Extracting Text from Proof

Yann Coscoy
INRIA–Sophia-Antipolis
Yann.Coscoy@inria.fr

Gilles Kahn
INRIA–Sophia-Antipolis
Gilles.Kahn@inria.fr

Laurent Théry
ATT Bell Laboratories
thery@research.att.com

1 Introduction

Almost all computer proof assistants today are used in the following manner. The user states a theorem to prove. Then using a variety of commands that can be recorded in a *proof script*, the user brings the interactive system to a state indicating that the theorem has been proved. The theorem is then archived for later reuse, and the corresponding proof script is kept *preciously*.

There are two basic reasons for safekeeping the proof script. First, a proof must often be verified again in a slightly different context, or a different version of the theorem may be needed. Thus, the script is kept as a model for constructing later variants of a proof. We remark in passing that the command language of some systems does not facilitate this task.

Second, the proof script is used as tangible evidence that the proof was actually carried out, and as a means of communicating its intellectual content. In the context of program verification, such evidence must be presented to industrial auditors. As a vehicle for the communication of proofs, we feel that proof scripts *alone* are largely inadequate. Decoding what a proof script actually does is the province of expert users of a given proof assistant. A proof script invokes a number of sophisticated tactics that are system specific, that attempt to do things that fail and are irrelevant for the final proof. Furthermore, these tactics may be refined as the proof assistant is being improved. On the other hand, a proof script contains invaluable information on the level of abstraction at which the proof is carried out. In her work, A. Cohn ([Cohn88]) tries to produce intelligible text from the proof script.

By contrast, in this paper, we will be concerned with proof assistants that construct a *proof object*, i.e. a data structure that explicitly represents the proof of facts established with the system. Proof objects are built by a number of modern proof assistants ([Coq91, Hol92, Lego92]), but they are rarely used for anything useful. They are generally considered to be exceedingly large and difficult to understand. On the basis of experiments carried out in the last three years with several computer proof assistants, we disagree with this commonly held view and find proof objects useful and important in many respects.

First it is possible to make good sense out of these proof objects, and this is what we will show in this paper. Second, proof objects are far more independent of the proof assistant than proof scripts and they form a better basis for understanding and displaying the intellectual content of a proof. As a result, they are very useful to debug automatic proof tactics. Last, if they can be built incrementally as in ALF [Alf93], proof objects provide a useful interactive feedback on what is going on in the proof.

Presenting proof objects in an intelligible form is non trivial. After a number of unsuccessful experiments with graphical representations, we are now convinced that the best method is to build transducers from proof objects to pseudo natural language.

2 Proof objects and their presentation

Different methods can be used in building proofs: resolution and tableaux methods are popular in the automatic theorem proving community, while natural deduction is favored for interactive proof assistants. Natural deduction, proposed by Gentzen [Gentzen69] and further elaborated by Prawitz [Prawitz65], is *natural* because it formalizes the reasoning used in ordinary mathematical text. This is also why it is popular in interactive theorem provers and the obvious candidate for experiments in producing text from formal proofs.

2.1 Natural Deduction Trees

The original format proposed by Gentzen for natural deduction proof is a tree format. For each connective, two sets of rules are provided. The first set is composed of *introduction rules* and describes how formulas containing such a connective may be formed. The second set is composed of *elimination rules* and describes how formulas governed by this connective can be used. As an example, there is one introduction rule and two eliminations rules for the conjunction:

$$\wedge\, intro : \frac{A \qquad B}{A \wedge B} \qquad\qquad \wedge\, elim : \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

A *deduction* in this system is a composition of instances of these rules forming a tree-like structure. An example of such a deduction is the following tree:

$$\dfrac{\dfrac{A \wedge B}{B}\,{\scriptstyle \wedge elim:} \qquad \dfrac{A \wedge B}{A}\,{\scriptstyle \wedge elim:}}{B \wedge A}\,{\scriptstyle \wedge intro:}$$

An additional mechanism in this formalism is the *discharge* operation which handles hypothetical reasoning. This mechanism can be explained with the introduction rule of implication. The rule is usually displayed as

$$\begin{array}{c} [A] \\ \vdots \\ \dfrac{B}{A \supset B} \end{array} \supset\, intro$$

where the brackets around A means that A can be used freely without justification (as a leaf) in the deduction leading to B. We then say that the hypothesis A is discharged by the rule. A *proof* is a deduction in which all leaves have been discharged. To link precisely an assumption with the rule that discharges it, each assumption is numbered. This number is then mentioned again as a superscript whenever the assumption is used and the rule that discharges an assumption is annotated as well. Applying this convention to the previous rules gives the following

$$\begin{array}{c} [A^i] \\ \vdots \\ \dfrac{B}{A \supset B}\,[\,i] \end{array} \supset\, intro$$

Introduction rules and elimination rules for the usual logical connectives in the context of first-order intuitionist logic are given in Figure 1 page 19.

The simple proof of *præclarum* illustrates the advantages and disadvantages of such a format.

$$\dfrac{\dfrac{\dfrac{\dfrac{x \wedge y}{x}\,{\scriptstyle \wedge\, elim}\quad \dfrac{(x \supset z) \wedge (y \supset t)}{x \supset z}\,{\scriptstyle \wedge\, elim}}{z}\,{\scriptstyle \supset\, elim}\quad \dfrac{\dfrac{x \wedge y}{y}\,{\scriptstyle \wedge\, elim}\quad \dfrac{(x \supset z) \wedge (y \supset t)}{y \supset t}\,{\scriptstyle \wedge\, elim}}{t}\,{\scriptstyle \supset\, elim}}{z \wedge t}\,{\scriptstyle \wedge\, intro}}{x \wedge y \supset z \wedge t}\,{\scriptstyle \supset\, intro}\,[2]}{(x \supset z) \wedge (y \supset t) \supset x \wedge y \supset z \wedge t}\,{\scriptstyle \supset\, intro}\,[1]$$

Using two dimensions to represent the structure of the proof makes operations such as finding the scope of an assumption or finding the premisses of a rule easier. The system EUODHILOS [Euo92] uses this presentation. The drawback is, though, that formulas share the horizontal dimension with the proof structure. Consequently, proof trees tend to grow much more in width than in height. This phenomenon is even amplified by the fact that the leaves, being assumptions, are large formulas. Even for small proofs, we have found the natural deduction proof-tree layout to be impractical.

2.2 Natural Deduction in Linear Format

As an alternative, several authors have proposed linear presentations of natural deduction proofs [Fitch52, Kal80]. This style is used in theorem provers such as Mural [Mur91] and TPS [Tps92]. The basic idea is to represent a proof as a vertical sequence of lines, where each line is either an axiom, an hypothesis, or the consequence of previous lines. To give an idea of such format, here is the same proof of *præclarum* in the format described in [Fitch52]:

1	$(x \supset z) \wedge (y \supset t)$	*hyp*
2	$x \wedge y$	*hyp*
3	x	2, $\wedge$ *elim*
4	$x \supset z$	1, $\wedge$ *elim*
5	z	3 and 4, $\supset$ *elim*
6	y	2, $\wedge$ *elim*
7	$y \supset t$	1, $\wedge$ *elim*
8	t	3 and 4, $\supset$ *elim*
9	$z \wedge t$	8 and 5, $\wedge$ *intro*
10	$x \wedge y \supset z \wedge t$	2-9, $\supset$ *intro*
11	$(x \supset z) \wedge (y \supset t) \supset x \wedge y \supset z \wedge t$	1-10, $\supset$ *intro*

Each line contains a number, a formula and the justification of the formula. For example, at line 5, "z" is the consequence of applying $\supset$ *elim* with lines 3 and 4. A special symbol *hyp* is used when introducing an hypothesis. Additionally, a vertical bar is used to denote a *subordinate proof*. Finally discharging an assumption is represented by a justification taking a subordinate proof whose first line is an *hyp*. Lines 10 and 11 are examples of such a phenomenon. Note that assumptions, once allocated a line number, need not be repeated.

The linear style gives obviously a more vertical presentation than the tree style. But because of constant references to earlier lines, such proofs are reminiscent of assembly language programs and reading them is just as tedious.

2.3 Natural Deduction in Functional Format

The last format we envision is used in systems like [Coq91, Lego92, Nuprl86]. In these systems, proving a proposition means constructing a λ-term of a given type. In this section, we just explain intuitively why typed λ-terms represent natural deduction proofs. Complete introductions can be found in [Huet90, Mart70, Thomp91].

A natural functional reading of the rules of Figure 1 can be given in term of functions over proofs. For example, the $\wedge$*introduction* can be reformulated as a function that takes as arguments a proof of A and a proof of B and returns a proof of $A \wedge B$. Moreover the discharge mechanism is an exact equivalent of the λ notation. Combining both remarks, one goes one step further and notices that $\lambda h.\, M$ also represents a function over proofs. For example the identity function $\lambda h.\, h$, if h is a proof of A, returns a proof of A namely h. The isomorphism of Curry Howard simply says that a proof of $A \supset B$ corresponds exactly to a function that takes a proof of A and returns a proof of B. Reading the ":" notation as *is a proof of*, the previous λ-term can be written:

$$(\lambda h\colon A.\, h)\colon A \supset A$$

A consequence is also that application corresponds to modus ponens:

$$\lambda h\colon A.\, \lambda i\colon A \supset B.\, (i\, h)\colon A \supset (A \supset B) \supset B$$

An identical phenomenon occurs for universal quantification. The proof of $\forall x\colon T.\, (P\, x)$ corresponds to a function that given any element e in T returns a proof of $P\, e$. So generalization and specialization are also captured by means of abstraction and application. As an example we have:

$$(\lambda A\colon \mathit{Prop}.\, \lambda h\colon A.\, h)\colon \forall A\colon \mathit{Prop}.\, A \supset A$$

where *Prop* is the type of propositions.

Using this idea, the following three functions represent the rules for $\wedge$:

$$\wedge\mathit{intro}\colon \forall A, B\colon \mathit{Prop}.\, A \supset B \supset (A \wedge B)$$
$$\wedge\mathit{elim}_1\colon \forall A, B\colon \mathit{Prop}.\, (A \wedge B) \supset A \qquad \wedge\mathit{elim}_2\colon \forall A, B\colon \mathit{Prop}.\, (A \wedge B) \supset B$$

and the λ-term below corresponds to the previous proof of *præclarum*:

$$\lambda h\colon (x \supset z) \wedge (y \supset t).\, \lambda i\colon (x \wedge y).$$
$$\quad \wedge\mathit{intro}\ z\ t\ (\wedge\mathit{elim}_1\ (x \supset z)\ (y \supset t)\ h\ (\wedge\mathit{elim}_1\ x\ y\ i))\ (\wedge\mathit{elim}_2\ (x \supset z)\ (y \supset t)\ h\ (\wedge\mathit{elim}_2\ x\ y\ i))$$

We see that the functional format is very compact, since formulas that can be reobtained by type-checking do not occur in the proof term. This makes it well suited for computers, but somewhat difficult to read for humans. Still, we no longer have to read assembly code, but a functional program.

3 Extracting Text from Λ-terms

A number of authors ([Chester76, Ep93, Felty88, Huang94]) have investigated the possibility of producing text out of formal proofs. We approach the problem in a somewhat different fashion, viewing text production as a form of program extraction from proofs and using techniques that are familiar in code generation. We do not produce very fluid natural language and we handle only the

logical structure of the proofs, in the spirit of [Gentzen69]. Our approach is applicable to any of the proof objects we have presented previously[1]. The functional proof objects give the most concise and elegant definition, and we will use them for the rest of this paper. To construct a transducer from λ-terms to text, we must address a number of problems.

Type information

Trying to obtain a textual presentation from the raw λ–term is *not* possible: we need additional type information. To see this, consider the tautology $A \supset A$. A proof of this is the identity function $\lambda h\colon A.\ h$. If we decorate this term with type information, we obtain:

$$(\lambda h\colon A.\ h_A)_{A \supset A}$$

and it becomes clear that the text of the proof needs to make reference at least in two places to type information that is not structurally part of the λ-term:

$$(\lambda h\colon A.\ h_{\underline{A}})_{\underline{A \supset A}} \quad \triangleright \quad \begin{array}{l} \text{Assume } A\ (h) \\ \quad \text{By } (h) \text{ we have } \underline{A} \\ \text{We have proved } \underline{A \supset A} \end{array}$$

Textual variants

In fact, the proof above presumes that A is a proposition. The complete proof, including this assumption, is $\lambda A\colon \mathit{Prop}.\ \lambda h\colon A.\ h$, where *Prop* is a distinguished type that represents propositions. So we allow ourselves to use the word "Assume" because the type of A is *Prop*:

$$(\lambda A\colon \mathit{Prop}_{\underline{\mathit{Type}}}.\ (\lambda h\colon A_{\underline{\mathit{Prop}}}.\ h_A)_{A \supset A})_{\forall A:\mathit{Prop}.\ A \supset A} \quad \triangleright \quad \begin{array}{l} \underline{\text{Let}}\ A\colon\ \mathit{Prop} \\ \underline{\text{Assume}}\ A\ (h) \\ \quad \text{By } (h) \text{ we have } A \\ \text{We have proved } A \supset A \\ \text{We have proved } \forall A\colon \mathit{Prop}.\ A \supset A \end{array}$$

while we use "Let" for the outer λ-abstraction because the type of *Prop* is *Type*, another predefined type. In other circumstances, we will want to leave the λ-notation as it is, when the λ-abstraction denotes an authentic function. For example the successor function:

$$(\lambda h\colon nat.\ h + 1)_{\mathbb{N} \to \mathbb{N}}$$

This discussion shows that we need to use fairly different textual variants for a given construct (such as the λ-abstraction) depending on the type of its arguments. Type expressions are another example. A most general (dependent) product will be written $\Pi x\colon P.\ Q$. If Q is of type *Prop*, we write more conventionally $\forall x\colon P.\ Q$. If the product is non-dependent, i.e. x does not occur free in Q, one notes $P \to Q$. When additionally P and Q are of type *Prop*, we prefer $P \supset Q$.

Consider now an application MN and examine the type of M. We understand $M_{\forall x:P.\,Q}\ N$ as a specialization, while $M_{P \supset Q}\ N$ is an instance of *modus ponens* and $M_{P \to Q}\ N$ is a function application. The typed λ-term is a very compact notation, but examining the types brings out the logical structure of the proof.

[1] As a matter of fact, our early experiments have been carried out with a system using a tree representation of proofs [Théry94].

To sum up the analysis above, we propose a first attempt at transduction rules. In these rules, τ is assumed of type *Prop* and metavariable l may denote a list of bound variables. Expressions that don't match any rule are left unchanged, but for the conventions on displaying types described in the previous section.

Rules for abstraction

$(\lambda l{:}\, A_{Type}.\, M)_\tau$	$\triangleright$	Let $l:\ A$ M We have proved τ
$(\lambda h{:}\, A_{Prop}.\, M)_\tau$	$\triangleright$	Assume $A\ \ (h)$ $\quad M$ We have proved τ
$(\lambda x{:}\, A_{Set}.\, M)_\tau$	$\triangleright$	Consider an arbitrary x in A $\quad M$ We have τ, since x is arbitrary

Rules for application

$(M_{\forall x:P.\,Q}\ N)_\tau$	$\triangleright$	M In particular τ
$(M_{P \supset Q}\ N)_\tau$	$\triangleright$	- N - M We deduce τ

For identifiers, we make a slight distinction between assumptions appearing in the proof term (metavariable h) and theorems (metavariable T) found in the context.

Rules for identifiers

h_τ	$\triangleright$	By h we have τ
T_τ	$\triangleright$	Using T we get τ

Examples:
With the rules above, $\lambda A, B{:}\, Prop.\, \lambda h{:}\, A.\, \lambda h_0{:}\, A \supset B.\, (h_0\, h)$, the proof of *modus ponens* is displayed as

Let $A, B{:}\, Prop$
Assume $A\ \ (h)$
  Assume $A \supset B\ \ (h_0)$
    -By h we have A
    -By h_0 we have $A \supset B$
    We deduce B
  We have proved $(A \supset B) \supset B$
We have proved $A \supset (A \supset B) \supset B$
We have proved $\forall A, B{:}\, Prop.\ A \supset (A \supset B) \supset B$

and the proof (S) $\lambda A, B, C{:}\, Prop.\, \lambda h{:}\, A \supset B \supset C.\, \lambda h_0{:}\, A \supset B.\, \lambda h_1{:}\, A.\, (h\, h_1 (h_0\, h_1))$ reads

Let A, B, C: *Prop*
Assume $A \supset B \supset C$ (h)
  Assume $A \supset B$ (h_0)
    Assume A (h_1)
      -By h_1 we have A
      -By h_0 we have $A \supset B$
     -We deduce B
      -By h_1 we have A
      -By h we have $A \supset B \supset C$
     -We deduce $B \supset C$
    We deduce C
   We have proved $A \supset C$
  We have proved $(A \supset B) \supset A \supset C$
We have proved $(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$
We have proved $\forall A, B, C$: *Prop*. $(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$

While the texts above are very clear, they are also painfully lengthy. One reason is immediately apparent. Assumptions, corresponding to abstractions, are introduced (and discharged) one by one. Indeed, if the outer λ-abstraction had been decomposed as a succession of elementary bindings, the result might be even more verbose.

Repeated constructs

To improve the density of the proof text, we should

- ignore inessential intermediate results
- reduce the drift toward the right margin of the page caused by repeated indentations.

An effective technique of achieving this is to rewrite the rules for abstraction and application in the case where a given rule is being applied repeatedly.

Rules for iterated abstraction

$(\lambda l^1\colon A^1_{Type}.\cdots\lambda l^k\colon A^k_{Type}.\ M)_\tau$	$\triangleright$	Let $l^1:\ A^1$ $\vdots$ Let $l^k:\ A^k$ M We have proved τ
$(\lambda h^1\colon A^1_{Prop}.\cdots\lambda h^k\colon A^k_{Prop}\ M)_\tau$	$\triangleright$	Assume A^1 (h^1) and ... and A^k (h^k) M We have proved τ
$(\lambda x^1\colon A^1_{Set}.\cdots\lambda x^k\colon A^k_{Set}\ M)_\tau$	$\triangleright$	Choose arbitrarily x^1 in $A^1, \ldots, x^k$ in A^k M Thus we have τ

Rules for iterated application

$(M_{\forall x^1:P^1.\ \cdots \forall x^k:P^k.\,Q}\ N^1 \cdots N^k)_\tau$	$\triangleright$	M In particular τ
		-N^1 $\vdots$ -N^k
$(M_{P^1 \supset \cdots \supset P^k \supset Q}\ N^1 \cdots N^k)_\tau$	$\triangleright$	-M We deduce τ

Remarks:

1. The second rule of abstraction uses the connective "and" in a non-commutative manner: due to the dependence between types, a later assumption might refer to an earlier one [Ranta94]. The third rule uses the comma in the same way.

2. The rules for iterated application are a refinement of the familiar rule for representing curried applications with less parentheses:

$$((\cdots(M_{P^1 \to \cdots P^k \to Q}\ N^1)\cdots)N^k) \quad \triangleright \quad (M\ N^1 \cdots N^k)$$

3. Due to the rule of repeated implications, subproofs will occur in a more natural order.

4. As stated, the rules look non-deterministic, like the rule for curried applications. In fact, we always use them maximally, so that the algorithm is perfectly deterministic.

5. The format of the rules involving implications uses a dash "-", to emphasize the argument structure in the deduction. Because our style is generally postfix, if we write this symbol at the beginning of a subproof, we might have inaesthetic sequences of dashes. To avoid this, we require the dash to be printed in front of the conclusion of a subproof. The corresponding subproof itself, appearing above, is indented slightly to the right.

References to assumptions

To alleviate the presentation further, we choose a shorter rule to refer to assumptions introduced in the proof:

$$h_\tau \quad \triangleright \quad \text{We have } h$$

Additionally, we make a special case for (iterated) applications where the operator is a variable f (be it a local hypothesis or a theorem) and for applications where all operands are variables:

Rules for iterated application to variables

$(M_{P^1 \supset \cdots \supset P^k \supset Q}\ f^1 \cdots f^k)_\tau$	$\triangleright$	M With $f^1, \cdots, f^k$ we deduce τ
$(f_{P^1 \supset \cdots \supset P^k \supset Q}\ f^1 \cdots f^k)_\tau$	$\triangleright$	Using f with $f^1, \cdots, f^k$ we deduce τ
$(f_{\forall x^1:P^1.\ \cdots \forall x^j:P^j.\,Q^1 \supset \cdots \supset Q^k \supset R}\ M^1 \cdots M^j\ f^1 \cdots f^k)_\tau$	$\triangleright$	Applying f with $f^1, \cdots, f^k$ we get τ

Rules for iterated application of a variable

$(f_{\forall x^1:P^1.\cdots\forall x^k:P^k.\,Q}\ N^1\cdots N^k)_\tau$	▷	Specializing f we get τ
$(f_{P^1\supset\cdots\supset P^k\supset Q}\ N^1\cdots N^k)_\tau$	▷	-N^1 ⋮ -N^k From f we deduce τ
$(f_{\forall x^1:P^1.\cdots\forall x^j:P^j.\,Q^1\supset\cdots\supset Q^k\supset R}\ M^1\cdots M^j\ N^1\cdots N^k)_\tau$	▷	-N^1 ⋮ -N^k Applying f we get τ

On our previous examples (*modus ponens* and S) we obtain more acceptable results:

Let A, B: *Prop*
Assume A (h) and $A \supset B$ (h_0)
  Using h_0 with h we deduce B
We have proved $A \supset (A \supset B) \supset B$
We have proved $\forall A, B$: *Prop*. $A \supset (A \supset B) \supset B$

Let A, B, C: *Prop*
Assume $A \supset B \supset C$ (h) and $A \supset B$ (h_0) and A (h_1)
  - We have h_1
  - Using $h0$ with h_1 we deduce B
  From h we deduce C
We have proved $(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$
We have proved $\forall A, B, C$: *Prop*. $(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$

Let us try a new proof (*resolution*):

$$\lambda U\text{: } \mathit{Type}.\ \lambda P, Q\text{: } U \to \mathit{Prop}.\ \lambda a\text{: } U.\ \lambda h\text{: } (P\,a).\ \lambda h_0\text{: } (\forall x\text{: } U.\,(P\,x) \supset (Q\,x)).\ (h_0\,a\,h)$$

Let U: *Type*
Let P, Q: $U \to$ *Prop*
Let a: U
Assume $(P\,a)$ (h) and $\forall x\text{: } U.\,(P\,x) \supset (Q\,x)$ (h_0)
  Applying h_0 with h we get $(Q\,a)$
We have proved $(Pa) \supset (\forall x\text{: } U.\,(P\,x) \supset (Q\,x)) \supset (Q\,a)$
We have proved $\forall U\text{: } Type.\ \forall P, Q\text{: } U \to Prop.\ \forall a\text{: } U.\,(P\,a) \supset (\forall x\text{: } U.\,(P\,x) \supset (Q\,x)) \supset (Q\,a)$

To progress further, we consider theorems related with the method used to define new symbols, including the usual logical connectives.

Introduction theorems

New symbols are defined, possibly inductively, with the help of introduction theorems. Our concern being the proof structure, the only introduction that interests us are those which create propositions.

Such introduction theorems are of the form:

$$\mathtt{C}intro : \forall x^1\colon A^1 \ldots \forall x^n\colon A^n.\, \Phi^1 \supset \cdots \supset \Phi^i \supset (\mathtt{C}\, u^1 \ldots u^k)$$

where the subterms u^j are bound variables. For example, the disjunction is defined by two introduction theorems:

$$\vee intro_l\colon \forall A, B\colon Prop.\, A \supset A \vee B$$
$$\vee intro_r\colon \forall A, B\colon Prop.\, B \supset A \vee B$$

and the transitive closure R^* by the theorems:

$$R^*_0\colon \forall U\colon Type.\, \forall R\colon (Relation\, U).\, \forall x\colon U.\, (R^*\ U\ R\ x\ x)$$
$$R^*_n\colon \forall U\colon Type.\, \forall R\colon (Relation\, U).\, \forall x, y, z\colon U.\, (R\ x\ y) \supset (R^*\ U\ R\ y\ z) \supset (R^*\ U\ R\ x\ z)$$

Consider now a proof involving such theorems:

$$\lambda A, B, C\colon Prop.\, \lambda h\colon A.\, (\vee intro_r\, B\, (A \vee C)\, (\vee intro_l\, A\, C\, h))$$

Let $A, B, C\colon Prop$
Assume A (h)
- Applying $\vee intro_l$ with h we get $A \vee C$
Applying $\vee intro_r$ we get $B \vee (A \vee C)$
We have proved $A \supset B \vee (A \vee C)$
We have proved $\forall A, B, C\colon Prop.\, A \supset B \vee (A \vee C)$

Phrasing the proof as using two theorems is quite pedantic, since these facts constitute the definition of $\vee$. To make this distinction more apparent, we propose the following presentation rules:

Rules for introduction theorems

	$(\mathtt{C}intro\, M^1 \cdots M^n\, N^1 \cdots N^i)_\tau$	$\triangleright$	-N^1 $\vdots$ -N^i So by definition of C we have τ
$i = 0$			
	$(\mathtt{C}intro\, M^1 \cdots M^n)_\tau$	$\triangleright$	By definition of C we have τ
$i = 1$			
	$(\mathtt{C}intro\, M^1 \cdots M^n N)_\tau$	$\triangleright$	N By definition of C we have τ

Remarks:

1. Just as we did in the case of applications, we can use textual variants when applying an introduction theorem to variables. Such rules are easy to define, and we leave them to the imagination of the reader.

2. It is an advantage not to see the names of the introduction theorems in the proof text, because we know of no good principle of naming for them.

3. We could simply say "By definition", rather than "By definition of C" since the symbol C is the leading operator of τ.

Let A, B, C: *Prop*
Assume A (h)
  From h and the definition of $\vee$ we have $A \vee C$
  By definition of $\vee$ we have $B \vee (A \vee C)$
We have proved $A \supset B \vee (A \vee C)$
We have proved $\forall A, B, C$: *Prop*. $A \supset B \vee (A \vee C)$

Elimination theorems

Elimination theorems express how one uses new symbols. Elimination theorems have the following forms, depending on whether the symbol $\mathtt{C}$ is used to create a proposition or not:

$$\mathtt{C}elim : \forall x^1\!: A^1 \ldots . \forall x^n\!: A^n . \Phi^1 \supset \cdots \supset \Phi^i \supset (\mathtt{C}\, u^1 \ldots u^k) \supset B$$
$$\mathtt{C}elim : \forall x^1\!: A^1 \ldots . \forall x^n\!: A^n . \Phi^1 \supset \cdots \supset \Phi^i \supset \forall x\!: (\mathtt{C}\, u^1 \ldots u^k) . B$$

The elimination of disjunction is an example of elimination of the first kind:

$$\vee elim\!: \forall A, B, P\!: \mathit{Prop}. (A \supset P) \supset (B \supset P) \supset (A \vee B) \supset P$$

and the principle of induction over the naturals is an example for the second kind:

$$\mathbb{N}elim\!: \forall P\!: \mathbb{N} \to \mathit{Prop}. (P\, 0) \supset (\forall n\!: \mathbb{N}. (P\, n) \supset (P\, (\mathrm{Suc}\, n))) \supset \forall n\!: \mathbb{N}. (P\, n)$$

Note that both i and k may be 0, as in the following:

$$\bot elim\!: \forall P\!: \mathit{Prop}. \bot \supset P$$

Consider now the following proof of the commutativity of disjunction:

$$\lambda A, B\!: \mathit{Prop}. \lambda h\!: A \vee B.$$
$$(\vee elim\ A\ B\ (B \vee A)\ (\lambda i : A.\ \vee intro_r\ B\ A\ i)\ (\lambda j : B.\ \vee intro_l\ B\ A\ j)\ h)$$

Let A, B : *Prop*
Assume $A \vee B$ (h)
  Assume A (i)
    From i and the definition of $\vee$, we have $B \vee A$
  -We have proved $A \supset B \vee A$
  Assume B (j)
    From j and the definition of $\vee$, we have $B \vee A$
  -We have proved $B \supset B \vee A$
  -We have h
  Applying $\vee elim$ we get $B \vee A$
We have proved $A \vee B \supset B \vee A$
We have proved $\forall A, B$: *Prop*. $A \vee B \supset B \vee A$

If we observe the text produced by our transduction for the elimination, we notice that the layout of the arguments in repeated applications has the unhappy consequence that the argument to eliminate appears *last*. But this argument is the one that drives the reasoning. It is much more appropriate to give it first, and then to announce the possible cases. So we propose the following rules for elimination theorems of the first kind:

Rules for elimination theorems

$(\mathtt{C}elim\ M^1 \cdots M^n\, N^1 \cdots N^i\, P)_\tau$	$\triangleright$	P Therefore by definition of $\mathtt{C}$, to prove τ we have i cases: Case$_1$: $\quad N^1$ $\vdots$ Case$_i$: $\quad N^i$ So we have τ
$i = 0$ $(\mathtt{C}elim\ M^1 \cdots M^n P)_\tau$	$\triangleright$	P, by definition of $\mathtt{C}$ there is a contradiction So we can assert τ
$i = 1$ $(\mathtt{C}elim\ M^1 \cdots M^n\, N\, P)_\tau$	$\triangleright$	P Therefore by definition of $\mathtt{C}$ to prove τ $\quad N$ So we have τ

With these rules, we obtain results that are longer, but more perspicuous:

Let $A, B : Prop$
Assume $A \vee B$ (h)
  We have h
  Therefore by definition of $\vee$ to prove $B \vee A$, we have two cases:
  Case$_1$:
    Assume A (i)
      From i and the definition of $\vee$, we have $B \vee A$
    We have proved $A \supset B \vee A$
  Case$_2$:
    Assume B (j)
      From j and the definition of $\vee$, we have $B \vee A$
    We have proved $B \supset B \vee A$
  So we have $B \vee A$
We have proved $A \vee B \supset B \vee A$
We have proved $\forall A, B{:}\, Prop.\ A \vee B \supset B \vee A$

Slightly different rules are necessary for elimination theorems of the second kind.

$(\mathtt{C}elim\ M^1 \cdots M^n\, N^1 \cdots N^i)_\tau$	$\triangleright$	By definition of $\mathtt{C}$, to prove τ we have i cases: Case$_1$: $\quad N^1$ $\vdots$ Case$_i$: $\quad N^i$ So we have τ
$i = 0$ $(\mathtt{C}elim\ M^1 \cdots M^n)_\tau$	$\triangleright$	$\mathtt{C}$ is empty, so τ
$i = 1$ $(\mathtt{C}elim\ M^1 \cdots M^n\, N)_\tau$	$\triangleright$	By definition of $\mathtt{C}$ to prove τ, $\quad N$ So we have τ

To illustrate these rules, we look at a little inductive proof in Peano arithmetic. Assume $\leq$ is defined inductively by the following introduction theorems :

$$\begin{array}{ll} \leq intro_l\colon & \forall n\colon \mathbb{N}.\, n \leq n \\ \leq intro_r\colon & \forall n, m\colon \mathbb{N}.\, n \leq m \supset n \leq (\mathrm{Suc}\, m) \end{array}$$

Here is a proof that $\forall m\colon \mathbb{N}.\, (0 \leq m)$:

$$(\mathbb{N}elim\ (\lambda x.\, 0 \leq x)\ (\leq intro_l\, 0)\ \ \lambda m\colon \mathbb{N}.\, \lambda h\colon (0 \leq m).\ (\leq\ intro_r\, 0\, m\, h)\ )$$

and the way it is layed out now, without any rule that is specific of $\mathbb{N}elim$:

By definition of $\mathbb{N}$ to prove $\forall n\colon \mathbb{N}.\, 0 \leq n$, we have two cases:
Case$_1$:
  By definition of $\leq$ we have $0 \leq 0$
Case$_2$:
  Let $m\colon \mathbb{N}$
  Assume $0 \leq m$ $\ (h)$
    From h and the definition of $\leq$, we have $0 \leq (\mathrm{Suc}\, m)$
  We have proved $0 \leq m \supset 0 \leq (\mathrm{Suc}\, m)$
  We have proved $\forall m\colon \mathbb{N}.\, 0 \leq m \supset 0 \leq (\mathrm{Suc}\, m)$
So we have $\forall n\colon \mathbb{N}.\, 0 \leq n$

Omitting the conclusion of a proof

The two proofs above show that the application of elimination theorems accounts elegantly for two forms of reasoning:

- the commutativity of $\vee$ is proved *by cases*,
- the theorem on $\leq$ uses a proof *by induction*.

If the proof is by cases, the constructor that is eliminated doesn't occur in the contexts of the subordinate cases; otherwise the proof is by induction. But in both situations, the subordinate proofs introduce a context that they will finally discharge. Being told explicitly about the discharge of this local context seems inessential. In both examples above, it seems we could omit statements such as "We have proved $A \supset B \vee A$" or "We have proved $\forall m\colon \mathbb{N}.\, 0 \leq m \supset 0 \leq (\mathrm{Suc}\, m)$" without jeopardizing the clarity of the text. We have to be careful, however, not to lose the line of reasoning. When arguing by induction, we feel it useful to recall the goal of each subordinate case.

To specify this, let us first define a special (transducer) context called $*$ where the transduction *forgets* the conclusion of a series of λ-abstractions:

		Let $l^1\colon A^1$
		$\vdots$
		Let $l^k\colon A^k$
$(\lambda l^1\colon A^1_{Type}.\cdots\lambda l^k\colon A^k_{Type}.\, M)^*$	$\triangleright$	M^*
		Assume A^1 $\ (h^1)$ and $\ldots$ and A^k $\ (h^k)$
$(\lambda h^1\colon A^1_{Prop}.\cdots\lambda h^k\colon A^k_{Prop}\, M)^*$	$\triangleright$	M^*
		Choose arbitrarily x^1 in A^1, $\ldots$, x^k in A^k
$(\lambda x^1\colon A^1_{Set}.\cdots\lambda x^k\colon A^k_{Set}\, M)^*$	$\triangleright$	M^*
M^*	$\triangleright$	M

The last rule corresponds to the default behavior. If no $*$ rule is applicable, then one reverts to the other rules. Note that the $*$ rules are recursive. With the $*$ context, it is then possible to give a reformulation of the elimination rule in the propositional case:

		P Therefore by definition of C, to prove τ we have i cases: Case$_1$: we will prove τ_1 $N^{1\,*}$ $\vdots$ Case$_i$: we will prove τ_i $N^{i\,*}$ So we have τ
Induction Reasoning: $(\mathtt{C}elim\, M^1 \cdots M^n\, N^1_{\tau_1} \cdots N^i_{\tau_i}\, P)_\tau$	$\triangleright$	
		P Therefore by definition of C, to prove τ we have i cases: Case$_1$: $N^{1\,*}$ $\vdots$ Case$_i$: $N^{i\,*}$ So we have τ
Case Reasoning: $(\mathtt{C}elim\, M^1 \cdots M^n\, N^1 \cdots N^i\, P)_\tau$	$\triangleright$	

Similar rules can be given in the non propositional case. As a bonus, we can use the $*$ context at the top level of the proof object and announce the statement of the theorem to prove. Revisiting our examples, we get improved explanations:

Theorem: $\forall A, B\colon Prop.\ A \vee B \supset B \vee A$
Let $A, B : Prop$
Assume $A \vee B$ (h)
We have h
Therefore by definition of $\vee$ to prove $B \vee A$, we have two cases:
Case$_1$:
  Assume A (i)
  From i and the definition of $\vee$, we have $B \vee A$
Case$_2$:
  Assume B (j)
  From j and the definition of $\vee$, we have $B \vee A$
So we have $B \vee A$

Theorem: $\forall n\colon \mathbb{N}.\ 0 \leq n$
By definition of $\mathbb{N}$ to prove $\forall n\colon \mathbb{N}.\ 0 \leq n$, we have two cases:
Case$_1$: we will prove $0 \leq 0$
  By definition of $\leq$ we have $0 \leq 0$
Case$_2$: we will prove $\forall m\colon \mathbb{N}.\ 0 \leq m \supset 0 \leq (\mathrm{Suc}\, m)$
  Let $m\colon \mathbb{N}$
  Assume $0 \leq m$ (h)
  From h and the definition of $\leq$, we have $0 \leq (\mathrm{Suc}\, m)$
So we have $\forall n\colon \mathbb{N}.\ 0 \leq n$

Remarks:

1. Cases may be imbedded within cases. It is preferable to number cases absolutely, within a given proof, using a Dewey notation like "Case 1.2.1:".

2. Some axioms or theorems are of such frequent and quasi-implicit use that they do deserve specific linguistic wording. This is the case for theorems concerning usual logical constants, or the theorems used when reasoning classically, such as the axiom of the Excluded Middle. In fact, it is useful, for any given theory, to be able to associate specific wording and layout to any axiom or theorem. So, as an example we show in Fig. 2 and Fig. 3 rules we propose for the logical connectives. Notice that some rules demand their arguments to be λ-abstractions. However, the applicability of these rules is not reduced if we accept the η rule: $M \equiv \lambda x.\, M\,x$ (x does not appear free in M).

4 Further optimizations

Looking at the text obtained for the commutativity of $\vee$, we see that one aspect is less than idiomatic: we often give a name to an assumption, like h, i or j, although this assumption is used only once, and immediately. It is easy to check, on the body of a λ-term, that an assumption occurs only once. In that case, we can omit the name of the assumption and replace it, at its unique occurrence, by "the assumption". It is bit more tricky to check that its use occurs *immediately after* the assumption, because we have to track that fact through all of our rules. Assume we have been able to establish that an occurrence of h is unique and appears immediately after the introduction of h. Then we may drop the reference to h entirely. Using also the rules of Fig.2, the proof of commutativity of $\vee$ looks now like this:

Theorem: $\forall A, B\colon \mathit{Prop}.\ A \vee B \supset B \vee A$
Let $A, B : \mathit{Prop}$
Assume $A \vee B$
So we have two cases:
Case$_1$:
  Suppose A
  Obviously $B \vee A$
Case$_2$:
  Suppose B
  Obviously $B \vee A$
We have $B \vee A$ in both cases

The text for $(\lambda h\colon A.\, h)_\tau$ (the I combinator) becomes now too terse, so we write "Trivially τ". And for $(\lambda h\colon A.\, M)_\tau$ where h doesn't occur in M (the K combinator) we prefer:

M
A fortiori τ

A λ-term where x occurs only once and has the form;

$$\lambda x\colon (\mathtt{C}\, u^1 \ldots u^k).\, (\mathtt{C}elim\, M^1 \cdots M^n\, N^1 \cdots N^i\, x)$$

also deserves special treatment. Other possibilities, based on an analysis of dependence are approached in [Coscoy94]. Figure 4 shows the kind of results one obtains.

5 Conclusion

The rules discussed in this paper have, to a large extent, been implemented ([Massol93, Coscoy94]) as a back end for the Coq system [Coq91]. The implementation is part of the freely available user-interface package [CtCoq94]. User reaction confirms the usefulness of proof texts and shows that our approach remains practical for moderate size proofs. Clearly though, much work remains to obtain proof texts that are both compact and perspicuous. Most attractive is the fact that the system takes automatically into account the user's own inductive definitions. The use of normal definitions is not even recorded in the Coq (V5.8) proof object, so some proofs look mysteriously quick. As a consequence, one may develop a style of axiomatisation where inductive definitions are prevalent.

The implementation works in two parts. First, a program implemented within the Coq proof engine traverses the proof, recognizes patterns and equality of some subterms, checks occurrences of bound variables, decides what type annotations will appear in the proof text and computes Dewey numbers for all cases. Then, an annotated proof term is produced as input for a rule-driven pretty-printer. This tool applies the rules given in this paper, taking into account font selection and the limited width of the page. For large proofs, the computation of the first part will have to be optimized.

The exact wording proposed by our transduction rules may not be to everyone's taste. For example, instead of "Assume A", one may prefer "Assume A holds". This is very easy to change, and doesn't alter the basic argument of this paper: with a careful analysis of types, and an understanding of the role of specific theorems, it is possible to extract a good text from a typed λ-term. We have presented the basic elements of this analysis, but it is likely that it can be refined much further. We have left object-language formulae alone, but the work of [Ranta94] shows that much can be done in this area. Indeed a first area where this would be pleasant is for expressions of type $Type$, that may be rendered as substantives ([deBruijn87]). For other expressions, in the context of proof assistants, it seems more important to be able to introduce specialized mathematical notation for a given theory. We plan to implement a mechanism that loads pretty-printing rules for a given theory when the proof engine adds this theory to the current context. Beautifully, type theory allows one to load specific rules for the basic theorems of that theory with exactly the same mechanism.

Should the proof still be to verbose, it is possible to ask interactively for eliding subterms. We plan to keep a connection with the proof script, which might help in automating elision. But we will not try to discover ways to make the proof more abstract. In our view, it is the responsability of the user to introduce the right mathematical abstractions in the proof development, and if the user is unhappy with the proof text in that respect, then the proof itself should be improved.

Finally, users often ask whether such texts could also be used as input. We have not considered the issue, although we try to use distinctive phrases. The idea seems natural but non trivial. However, we have positive experience ([Théry94, Alf93]) with using partial proof text rather than sequents to represent the current state of the proof assistant. Such an approach introduces an entirely different concern in the production of text: from one step to the next, unless backtracking is specifically required, the text should grow monotonically for the user to feel comfortable. This precludes certain global optimizations, which may then be applied only when the proof (or a subordinate proof) is complete.

As a final remark, note that λ-terms are also used in denotational semantics ([Gordon79]). The ideas in this paper could serve in producing a pseudo natural language paraphrase of definitions that are hard to decipher, in view of the systematic use of continuations.

References

[Alf93] L. Magnusson, B. Nordström, *The ALF proof editor and its proof engine*, in Proceedings of the 1993 Types Worshop, Nijmegen, LNCS 806, 1994.

[Chester76] D. Chester, *The translation of Formal Proofs into English*, Artificial Intelligence 7 (1976), 261-278, 1976.

[Cohn88] A. Cohn, *Proof Accounts in HOL*, unpublished draft (Avra.Cohn@cl.cam.ac.uk).

[Coscoy94] Y. Coscoy, *Traductions de preuves en langage naturel pour le systeme Coq*, Rapport de Stage, Ecole Polytechnique, 1994.

[Coq91] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, B. Werner, *The Coq Proof Assistant User's Guide*, INRIA Technical Report no. 134, 1991.

[CtCoq94] Y. Bertot, *the CtCoq Interface*, available by ftp at babar.inria.fr:/pub/centaur/ctcoq.

[deBruijn87] N.G. de Bruijn, *The Mathematical Vernacular, a language for Mathematics with typed sets* in Proceedings from the Workshop on Programming Logic, P. Dybjer *et al.*, Programming Methodology Group, Volume 37, University of Göteborg, 1987.

[Ep93] A. Edgar, F.J. Pelletier, *Natural language explanation of Natural Deduction proofs*, in Proceedings of the First Conference of the Pacific Association for Computational Linguistics, Simon Fraser University, 1993.

[Euo92] H. Sawamura, T. Minami, K. Ohashi, *Proof Methods based on Sheet of Thought*, Research Report IIAS-RR-92-6E, Fujitsu Laboratories Ltd., Shizuoka, 1992.

[Felty88] A. Felty, *Proof explanation and revision*, Technical Report, University of Pennsylvania MS-CIS-88-17, 1988.

[Fitch52] F.B. Fitch, *Symbolic Logic: an introduction*, Ronald Press Company, 1952.

[Gentzen69] G. Gentzen, *Investigations into logical deduction*, in "The collected papers of Gerhard Gentzen", M.E. Szabo (Ed.), pp. 69-131, North Holland, 1969.

[Gordon79] M.J.C. Gordon, *The denotational description of programming languages: an introduction*, Springer-Verlag, 1979.

[Hol92] M.J.C. Gordon, T.F. Melham, *HOL: a proof generating system for higher-order logic*, Cambridge University Press, 1992.

[Huang94] X. Huang, *Reconstructing Proofs at the Assertion Level*, in Proceedings of CADE-12, Nancy, LNAI 814, Springer Verlag, 1994.

[Huet90] G. Huet, *A uniform approach to type theory*, Logical Foundations of Functional Programming, Addison-Wesley 1990.

[Kal80] D. Kalish, R. Montague, G. Mar, *Logic : techniques of formal reasoning*, Harcourt Brace and Company, 1980.

[Lego92] Z. Luo, R. Pollack, *LEGO proof development system : user's manual*, Technical Report, University of Edinburgh, 1992.

[Mart70] P. Martin-Löf, *A theory of types*, Technical Report, Departement of Mathematics, University of Stockholm, 1970.

[Massol93] A. Massol, *Présentation de Preuves en Langue Naturelle pour le Système Coq*, Rapport de DEA, Université de Nice-Sophia-Antipolis, 1993.

[Mur91] C.B. Jones, K.D. Jones, P.A. Lindsay, R. Moore, *MURAL: A Formal Development Support System*, Springer-Verlag, 1991.

[Nuprl86] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, J.T. Smith, *Implementing Mathematics with the Nuprl Proof Development System* Prentice-Hall, 1986.

[Prawitz65] D. Prawitz, *Natural Deduction, a proof-theoretical study*, Almqvist & Wiksell, 1965.

[Ranta94] A. Ranta, *Type Theory and the Informal Language of Mathematics*, in Proceedings of the 1993 Types Worshop, Nijmegen, LNCS 806, 1994.

[Théry94] L. Théry, *Une méthode distribuée de création d'interfaces et ses applications aux démonstrateurs de théorèmes*, PhD, Université Denis Diderot, Paris, 1994.

[Thomp91] S. Thompson, *Type Theory and Functional Programming*, Addison-Wesley, 1991.

[Tps92] P.B. Andrews, S. Issar, D. Newmith, F. Pfenning, *The TPS Theorem Proving System*, Journal of Symbolic Logic 57 (1992), 353-354, 1992.

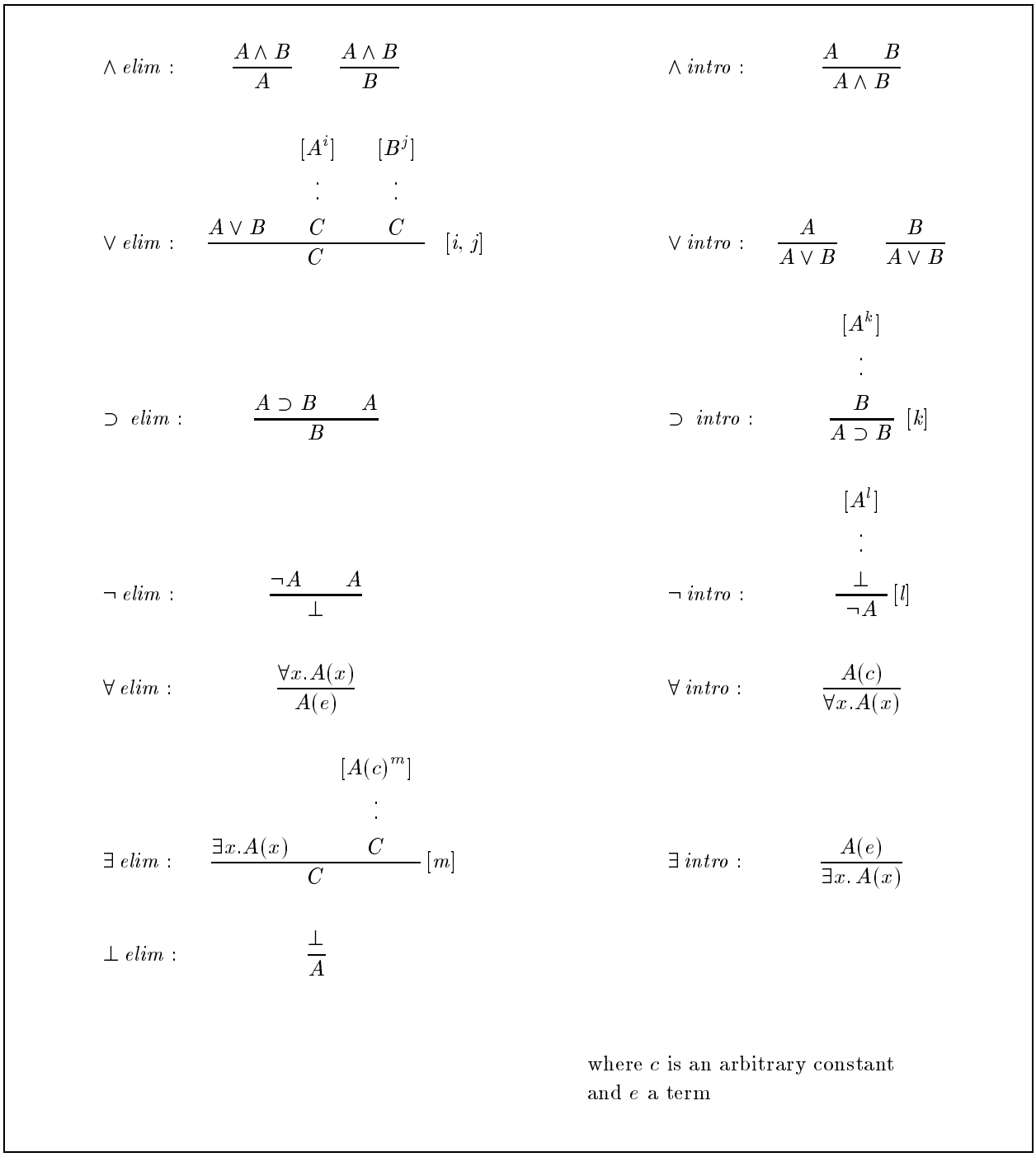

Figure 1: Natural deduction rules for intuitionistic first order logic

$\underline{\wedge intro}$: $\forall A, B$: $Prop.\ A \supset B \supset A \wedge B$

$(\wedge intro\ A\ B\ P\ Q)_\tau$ ▷
- P
- Q

Altogether we have τ

$\underline{\vee intro_l}$: $\forall A, B$: $Prop.\ A \supset A \vee B$

$(\vee intro_l\ A\ B\ P)_\tau$ ▷
P

Obviously τ

$\underline{\vee intro_r}$: $\forall A, B$: $Prop.\ B \supset A \vee B$

$(\vee intro_r\ A\ B\ P)_\tau$ ▷
P

Obviously τ

$\underline{\exists intro}$: $\forall A$: $Set.\forall P$: $A \rightarrow Prop.\forall y$: $A.\ (P y) \supset \exists x$: $A.\ (P\ x)$

$(\exists intro\ A\ P\ c\ Q)_\tau$ ▷
Q

The element c proves τ

$\underline{\neg intro}$: $\forall A$: $Prop.\ (A \supset \bot) \supset \neg A$

$(\neg intro\ A\ (\lambda i$: $A.P))_\tau$ ▷
Suppose $A\ (i)$

P

So we have $\neg A$ (negating i)

$\underline{\wedge elim}$: $\forall A, B, C$: $Prop.\ (A \supset B \supset C) \supset (A \wedge B) \supset C$

$(\wedge elim\ A\ B\ C\ (\lambda i$: $A.\ \lambda j$: $B.\ P)\ Q)_\tau$ ▷
Q

We know A (i) and B (j)

P

$\underline{\vee elim}$: $\forall A, B, C$: $Prop.\ (A \supset C) \supset (B \supset C) \supset (A \vee B) \supset C$

$(\vee elim\ A\ B\ C\ (\lambda i$: $A.\ P)\ (\lambda j$: $B.\ Q)\ R)_\tau$ ▷
R

So we have two cases

Case$_1$:

Suppose A (i)

P

Case$_2$:

Suppose B (j)

Q

We have τ in both cases

Figure 2: Text for familiar logical connectives

$\underline{\exists elim}: \forall A\colon Set.\, \forall P\colon A \to Prop. \forall C\colon Prop.\, (\forall y\colon A.\, (P\, y) \supset C) \supset (\exists x\colon (P\, x)) \supset C$

$(\exists elim\ A\ P\ (\lambda y\colon A.\, \lambda i : T.\, Q)\ R)_\tau$ $\triangleright$

R
So there exists a y in A such that T (i)
Q
We have τ (independently of y)

$\underline{\neg elim}: \forall A\colon Prop.\, A \supset \neg A \supset \bot$

$(\neg elim\ A\ P\ Q)_\tau$ $\triangleright$

- P
- Q
We have a contradiction

$\underline{\bot elim}: \forall P\colon Prop.\, \bot \supset P$

$(\bot elim\ P\ Q)_\tau$ $\triangleright$

Q
So we can assert τ

Classical Reasoning:

$\underline{ExMiddle}: \forall A\colon Prop.\, A \vee \neg A$

$(\vee elim\ A\ \neg A\ C\ (\lambda i\colon A.\, P)\ (\lambda j\colon \neg A.\, Q)\ (ExMiddle\ A))_\tau$ $\triangleright$

If we have A (i)
P
Otherwise $\neg A$ (j)
Q
So (classically) τ

$\underline{Contrad}: \forall A\colon Prop.\, (\neg A \supset \bot) \supset A$

$(Contrad\ A\ (\lambda i\colon A.P))_\tau$ $\triangleright$

Suppose $\neg A$ (i)
P
So (classically) we have A (negating i)

$\underline{ClassicQuantif}: \forall A\colon Set.\, \forall P\colon A \to Prop. \neg(\forall x\colon A.\, (P\, x)) \supset (\exists x\colon A.\, \neg(P\, x))$

$(ClassicQuantif\ M)_\tau$ $\triangleright$

M
Therefore (classically) we have τ

Figure 3: Text for familiar logical connectives and classical reasoning

Example1:

Theorem: *præclarum*
Statement
$\forall x, y, z, t\colon Prop.\, (x \supset z) \wedge (y \supset t) \supset x \wedge y \supset z \wedge t$
Proof
Let x, y, z, t be propositions
Assume we know $x \supset z$ (i) and $y \supset t$ (j)
Assume we know x (k) and y (l)
- Using i with k we deduce z
- Using j with l we deduce t
Altogether we have $z \wedge t$

Example2:

Theorem: *Symmetric and transitive relation is nearly reflexive.*
Statement
$\forall A\colon Set.\, \forall R\colon A \to A \to Prop.\, (\forall x, y\colon A.\, (R\, x\, y) \supset (R\, y\, x)) \supset$
$(\forall x, y, z\colon A,\, (R\, x\, y) \supset (R\, y\, z) \supset (R\, x\, z)) \supset \forall x\colon A.\, (\exists y_0\colon A.\, (R\, x\, y_0)) \supset (R\, x\, x)$
Proof
Let A be a set
Let $R\colon A \to A \to Prop$
Assume $\forall x, y\colon A.\, (R\, x\, y) \supset (R\, y\, x)$ $(symmetry)$
and $\forall x, y, z\colon A.\, (R\, x\, y) \supset (R\, y\, z) \supset (R\, x\, z)$ $(transitivity)$
Consider an arbitrary x in A
Assume there exists an element y_0 in A such that $R\, x\, y_0$
Using *symmetry* we deduce $R\, y_0\, x$
Using *transitivity* we deduce $R\, x\, x$

Example3:

Theorem: *Drinker's Principle*
Statement
$\forall A\colon Set.\, \forall P\colon A \to Prop.\, \forall y\colon A. \exists x\colon A.\, (P\, x) \supset \forall z\colon A.\, (P\, z).$
Proof
Let A be a set
Let $P\colon A \to Prop$
Consider an arbitrary y in A
If we have $\forall z\colon A.\, (P\, z)$
A fortiori $(P\, y) \supset \forall z\colon A.\, (P\, z)$
The element y proves $\exists x\colon A.\, (P\, x) \supset \forall z\colon A.\, (P z)$
Otherwise $\neg(\forall z\colon A.\, (P\, z))$
Therefore (classically) we have $\exists z\colon A. \neg(P\, z)$
So there exists a t in A such that $\neg(P\, t)$ (i)
Assume $(P\, t)$
From i, we have a contradiction
So we can assert $\forall z\colon A.\, (P\, z)$
We have proved $(P\, t) \supset \forall z\colon A.\, (P\, z)$
The element t proves $\exists x\colon A.\, (P\, x) \supset \forall z\colon A.\, (P\, z)$
We have $\exists x\colon A.\, (P\, x) \supset \forall z\colon A.\, (P\, z)$ (independently of t)
So (classically) $\exists x\colon A.\, (P\, x) \supset \forall z\colon A.\, (P\, z)$

Figure 4: Three examples