its conclusion *is* a proof-tree having B as its conclusion, when $A \overset{\mathrm{def}}{=\!=} B$ or when $B \overset{\mathrm{def}}{=\!=} A$ (for a more general setting, we refer the reader to *deduction modulo* as defined in [4]). However, for pedagogical purposes, we prefer to make such steps explicit, at least at the beginning. In order to distinguish such steps from regular deduction steps, we present them using dot lines instead of plain lines. In the previous example, we could then write

$$\frac{D \wedge (\forall x\ P(x) \Rightarrow Q(x,y)) \vee E}{D \wedge R(y) \vee E}\ R\,\mathrm{def}$$

The reverse replacement can also be done by invoking $R\,\mathrm{def}$.

3.7 Set theoretic constructs

Definitions are extensively used when dealing with set-theoretic constructs. Besides the extensionnality axiom

- $A = B \Leftrightarrow (\forall x,\ x \in A \Leftrightarrow x \in B)$

we have the following definitions:

- $A \subseteq B \stackrel{\mathrm{def}}{=} (\forall x\, x \in A \Rightarrow x \in B)$
- $x \in A \cap B \stackrel{\mathrm{def}}{=} x \in A \wedge x \in B$
- $x \in A \cup B \stackrel{\mathrm{def}}{=} x \in A \vee x \in B$
- $x \in \emptyset \stackrel{\mathrm{def}}{=} \bot$
- $x \in \{a\} \stackrel{\mathrm{def}}{=} x = a$
- $x \in \{a_1, \ldots a_n\} \stackrel{\mathrm{def}}{=} x = a_1 \vee \ldots \vee x = a_n$
- $A \in \mathcal{P}(B) \stackrel{\mathrm{def}}{=} A \subseteq B$

From these definitions we prove convenient derived rules, given in Figure 6. From an epistemological point of view, students get a taste on mathematical foundations: relying on a solid basis to define new objects and some convenient abstract rules to reason about.

3.8 Induction

The usual induction principle is formalized by the following deduction rule:

$$\frac{P(0) \qquad \forall n\ \ P(n) \Rightarrow P(S(n))}{\forall n\ P(n)}\ \text{nat-rec}$$

We also provide Peano axioms and then can propose exercises on elementary algebraic properties of addition and multiplication.

$$\frac{}{\forall n\ \ n + 0 = n}\ {+0} \qquad \frac{}{\forall n\, \forall m\ \ n + S(m) = S(n+m)}\ {+\mathrm{S}}$$

$$\frac{}{\forall n\ \ n \times 0 = 0}\ {\times 0} \qquad \frac{}{\forall n\, \forall m\ \ n \times S(m) = (n \times m) + n}\ {\times \mathrm{S}}$$

$$\frac{\begin{array}{c}\overbrace{x \in A}^{[n]}\\ \vdots\\ x \in B\end{array}}{A \subseteq B}\,{\scriptstyle \Rightarrow_{\mathrm{I}}\forall_{\mathrm{I}}\subseteq[n]} \qquad \frac{x \in A \qquad A \subseteq B}{x \in B}\,{\scriptstyle \subseteq\forall_{\mathrm{E}}\Rightarrow_{\mathrm{E}}} \qquad \frac{\begin{array}{cc}\overbrace{x \in A}^{[n]} & \overbrace{x \in B}^{[m]}\\ \vdots & \vdots\\ x \in B & x \in A\end{array}}{A = B}\,{\scriptstyle \mathrm{ext}[n,m]}$$

$$\frac{x \in A \qquad x \in B}{x \in A \cap B}\,{\scriptstyle \wedge_{\mathrm{I}}\cap} \qquad \frac{x \in A \cap B}{x \in A}\,{\scriptstyle \cap\wedge_{\mathrm{E1}}} \qquad \frac{x \in A \cap B}{x \in B}\,{\scriptstyle \cap\wedge_{\mathrm{E2}}}$$

$$\frac{x \in A}{x \in A \cup B}\,{\scriptstyle \vee_{\mathrm{I1}}\cup} \qquad \frac{x \in B}{x \in A \cup B}\,{\scriptstyle \vee_{\mathrm{I2}}\cup} \qquad \frac{\begin{array}{ccc} & \overbrace{x \in A}^{[n]} & \overbrace{x \in B}^{[m]}\\ & \vdots & \vdots\\ x \in A \cup B & P & P\end{array}}{P}\,{\scriptstyle \cup\vee_{\mathrm{E}}[n,m]}$$

Fig. 6. Derived rules for set-theoretic notations

Predicates $\leq$ and $<$ can be defined by $m \leq n \overset{\mathrm{def}}{=\!=} \exists k\; n = k + m$ and $m < n \overset{\mathrm{def}}{=\!=} S(m) \leq n$. Then we can derive "strong" (or noetherian) induction on natural numbers:

$$\frac{\forall n\; (\forall m\; m < n \Rightarrow P(m)) \Rightarrow P(n)}{\forall n\; P(n)}\,{\scriptstyle \mathrm{strong\ nat-rec}}$$

A natural extension is to work on structural induction on ML-style lists or binary trees.

3.9 Example

In order to illustrate the above ideas, we consider the example depicted in Figure 7, which is a typical examination problem. In order to simplify notations, we do not type variables, but we want to emphasize that n is a natural variable (hence we can apply inductive reasoning over it). Also, for sake of simplicity, we start by defining some predicates. Hence, predicate H_1 states that everybody has a father. Predicates H_2 states that everybody is its own 0-ancestor, and predicate H_3 states that the $n + 1$-ancestor is defined as the father of the n-ancestor. We want to prove that for any $n \in \mathbb{N}$, everybody has an n-ancestor, that is, for any $n \in \mathbb{N}$, the property $Q(n)$ holds. The way we teach this example is the following one. First, we remark that we want to prove some statement, without any additional hypothesis. Fortunately, the goal corresponds to an implication. Hence, it suffices to prove the right part, admitting the left part as an hypothesis. Formally, this corresponds to an aplication of an $\Rightarrow_I$ rule. We continue this bottom-up proof by applying the rule $\Rightarrow_I$ twice again. Now, we face a property of the kind $\forall n$ ... where n is a variable ranging over naturals, so we can apply induction. The next goal is split into two other sub-goals. Here, we can see the manner we compose proofs, for example the tree T_1 can be proved separately, and then it can be plugged in the overall proof. But we have to pay attention to the hypothesis that remain active at the end of the proof T_1. Other interesting points are the way we unfold definitions, and the use of equational reasoning.

4 Pedagogical Issues and Assessment

4.1 Technical Advantages of GPND for Teaching Logic

We already mentionned that GPND proof-trees reflect usual reasoning much better than the truth-table approach. This is especially clear on implication. The fact that the rules for $\Rightarrow$ and for $\wedge$ look completely different is a chance, as it helps beginners not to confuse between these two connectors.

A very interesting point, from a pedagogical perspective, is that proof trees allow us to point out errors with accuracy. It occurs quite often that some student comes with rough and obscure ideas and still believes that his argument is good. It is much more difficult to show him where are his mistakes on informal or semi-formal writings than on proof trees. If a rule is wrongly applied, we can first say "this does not conform to the you Law, on which we agreed" and may add: "if your rule was right, you would get this or that undesired consequence". This turns out quite convincing with all kind of students. Here are some typical errors that can be pointed out in GPND proof style:

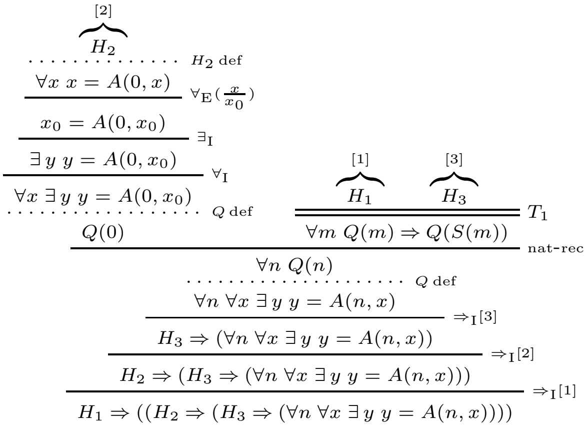
- Incorrect use of a deduction rule (especially $\vee_{\mathrm{E}}$).
- Violation of the scope of an hypothesis (for instance, a hypothesis available in a branch of a case analysis is used in the other branch).
- Violation of the side condition of $\exists_{\mathrm{E}}$ or $\forall_{\mathrm{I}}$, using a "convenient" choice for $\exists x$ or $\forall x$ instead of a fresh variable.
- Exploiting an implication, or a rule, without proving its premises.

Such errors correspond to typical wrong reasoning written informally.

Another benefit is that GPND enforces a precise understanding of the distinction between available and discharged hypotheses. This is particularly important for a rigorous treatment of quantifiers and of inductions, especially when we they are embedded. It is sometimes crucial, when proving a property of the form $\forall n\ \forall m\ P(n, m)$ by induction on n, that the inductive property $\forall m\ P(n, m)$ remains universally quantified, because the m we need for $S(n)$ may come from a different value in the induction hypothesis. A well-known example, among others, is the proof of strong induction using basic induction on the property $\forall m\ m \leq n \Rightarrow P(m)$.

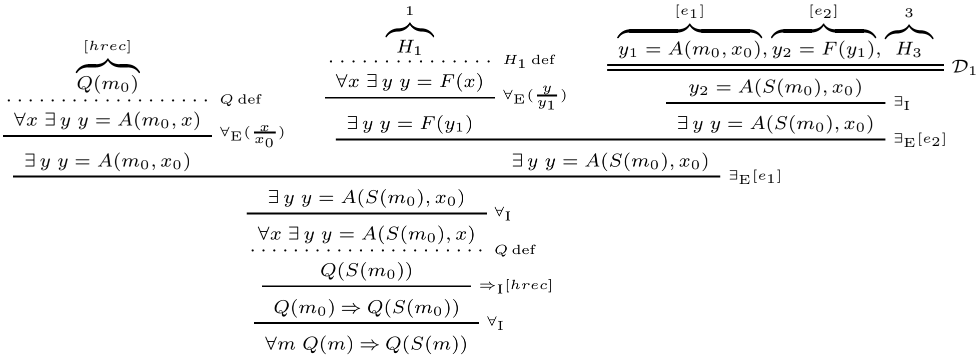
Let us now mention a number of issues showing the relevance of the GPND approach to computer science, including formal methods.

Let us note $Q(n) \overset{\text{def}}{=\!=} \forall x\ \exists\, y\ y = A(n,x)$, $H_1 \overset{\text{def}}{=\!=} \forall x\ \exists\, y\ y = F(x)$, $H_2 \overset{\text{def}}{=\!=} \forall x\ x = A(0,x)$, $H_3 \overset{\text{def}}{=\!=} \forall n\, \forall x\ F(A(n,x)) = A(S(n),x)$.



[2]
H_2
H_2 def
$\forall x\ x = A(0,x)$
$\forall_{\mathrm{E}}(\frac{x}{x_0})$
$x_0 = A(0,x_0)$
$\exists_{\mathrm{I}}$
$\exists\, y\ y = A(0,x_0)$
$\forall_{\mathrm{I}}$
[1]
[3]
H_1
H_3
$\forall x\ \exists\, y\ y = A(0,x_0)$
Q def
T_1
$Q(0)$
$\forall m\ Q(m) \Rightarrow Q(S(m))$
nat-rec
$\forall n\ Q(n)$
Q def
$\forall n\, \forall x\ \exists\, y\ y = A(n,x)$
$\Rightarrow_{\mathrm{I}}[3]$
$H_3 \Rightarrow (\forall n\, \forall x\ \exists\, y\ y = A(n,x))$
$\Rightarrow_{\mathrm{I}}[2]$
$H_2 \Rightarrow (H_3 \Rightarrow (\forall n\, \forall x\ \exists\, y\ y = A(n,x)))$
$\Rightarrow_{\mathrm{I}}[1]$
$H_1 \Rightarrow ((H_2 \Rightarrow (H_3 \Rightarrow (\forall n\, \forall x\ \exists\, y\ y = A(n,x))))$

where the tree T_1 is



1
$[e_1]$
$[e_2]$
3
$[hrec]$
H_1
$y_1 = A(m_0,x_0), y_2 = F(y_1),\ H_3$
$\mathcal{D}_1$
$Q(m_0)$
H_1 def
$\forall x\ \exists\, y\ y = F(x)$
$\forall_{\mathrm{E}}(\frac{y}{y_1})$
$y_2 = A(S(m_0),x_0)$
Q def
$\exists_{\mathrm{I}}$
$\forall x\ \exists\, y\ y = A(m_0,x)$
$\forall_{\mathrm{E}}(\frac{x}{x_0})$
$\exists\, y\ y = F(y_1)$
$\exists\, y\ y = A(S(m_0),x_0)$
$\exists_{\mathrm{E}}[e_2]$
$\exists\, y\ y = A(m_0,x_0)$
$\exists\, y\ y = A(S(m_0),x_0)$
$\exists_{\mathrm{E}}[e_1]$
$\exists\, y\ y = A(S(m_0),x_0)$
$\forall_{\mathrm{I}}$
$\forall x\ \exists\, y\ y = A(S(m_0),x)$
Q def
$Q(S(m_0))$
$\Rightarrow_{\mathrm{I}}[hrec]$
$Q(m_0) \Rightarrow Q(S(m_0))$
$\forall_{\mathrm{I}}$
$\forall m\ Q(m) \Rightarrow Q(S(m))$

and $\mathcal{D}_1$ is

$$\mathcal{D}_1 \left\{ \begin{array}{ll} & y_2 \\ = & \quad \{\text{hypothesis } e_2\} \\ & F(y_1) \\ = & \quad \{\text{hypothesis } e_1\} \\ & F(A(m_0,x_0)) \\ = & \quad \{H_3 \text{ by hypothesis 3, with } \forall_{\mathrm{E}}\ \frac{n}{m_0} \text{ and } \forall_{\mathrm{E}}\ \frac{x}{x_0}\} \\ & A(S(m_0),x_0) \end{array} \right.$$

Fig. 7. Example “Everybody has n-ancestors”

Case analysis. The elimination rule for $\vee$ states exactly how a disjunctive piece of information can be exploited. This is clearly related to algorithmic constructs such if...then...else..., case...of..., switch...

Tree-like data structures. Trees are a ubiquitous concept in computer science. Their handling is intuitive. Here is an opportunity to introduce and manipulate them at an abstract level, without reference to an implementation.

Modularity. Even middle-size proofs cannot be displayed using a monolithic proof-tree on a single sheet of paper. Structuring a proof in sub-trees with a clear interface allows one to handle this issue. Here, the interface is defined by the set of hypotheses and the conclusion.

Problem solving. Faced to proving a goal from given hypotheses, many steps can be carried out just by examining the form of the formulas at hand. If the conclusion is among the hypotheses, we are done. Else, if the conclusion is not atomic, it can be decomposed along a divide-and-conquer approach, using an introduction rule. However, some of them ($\vee_{\mathrm{I1}}$, $\vee_{\mathrm{I2}}$ and $\exists_{\mathrm{I}}$) are dangerous as they may drive into a dead end We warn the students to postpone as far as possible the use of these rules. These are the places where thinking on the contents of hypotheses and creativity are needed. This method can be carried out in parallel on the corresponding informal reasoning.

Variables and scope. It is clear that the notions of free, bound variables with their scopes are developed when introducing the syntax of first-order formulas. Admittedly, this comes from formal logic, not specifically GPND. Looking at programming languages, logic variables are closer to the concept to be found in the functionnal paradigm than in the imperative paradigm.

What is more specific to GPND is that scopes are also related to hypotheses, more exactly names of hypotheses: the scope of an hypothesis is the largest subtreee where it is available (still not discharged). One can even speak about local and global hypotheses. Scopes are then discussed already in the framework of propositional logic.

Rule-based formalisms. Many formalisms used in computer science a rule-based presentation: typing systems, operational semantics, etc. Studying GPND is then a good training in order to prepare more advanced courses.

Introduction to proof assistants. In the area of formal methods and software verification, well-recognized proof assistants such as Coq [12,1] and Isabelle/HOL [9] are available and commonly used. Their theoretic foundations are logic systems quite close to GPND.

4.2 Pedagogical Issues and pitfalls

A number of pedagogical issues have to be taken into account, in order to make GPND an efficient tool for teaching.

We already mentioned that proof trees allow us to point out mistakes with accuracy. To this effect, we insist that justifications (labels used in deduction steps) are mandatory: often students suffer from a lack of precise ideas on what their are really doing, or at least from a poor ability to communicate their arguments. In this spirit we tend to demand more than what is generally given in textbooks. For instance, the elimination rule for $\forall$ makes explicit the term t to be substituted to the quantified variable x: $\forall_{\mathrm{E}}(\frac{x}{t})$.

There is a pedagogical issue with implication: in GPND, it is impossible[3] to get a theorem without using $\Rightarrow_{\mathrm{I}}$. Hence we have to consider hypothesis management very early. As explained in 3.1, we fix this issue by temporary considering fake rules such as TRI and DLI. A drawback is that some students tend to think that any rule is good, provided it looks plausible. So we have to insist heavily, from the beginning, that TRI and so on should be forgotten and that the right rules are coming. A complementary approach is to work on deductions under hypotheses with simple rules such as $\Rightarrow_{\mathrm{E}}$ $\forall_{\mathrm{E}}$ and rules for $\wedge$. In fact we let students discover these rules in the first exercise session.

We already mentionned some wrong attempts about $\vee_{\mathrm{E}}$ and $\exists_{\mathrm{E}}$, respectively in 3.2 and 3.3. How to deal with space consuming proofs was considered from the beginning (3.1), through modularity, and using a special notation for equational reasoning (see 3.5).

4.3 An alternative to GPND: sequents

Natural deduction can be presented in terms of *sequents* $\Gamma \vdash C$, where Γ is a multiset of formulas and C a formula, whose intuitive meaning is "given the conjunction of hypotheses in Γ, the conclusion C holds". Rules have several sequents as premises and a sequent A formula A is a theorem if the sequent $\vdash A$ can be derived. A *proof tree* is a tree labelled as follows: leaves are labelled by axioms, i.e. sequents $\Gamma \vdash C$ where $C \in \Gamma$. Although this approach may be prefered for the meta-theoretical study of natural deduction [5], it puts the emphasis more on provability than on proofs.

Note that, from a pedagogical perspective, the sequent based presentation is closer to inference systems used for typing or structured operational semantics. But it is a bit far from the objectives of an introductory course to logic. Moreover, once somebody is familiar with a deduction system, it is reasonnable to expect that she or he can easily move to another presentation of it or to another inference system.

[3] At least for intuitionistic logic. But it is clear that PEM cannot be exploited without hypothesis management, except if the conclusion is just an instance of $A \vee \neg A$. For example, all theorems of the form $\neg P \vee Q$, where Q can be deduced from P, are proved by case analysis on $P \vee \neg P$, i.e. using $\vee_{\mathrm{E}}$.

4.4 Assessment

This course was given to an audience of 150 to 200 students per year, with the following rythm: one lecture (1h30) and one exercise class (1h30) per week, during 11 weeks. About 3 weeks are needed for discovering proof-trees, the 3 first propositionnal connectors; then another 3 weeks for quantifiers and negation; the next 3 weeks are devoted to set-theoretic notions ans the last 2 weeks to induction.

The course got a good ranking from the students, which is quite satisfactory since most of the material is new for them. It turns out that they like to play with trees. However, our main goal in introducing formal proofs as a first year course was to improve the ability of students in reasoning beyond the formal framework of GPND, that is detecting wrong deduction and convincing proof in natural language. We brought students from approximate reasoning in natural language to formal proofs, and we expected them to do the opposite by themselves. After four years we had to admit that we partially failed. Indeed, some students were still handicapped when asked to prove a statements in natural language whereas they were perfectly able to build the proof tree in GPND style. This observation brought us to the conclusion that we need more time transferring the lessons learned in GPND back to the free reasoning. This includes proof guidelines: how to decompose a statement to proof into subgoals, how to find the hypothesis and what should finally be proved and how a proof tree can be told as an argumentative discourse. On this last point we plan to extend the teaching with a project – in collaboration with a course on functional programming – that consists in a systematic translation of a GPND proof tree into a reasoning in natural language.

We are anyway convinced that working on proof-trees help students to get a more structured mind. In order to strengthen the work done so far, connections have to be established with other courses given in second year on logic, automata and languages, proofs and algorithmics. Students happen to ask for building proof-trees 2 or 3 years later. When one of them is stuck at the beginning of a proof, suggesting her or him to start a proof-tree turns out quite helpful. On the teaching side, colleagues have to be convinced that our approach is good and can be reused to some extent. We are confident that progress will be done in this direction, because our teaching team became quite enthusiastic, though most teachers discovered natural deduction in this course.

5 Conclusion

We advocated that GPND is the good way to introduce logic to beginners, at least for students in computer science. Everybody gets a chance to better understand what is a reasoning and to improve her or his reasoning abilities. What about other scholars ? It is often advocated that computer science should be taught much earlier in the curriculum, notably in the highschool. Computer scientists should contribute to this chapter of mathematics. In particular, proof-trees are simple to understand and funny. They require no mathematical background. We think that they could be introduced at the highschool, at least for propositional logic, thus helping scholars in their scientific activities.

Let us finish with some perspectives. We limited ourselves to a pencil and paper approach, mainly because we didn't have enough time slots to do otherwise. We plan to use a proof assistant in a next version of the course. However, we will have to take care of the danger of button-pushing: existing proof assistants are good at helping the user to find proofs and to automatize tedious tasks. Hoawever we want here the user to be aware of the elementary deduction steps. In a pedagogical use, a proof-assistant should just be used as a proof checker.

References

1. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer Verlag, 2004.
2. N. De Bruijn. Automath, a language for mathematics. In *Automation and Reasoning, vol 2, Classical papers on computational logic 1967-1970.* Springer Verlag, 1983. from a technical report, Eindowen, 1968.
3. Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume I. North-Holland Publishing Company, Amsterdam, third printing edition, 1974.
4. G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31:2003, 1998.
5. J. Gallier. Constructive logics: Part i: a tutorial on proof systems and typed λ-calculi. *Theor. Comput. Sci.*, 110(2):249–339, 1993.
6. Gehrard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1935.
7. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.
8. W. A. Howard. The Formulae-As-Types Notion Of Construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., New York, N.Y., 1980.
9. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
10. Dag Prawitz. *Natural Deduction. A Proof-Theoretical Study*, volume 3 of *Stockholm Studies in Philosophy.* Almqvist & Wiksell, Stockholm, 1965.
11. Peter Schroeder-Heister. Validity concepts in proof-theoretic semantics. *Synthese*, 148(3):525–571, February 2006.
12. The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.2*, 2008. `http://coq.inria.fr`.