A Logically Saturated Extension of $\bar{\lambda}\mu\tilde{\mu}$

Lionel Elie Mamane, Herman Geuvers, and James McKinna

Institute for Computing and Information Sciences
Radboud University Nijmegen
lionel@mamane.lu, herman@cs.ru.nl, james@cs.ru.nl

Abstract. This paper presents a proof language based on the work of Sacerdoti Coen [1,2], Kirchner [3] and Autexier [4] on $\bar{\lambda}\mu\tilde{\mu}$, a calculus introduced by Curien and Herbelin [5,6]. Just as $\bar{\lambda}\mu\tilde{\mu}$ preserves several proof structures that are identified by the λ-calculus, the proof language presented here aims to preserve as much proof structure as reasonable; we call that property being *logically saturated*. This leads to several advantages when the language is used as a generic exchange language for proofs, as well as for other uses.

We equip the calculus with a simple rendering in pseudo-natural language that aims to give people tools to read, understand and exchange terms of the language. We show how this rendering can, at the cost of some more complexity, be made to produce text that is more natural and idiomatic, or in the style of a declarative proof language like Isar or Mizar.

1 Introduction

Effective Mathematical Knowledge Management requires languages for the representation of proofs at a level that is aware of the logical reasoning without committing to the technical details of a proof representation in e.g. a proof assistant or a derivation system. We aim at a proof language that is general enough to capturase different notions of proofs and that captures the logical structure of a proof in detail; a language that differentiates distinct proofs, but identifies two texts that represent the same proof. Such a proof language can be used as a common ground for interchange between different systems, as a language to speak about proofs and transformations thereof (e.g. automatic proof enhancement, rendering into natural language, ...).

Another requirement of such a proof language would be to have a nice natural language-style pretty-printing; the latter transformation ideally being simple enough to be done in one's head, so that a term in that language be readable by itself for someone that knows the language. In this respect, the natural language transformation of $\bar{\lambda}\mu\tilde{\mu}$ in [1] is very attractive: the transformation is purely structural, and the term is read strictly from left to right. However, it does not satisfyingly treat the whole calculus and $\bar{\lambda}\mu\tilde{\mu}$ (as extended to predicate logic in Fellowship [3]) still identifies proofs we'd like to differentiate.

This paper presents a more discerning extension of $\bar{\lambda}\mu\tilde{\mu}$ and a basic rendering of it in pseudo-natural language. We show how the rendering can be enhanced to

J. Carette et al. (Eds.): Calculemus/MKM 2009, LNAI 5625, pp. 405–421, 2009.

produce text that is more pleasing to read, and sketch how $\bar{\lambda}\mu\tilde{\mu}$ can be translated into the input language of proof assistants. The language presented here covers only implicational logic and disjunction (with an extension to full propositional logic given in the appendix and in [7]) and only proofs where every step taken is an atomic step of reasoning. So, seen from the viewpoint of proof assistants, we only deal with proofs where no automation is used. Naturally, the language will be extended in future work to address these limitations.

1.1 What Is a Proof?

From a logician's point of view, a proof is a derivation in a formal system of rules. From a more general mathematician point of view, it is a text that convinces his peers, for example a text that convinces that if they would spend enough time, they would be able to produce a fully formal derivation.

With our view centred on proof assistants, we consider the user input to the proof assistant to be a good candidate for the right notion of proof. A good test for the suitability of a candidate proof format is thus how well it captures these "proof assistant proofs".

This notion of proof is both coarser and finer than logician's proofs:

- It is coarser, because it glosses over automation done by the proof assistant; if the automation procedure changes, and finds a different logician's proof of a step done by automation, we still consider it the same proof.
- It is finer, because it separates cases where the same logician's proof (e.g. a natural deduction derivation) is produced in different ways, e.g. by a top-down proof or by a bottom-up proof, or by a proof that is partially top-down and partially bottom-up.

1.2 Design

Differentiating Power. We have already mentioned that we want our proof language to distinguish texts that code for different proofs, but identify texts that code for the same proof. This naturally begs the question: when do two texts represent different proofs and when do they represent the same proof?

We want to preserve the *intentional* content of a proof, the story that is being told. For example, a proof that first establishes A, then B and from these two concludes C is not the same as a proof that first established B, then A and then concludes. So we want our language to distinguish the order in which things happen, and to distinguish a forward-style (bottom-up) proof from a backward-style (top-down) proof, and to distinguish these from proofs that are done partially forwards and partially backwards.

As we focus on the logical content of proofs, it seems natural that we identify texts that vary only by purely linguistic differences. For example, the proofs at the right differ only linguistically from the corresponding proof at the left:

case 1: A holds ...	either A ...
case 2: $\neg A$ holds ...	or $\neg A$ ...

H: A by B hence C	H: A by B thus C by H

But if the difference comes from application of a different reasoning step, a different deduction rule, then it is not the same proof and the language should distinguish them. For example:

we have $A \to C \wedge B$ in particular we have $A \to C$ we already established A in lemma 5 thus C	we have $A \to C \wedge B$ that is, we have $A \to C$ (x) and we have B (y) by lemma 5 and x, we conclude C

The left proof uses a projection (from $A \wedge B$, we deduce *only* A), while the right proof uses a full decomposition (from $A \wedge B$, we deduce *both* A and B), it is not the same deduction rule.

Saturated System. In the design of a proof language, one usually tries to make it *minimal* at the logical level: A deduction rule that can be derived from others is considered superfluous and is therefore removed. We however, aim for a language that is *saturated*: Any step that an author can reasonably see as an atomic step, as a rule of reasoning that his reader will not doubt, should be a rule of the language. Any deduction rule that a proof assistant, or a logic, can reasonably choose as part of its "minimal set" should be a rule of the language. We don't claim that our system is the final answer to the quest for a saturated system, but we think we have come a long way. It should be tested on concrete proof examples to see whether anything is missing. We now give a pointwise discussion of the use of a saturated language for proofs.

• When the language is used as a proof *interchange* language, it allows the language to be neutral towards the choices of primitives made by different proof assistants; the language then is not closer to any one specific family of proof assistants than to another. By its saturation, it is close to all of them. A prime example of this is the implementation of classical logic: Some proof assistants implement intuitionistic logic augmented with an axiom, e.g. excluded middle. Others, such as PVS, use the reasoning with multiple goals of sequent calculus, where proving any one of the goals finishes the proof. A proof in one style can be transformed into the other style mechanically, but this produces a *different* proof of the same proposition. A language that provides classical logic solely through the double negation rule cannot faithfully represent PVS proofs.

• It makes the language particularly well suited to talk about proof manipulations, e.g. an algorithm that transforms proofs from one system into another. Because the saturated language has the concepts and rules from both systems, the transformation can be expressed as a transformation of terms of that language.

• It gives a tool to study and characterise what kinds of proofs a system can handle, because these can be expressed as sublanguage of the saturated language.

• When the language is used as a proof *authoring* language, it has the advantage to present all choices in a uniform way, without arbitrary distinction between

which (in the user's intuition) atomic step is atomic for the system and which step is a lemma application. There is no reason the user should have to care about that distinction.

Taking all this together, imagine a user that wants to work in classical logic, but is used only to its expression as intuitionistic logic plus double negation law. He ploughs on with his proof, but his proof assistant keeps track of the alternative goals he could be proving instead of the goal he is thinking of, and informing him of that list in a side-window. The user keeps an eye on it, and notices that this other goal seems easier to prove at this point. He does so. He doesn't understand the proof he has written, but he asks the system to transform it into an intuitionistic logic plus excluded middle proof, and he has a proof he can read and understand. By being based on a saturated logical system, the proof assistant has made its user's life easier.

Naturally, the kernel of the proof assistant can still happen in a minimal system, interpreting the other rules as lemma applications. A next version of the proof assistant may actually use a different minimal set of rules, and no one will notice. This is a kind of "abstract datatype" approach to logic: One does not need to look into the choices the proof assistant has made; one is free to do the things the logic one works in allows, the system implementing the abstract signature maps some steps to atomic steps and some others to lemmas, but this is none of our concern.

Human Factor. The proof language should also cater for human use, which amounts to the following two criteria.

• **Understandability.** Expressions of the language should *mean* something to a reader, be understandable. This is ensured if the user knows a transformation to natural language that is simple enough that he can do it in his head, if every construct of the language has a clear semantics and maps to a concept or a rule that the reader recognises.

• **Flexibility.** The language should capture different notions of a human's natural language view of a rigorous proof.

2 $\bar{\lambda}\mu\tilde{\mu}$

The $\bar{\lambda}\mu\tilde{\mu}$ calculus, which covers implication logic, is made up of three interdependent syntactical categories, namely *terms*, *environments* and *commands*:

Syntax	Typing judgement
$v ::= x \mid \lambda x : T.v \mid \mu\alpha : T.c$	$\Gamma \vdash v : T \mid \Delta$
$e ::= \alpha \mid v \circ e \mid \tilde{\mu}x : T.c$	$\Gamma \mid e : T \vdash \Delta$
$c ::= \langle v \Vert e \rangle$	$c : (\Gamma \vdash \Delta)$

Its typing makes use of a *hypothesis context* (Γ, which is a set of declarations $\{x_1 : T_1, \ldots, x_n : T_n\}$ where the T_i are simple types and all x_i are different) and a *goal context* (Δ, which is a set of declarations $\{\alpha_1 : T_1, \ldots, \alpha_n : T_n\}$ where the

T_i are simple types and all α_i are different). The part between $\vdash$ and $\mid$ is the *stoup* and contains a distinguished nameless formula.

$$\Gamma, x : T \vdash x : T \mid \Delta \qquad \Gamma \mid \alpha : T \vdash \alpha : T, \Delta$$

$$\frac{c : (\Gamma \vdash \alpha : T, \Delta)}{\Gamma \vdash (\mu\alpha : T.c) : T \mid \Delta} \qquad \frac{c : (\Gamma, x : T \vdash \Delta)}{\Gamma \mid (\tilde{\mu}x : T.c) : T \vdash \Delta} \qquad \frac{\Gamma \vdash v : T \mid \Delta \qquad \Gamma \mid e : T \vdash \Delta}{\langle v \| e \rangle : (\Gamma \vdash \Delta)}$$

$$\frac{\Gamma \vdash v : T \mid \Delta \qquad \Gamma \mid e : T' \vdash \Delta}{\Gamma \mid v \circ e : T \to T' \vdash \Delta} \qquad \frac{\Gamma, x : T \vdash v : T' \mid \Delta}{\Gamma \vdash (\lambda x : T.v) : T \to T' \mid \Delta}$$

Intuitionistic Fragment. Intuitionistic logic is obtained by restricting the use of environment variables to only the most recently (innermost) bound one.

Definition 1. *α is said to be used intuitionistically in c iff it occurs only in positions where it is the most recently bound environment variable. Equivalently, no path from the μ that binds α to an occurrence of α traverses a μ.*

The three syntactical categories are to be understood as:

term v proves the sequent $\Gamma \vdash T, \Delta$. T is singled out as the thesis one is currently working on; switching is allowed, but is an explicit step: a μ captures the current thesis, gives it a name so that it can be referred back to later and goes to a "neutral" state where no formula is distinguished.
The natural language rendering of a term v thus naturally is some text that is a proof of the sequent that types v, with the type of v as focus (current thesis) at the beginning of the text.

environment e expects (consumes) a proof of T (a term v typed by $\Gamma \vdash v : T \mid \Delta$) and continues further with the proof of sequent $\Gamma, T \vdash \Delta$, using the v it has consumed. A $\tilde{\mu}$ is the dual of μ; it captures the consumed proof and gives it a name.
The natural language rendering of an environment thus naturally is a context; that is some text containing a placeholder, a hole, such that if the placeholder is filled in with a proof of $\Gamma \vdash T, \Delta$, then the result is a proof of $\Gamma \vdash \Delta$. Furthermore, in this paper, the placeholder will, in the spirit of [1], always be at the very beginning of the rendering.

command combination of a term and an environment (a provider and a consumer) typed by the same Γ, T and Δ into a "closed" whole proving the sequent $\Gamma \vdash \Delta$, which is the type of c. The type of commands does not have a stoup (no singled out formula).
The natural language rendering of the command $\langle v \| e \rangle$ thus naturally is the rendering of e with the placeholder filled in with the rendering of v. In this paper, this amounts to the concatenation of the rendering of v and the rendering of e.

Definition 2. *A command whose environment ends in α is said to* conclude α. *It* ultimately concludes α *if it concludes α or ends in a binder (i.e. $\tilde{\mu}x : T.c$) whose binding domain (i.e. c) ultimately concludes α.*

For example, $\langle v \| v' \circ \alpha \rangle$ concludes α, but $\langle v \| v' \circ \tilde{\mu}y : T. \langle v_0 \| v_1 \circ \alpha \rangle \rangle$ does not. The latter ultimately concludes α.

3 Basic Pseudo-natural Language Rendering

The purpose of this rendering is to be a purely depth-0 structural, left-to-right reading of $\bar{\lambda}\mu\tilde{\mu}$ expressions, that is faithful to the proof the expression codes for. It is the rendering of [1], extended to handle the whole calculus and not only the intuitionistic fragment. $\boxed{\hookrightarrow}$ is an increase in indentation level and $\boxed{\hookleftarrow}$ a decrease.

$$
\begin{aligned}
[\![x]\!] &:= \text{by } x & [\![\alpha]\!] &:= \boxed{\hookleftarrow} \text{ done proving } \alpha \\
[\![\lambda x : T.v]\!] &:= \text{assume } T\ (x)\ [\![v]\!] & [\![v \circ e]\!] &:= \text{and } [\![v]\!]\ [\![e]\!] \\
[\![\mu\alpha : T.c]\!] &:= \text{thesis } T\ (\alpha) & [\![\tilde{\mu}x : T.c]\!] &:= \text{we have proven } T\ (x) \\
& \quad \boxed{\hookrightarrow} [\![c]\!] & & \quad [\![c]\!] \\
[\![\langle v \| e \rangle]\!] &:= [\![v]\!]\ [\![e]\!]
\end{aligned}
$$

Example 1. This term

$$\lambda x_R : P \to R.\lambda x_P : Q \to S \to P.\lambda y_S : S.\lambda y_Q : Q.\mu\alpha : R. \\ \langle x_P \| y_Q \circ y_S \circ \tilde{\mu} y_P : P.\, \langle x_R \| y_P \circ \alpha \rangle \rangle$$

of type $(P \to R) \to (Q \to S \to P) \to S \to Q \to R$ renders as

assume $P \to R$ (x_R)
assume $Q \to S \to P$ (x_P)
assume S (y_S)
assume Q (y_Q)
thesis R (α)
  by x_P and by y_Q and by y_S
  we have proven P (y_P)
  by x_R and by y_P
done proving α

The following term of the same type:

$$\lambda x_R : P \to R.\lambda x_P : Q \to S \to P.\lambda y_S : S.\lambda y_Q : Q.\mu\alpha : R. \\ \left\langle x_R \middle\| \left(\mu\beta : P.\, \langle x_P \| y_Q \circ y_S \circ \beta \rangle\right) \circ \alpha \right\rangle$$

renders as

assume $P \to R$ (x_R)
assume $Q \to S \to P$ (x_P)
assume S (y_S)
assume Q (y_Q)
thesis R (α)
  by x_R and thesis P (β)
    by x_P and by y_Q and by y_S
  done proving β
done proving α

The previous term was a forward (bottom-up) proof, this is backward (top-down) proof. In this manner, $\bar{\lambda}\mu\tilde{\mu}$ allows to choose at every step whether it is done backwards or forwards.

Classical Logic. Sequent calculus handles classical logic by allowing a set of formulas on the right hand side of the $\vdash$. In terms of ordinary logical arguments, this means that one maintains a *set* of goals throughout the reasoning; concluding any one of these goals concludes the whole proof. In a $\bar{\lambda}\mu\tilde{\mu}$ term, there is one goal "in focus" (the one before the stoup) and the other ones are named by environment variables; we can switch to another goal by using its name. To show how our extension of the [1] rendering to classical logic works we give as an example a proof of Peirce's law, $((P \to Q) \to P) \to P$. This uses the "goal switching" facility.

$$\lambda x : (P \to Q) \to P.\mu\alpha : P.\, \langle x \| (\mu\beta : P \to Q.\, \langle \lambda y : P.\mu\gamma : Q.\, \langle y \| \alpha \rangle \| \beta \rangle) \circ \alpha \rangle$$

which renders as

assume $(P \to Q) \to P$ (x)
thesis: P (α)
  by x and thesis $P \to Q$ (β)
    assume P (y)
    thesis: Q (γ)
      by y
    done proving α
  done proving β
done proving α

The classical logic step is the first "done proving α". Intuitionistically, one would have to prove γ at this point, but we conclude α instead, which concludes the whole proof.

4 Enhanced Pseudo-natural Language

We present several enhancements to the basic transformation, that do not break faithfulness to the proof the expression embodies. In order to keep the presentation simple, we will not discuss the interaction between the enhancements explicitly, unless there is an interesting or problematic point.

Backwards Proofs. This enhancement, namely replacing "and thesis" by "the thesis is reduced to", was already proposed in [1].

$$[\![(\mu\alpha : T.c) \circ e]\!] := \text{the thesis is reduced to } T \qquad (\alpha)$$
$$\boxed{\hookrightarrow}\, [\![c]\!]\, [\![e]\!]$$

It makes the intent of backwards proofs much more clear. Read the previous example again while mentally doing the replacement.

Intuitionistic Logic. Here, we recognise when single-goal logic (*i.e.* intuitionistic logic, plus eventually a classical logic axiom) is used and adapt the rendering. This consists in omitting the "(α)" when rendering a $\mu\alpha : T.c$ when α is used intuitionistically in c, combined with these rules; ↵ is a line break.

$$\begin{aligned}
[\![\alpha]\!] \text{ when the innermost parent } \mu \text{ binds } \alpha \\
:= \boxed{\hookleftarrow} \text{ done} \\
[\![\mu\alpha : T.c]\!] \text{ when } c \text{ ultimately concludes } \alpha \\
:= \text{we have to prove } T\ (\alpha)\text{↵}\boxed{\hookrightarrow}[\![c]\!]
\end{aligned}$$

With this improvement, our natural language translation gives the same result as the one in [1] on single-goal (sub)proofs, and also handles multiple-goal proofs.

Announcing Thesis Changes. The basic rendering informs the reader that what has been proven was not what the reader thinks of as "the current thesis" only *at the end* of a subproof. We see that e.g. in "done proving β" or "we have proven $T\ (x)$". That is essentially inherited from a prefix depth-first left-right reading of $\bar{\lambda}\mu\tilde{\mu}$ terms. It enhances the readability of the proof if such changes are announced at the start of the corresponding subproof, rather than at the end. This is typically also required in the proof input language of proof assistants. There are essentially three situations where such a thesis change happens:

Switching to another goal in Δ, a thing that is implicit in the standard sequent calculus, but is made explicit by the stoup structure of $\bar{\lambda}\mu\tilde{\mu}$. This corresponds to the pattern $\mu\alpha : T.c$ where c does not ultimately conclude α (but, say $\beta : T'$). We want to announce the thesis $T'(\beta)$, however, in a pattern like $\mu\alpha : T.\langle v \| \tilde{\mu}x : T''.\langle v' \| c\rangle\rangle$ we want to delay the announcement until we are under the $\tilde{\mu}$ binder. As a solution, we use a subscript in the transformation to keep track of the thesis currently active in the natural language text.

$$\begin{aligned}
[\![\langle v \| e\rangle]\!]_\beta \text{ when } e \text{ does not conclude } \beta \text{ and concludes } \alpha \\
:= \text{we now consider thesis } \alpha \\
[\![v]\!]_\alpha [\![e]\!]_\alpha \\
[\![\mu\beta : T.c]\!]_\alpha := \text{thesis } T\ (\beta)\text{↵}\boxed{\hookrightarrow}[\![c]\!]_\beta
\end{aligned}$$

This rendering keeps implicit in the natural language text that the active thesis is the most recently introduced one.

A cut. If the root of the term of a command is a μ, then the basic rendering already makes the announcement; we just tweak the text a bit:

$$\begin{aligned}
[\![\langle \mu\alpha : T.c \| e\rangle]\!] := \text{we now prove } T\ (\alpha) \\
\boxed{\hookrightarrow}[\![c]\!] \\
[\![e]\!]
\end{aligned}$$

As to the pattern $\langle \lambda x : T.v \| e\rangle$, it is dismissed as "bad style": It is a proof that does a thesis change, but refuses to announce it; fixing it crosses the

line of showing a *better* proof than the one written, not the proof written. It is suggested to η-expand this term to $\langle \mu\alpha : T.\, \langle v \| \alpha \rangle \| e \rangle$, which can be done programmatically as part of a "proof enhancement" transformation. A similar thing happens with the pattern $v \circ e$, where the root of v is a recursive constructor other than μ (e.g. λ), with the same solution.

The pattern $\langle v \| v_1 \circ \ldots \circ v_n \circ \tilde{\mu}x : T.c \rangle$. We can describe the environment part more succinctly by writing $e(\tilde{\mu}x : T.c)$: it is an environment that finishes with $\tilde{\mu}x : T.c$ and $e(\cdot)$ is that environment with the $\tilde{\mu}$ removed and replaced by a placeholder $\cdot$. A rendering would be

$$\begin{aligned} [\![\cdot]\!] &:= \boxed{\hookleftarrow} \text{ done} \\ [\![\, \langle v \| e(\tilde{\mu}x : T.c) \rangle]\!] &:= \text{we now prove } T\ (x) \\ &\quad \boxed{\hookrightarrow}\, [\![v]\!]\, [\![e(\cdot)]\!] \\ &\quad [\![c]\!] \end{aligned}$$

It forms a critical pair with the "detect a cut" rule, resolved with

$$\begin{aligned} [\![\, \langle \mu\alpha : T.c \| \tilde{\mu}x : T.c' \rangle]\!] &:= \text{we now prove } T\ (x, \alpha) \\ &\quad \boxed{\hookrightarrow}\, [\![c]\!] \\ &\quad [\![c']\!] \\ [\![\, \langle \mu\alpha : T.c \| e(\tilde{\mu}x : T'.c') \rangle]\!] &:= \text{we now prove } T'\ (x) \\ &\quad \boxed{\hookrightarrow} \text{ we now prove } T\ (\alpha) \\ &\qquad \boxed{\hookrightarrow}\, [\![c]\!] \\ &\qquad [\![e(\cdot)]\!] \\ &\quad [\![c']\!] \end{aligned}$$

These rules catch occurrences of μ when its type is not the current thesis in the text; this allows to enhance the rendering of the other occurrences (those that capture the current thesis and give it a name):

$$\begin{aligned} [\![\mu\alpha : T.c]\!] &:= \text{left to prove: } T\ (\alpha) \\ &\quad \boxed{\hookrightarrow}\, [\![c]\!] \end{aligned}$$

Remark 1. The rules introduced here have the big disadvantage that their structural depth (the depth at which they have to look into an expression before deciding how to render its root) is unbounded. This can be fixed by changing the syntax a bit, so that in a command the terminal constructors of the environment are available at depth 1:

$$\begin{array}{lll} E ::= \cdot \mid v \circ E & & \text{all non-terminal environment constructors} \\ e \ ::= \alpha \mid \tilde{\mu}x : T.c & & \text{all terminal environment constructors} \\ c \ ::= \langle v | E | e \rangle & & \end{array}$$

The meaning of the new syntax command $\langle v | v_1 \circ \ldots \circ v_n \circ \cdot | e \rangle$ is just $\langle v \| v_1 \circ \ldots \circ v_n \circ e \rangle$ The typing rules can be adapted accordingly.

From Binary to *n*-ary. $\circ$ is a binary constructor, but one can recognise sequences of it and treat it as an n-ary constructor; this is here combined with controlling its interaction with term variables and μ more closely:

$$\begin{aligned}
[\![x_0 \circ x_1 \circ \cdots \circ x_n \circ e]\!] &:= \text{by } x_0, x_1, \ldots, x_{n-1} \text{ and } x_n \ [\![e]\!] \\
[\![\langle x \| x_0 \circ x_1 \circ \cdots \circ x_n \circ e \rangle]\!] &:= \text{by } x, x_0, x_1, \ldots, x_{n-1} \text{ and } x_n \ [\![e]\!] \\
[\![(\mu\alpha_0 : T_0.c_0) \circ \cdots \circ x_j \circ \ldots & \\
\cdots \circ (\mu\alpha_i : T_i.c_i) \circ \cdots \circ e]\!] &:= \text{the thesis is reduced to:} \\
&\quad \bullet\ T_0\ (\alpha_0) \\
&\quad\ \boxed{\hookrightarrow} [\![c_0]\!] \\
&\quad \ldots \\
&\quad \bullet \text{ assumption } x_j \\
&\quad \ldots \\
&\quad \bullet\ T_i\ (\alpha_i) \\
&\quad\ \boxed{\hookrightarrow} [\![c_i]\!] \\
&\quad \ldots \\
&\quad [\![e]\!]
\end{aligned}$$

5 Saturation

There is a variety of alternative ways to handle implication in the calculus in [6]. In our quest for a saturated calculus, we examine them all and give them a natural language rendering.

ι_2 The first, and the only one we decide to keep as is, is called ι_2 in [6], but this conflicts with another constructor with the same notation; we thus rename it to $\lambda\_ : A$, in line with the convention that _ is a special name for "do not bind":

$$v ::= \ldots \mid \lambda\_ : A.v \qquad\qquad \frac{\Gamma \vdash v : T' \mid \Delta}{\Gamma \vdash (\lambda\_ : T.v) : T \to T' \mid \Delta}$$

$$[\![\lambda\_ : T.v]\!] := \text{assumption } T \text{ in thesis is not necessary. } [\![v]\!]$$

If the transformation has access to typing information (e.g. because every expression is annotated with its type), then one can use:

$$[\![\lambda\_ : T.v]\!] := \text{it suffices to prove } t(v). \ [\![v]\!]$$

where $t(v)$ is the type of v. This introduces some redundancy if a μ follows immediately; this redundancy can be avoided with

$$[\![\lambda\_ : A.\mu\alpha : B.c]\!] := \text{it suffices to prove } B\ (\alpha)\hookleftarrow\boxed{\hookrightarrow} [\![c]\!]$$

ι_1 As to its companion ι_1,

$$v ::= \ldots \mid \iota_1(e) \qquad \frac{\Gamma \mid e : A \vdash \Delta}{\Gamma \vdash \iota_1(e) : A \to B \mid \Delta}$$

while the logical step performed is naturally and immediately accepted as admissible, it is not intuitively seen as one atomic step; it is more natural to decompose it into two steps, namely assuming A and erasing goal B. We thus introduce a "no binding" version of μ:

$$v ::= \ldots \mid \mu\_ : T.c \qquad \frac{c : (\Gamma \vdash \Delta)}{\Gamma \vdash (\mu\_ : T.c) : T \mid \Delta}$$

$$[\![\mu\_ : T.c]\!] := \text{we give up on the current thesis} \hookleftarrow \boxed{\hookrightarrow} [\![c]\!]$$

This $\mu\_ : T.c$ has a natural dual in a putative $\tilde{\mu}\_ : T.c$:

$$e ::= \ldots \mid \tilde{\mu}\_ : T.c \qquad \frac{c : (\Gamma \vdash \Delta)}{\Gamma \mid (\tilde{\mu}\_ : T.c) : T \vdash \Delta}$$

but while $\mu\_ : T.c$ fulfils a real role (e.g. in $\lambda x : T_x.\mu\_ : \bot.\langle x \| \beta \rangle$, the current goal $\bot$ really is not useful in the rest of the proof; it is thus not useful to bother to give it a fresh name α to never refer to it later), any instance of $\tilde{\mu}\_ : T.c$ shows that the proof contains a completely non useful part: it takes the effort to prove T, but then just throws that result away. $\langle x \| y \circ \tilde{\mu}\_ : T. \ldots \rangle$ would correspond to something like "by x and y, we have proven T, but don't use that fact in the rest of the proof ...".

Furthermore, ι_1 is an instance of a bigger problem in the context of the rest of the natural language translation: term constructors that syntactically recurse into the environment category do not fit well. We have not found a nice phrase that turns what follows (which, being the translation of an environment, consumes a proof) into something which provides a proof. The best we could do was a rather weak and unnatural "the other goals follow from $A \hookleftarrow [\![e]\!]$", which had to be combined with changing the translation for $\tilde{\mu}x : T.c$ to "we can now assume $T \hookleftarrow [\![c]\!]$", because e.g. in the expression $\iota_1(\tilde{\mu}x : A.c)$ of type $A \to B$, A has not been proven, but assumed, so "we have proven" does not fit anymore. It makes the translation of $\tilde{\mu}$ in other situations weaker, but not wrong:

$$\langle \mu\alpha : T. \ldots \| \tilde{\mu}x : T. \ldots \rangle$$

thesis T (α)
...
done proving α
we can now assume T (x)
...

A construct that has no good natural language rendering cannot be a natural proof step for vernacular proofs and is thus not necessary to form a saturated system.

$\boldsymbol{\lambda(x : A, \alpha : B).c}$ naturally feels like two steps and is advantageously replaced by $\lambda x : A.\mu\alpha : B.c$.

$$v ::= \ldots \mid \lambda(x : A, \alpha : B).e \qquad \frac{c : (\Gamma, x : A \vdash \alpha : B, \Delta)}{\Gamma \vdash (\lambda(x : A, \alpha : B).c) : A \to B \mid \Delta}$$

$\boldsymbol{\lambda\alpha : B}$ takes an environment as argument, which raises the problems already discussed.

$$v ::= \ldots \mid \lambda\alpha : B.e \qquad \frac{\Gamma \mid e : A \vdash \alpha : B, \Delta}{\Gamma \vdash (\lambda\alpha : B.e) : A \to B \mid \Delta}$$

$[\![\lambda\alpha : B.e]\!] :=$ we now prove B, which follows from A

6 Link to Proof Assistants

We here succinctly treat the transformation of intuitionistic-logic $\bar{\lambda}\mu\tilde{\mu}$ terms (not the extended calculus) to Isar proofs by way of an example. A transformation into Mizar is similar (except that Mizar can *only* do forward steps, no backward step). Also a transformation into PVS has been designed, but it is less direct. Both are not included here for lack of space. All these translations have been tested by evaluating them manually on a few examples and having the corresponding prover accept the result. It should be noted that not all $\bar{\lambda}\mu\tilde{\mu}$ terms can be mapped faithfully to an Isar proof: there are proof constructs in our saturated system that Isar cannot capture. We could syntactically single out the $\bar{\lambda}\mu\tilde{\mu}$ terms that map to an Isar proof, but we take a different approach by describing a transformation of $\bar{\lambda}\mu\tilde{\mu}$ terms. The terms that are invariant under this transformation are the ones corresponding to Isar proofs.
Replace any pattern in the left column by the one in the right column:

$\langle v \Vert \cdots \circ (\lambda x : T.v') \circ \ldots \rangle$	$\langle v \Vert \cdots \circ (\mu\alpha : T_\alpha.\, \langle \lambda x : T.v' \Vert \alpha \rangle) \circ \ldots \rangle$
$\langle \lambda x : T.v \Vert v_0 \circ \ldots \rangle$	$\langle \lambda x : T.v \Vert \tilde{\mu}y : T'.\, \langle x \Vert v_0 \circ \ldots \rangle \rangle$
$\langle v \Vert \cdots \circ (\mu\alpha : T_\alpha.c) \circ x \circ \ldots \rangle$	$\langle v \Vert \cdots \circ (\mu\alpha : T_\alpha.c) \circ (\mu\beta : T_\beta.\, \langle x \Vert \beta \rangle) \circ \ldots \rangle$
$\langle \mu\alpha : T_\alpha.c \Vert e \rangle$	$c\{\alpha := e\}$
$\lambda x : T.x'$	$\lambda x : T.\mu\alpha : T'.\, \langle x \Vert \alpha \rangle$

Most of these transformation rules are interesting by themselves and fall under "the pattern on the left is bad proof style", but others have their origins in idiosyncrasies of Isar.

We do not deal here with mapping Isar proofs to $\bar{\lambda}\mu\tilde{\mu}$, but we claim that any Isar proof that does not use automation can be mapped faithfully to $\bar{\lambda}\mu\tilde{\mu}$. The Isar commands not used by the transformation below are mostly either syntactic sugar or logically equivalent to a basic command that we do use, or are concerned with automation.

The transformation follows. The purpose of cl is to count the lambdas in a term; we don't spell out its definition. We use the alternate syntax of page 413 for clarity.

$$
\begin{aligned}
[\![\mu\alpha : T_\alpha.\langle v|E|\tilde{\mu}x : T_x.c\rangle]\!] &:= \texttt{have } x : T\ [\![\langle v|E|\alpha\rangle]\!] \\
&\quad\ [\![\mu\alpha : T_\alpha.c]\!] \\
[\![\mu\alpha : T.c]\!] &:= \texttt{show } T\ [\![c]\!] \\
[\![\lambda x : T.v]\!] &:= \texttt{assume } x : T\ [\![v]\!] \\
[\![\langle x|y_0 \circ \cdots \circ y_{n-1} \circ \cdot|\alpha\rangle]\!] &:= \texttt{by (rule mp, ...}(n \text{ times})\texttt{..., rule mp,} \\
&\qquad\qquad \texttt{fact } x\texttt{, fact } y_0\texttt{, ..., fact } y_{n-1}\texttt{)}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\langle x|y_0 \circ \cdots \circ y_{n-1} \circ v_0 \circ \cdots \circ v_{p-1} \circ \cdot|\alpha\rangle]\!] := \\
&\qquad \texttt{proof (rule mp, ...}(n+p \text{ times})\texttt{..., rule mp,} \\
&\qquad\qquad \texttt{fact } x\texttt{, fact } y_0\texttt{, ..., fact } y_{n-1}\texttt{)}\ \boxed{\hookrightarrow} \\
&\qquad\quad [\![v_0]\!] \\
&\qquad\quad \ldots \\
&\qquad\quad [\![v_{p-1}]\!]\ \boxed{\hookleftarrow} \\
&\qquad \texttt{qed} \\
&[\![\langle v|\ \cdot\ |\alpha\rangle]\!] := \\
&\qquad \texttt{proof (rule impI, ...}\mathrm{cl}(v) \text{ times}\texttt{..., rule impI)}\ \boxed{\hookrightarrow} \\
&\qquad\quad [\![v]\!]\ \boxed{\hookleftarrow} \\
&\qquad \texttt{qed}
\end{aligned}
$$

In the rule for $[\![\mu\alpha : T_\alpha.\langle v|E|\tilde{\mu}x : T_x.c\rangle]\!]$, $\langle v|E|\alpha\rangle$ is not well-typed (the α may be unbound or of the wrong type), but that doesn't matter, because the 'name' α is never used in the Isar output (it translates to `qed`). One can vary on this translation, producing different Isar proofs – that we claim have the same logical structure.

7 Disjunction

As an example to extension to propositional logic, we briefly treat disjunction; again these constructs are catalogued in [6]; the natural language rendering is ours:

$$
\begin{aligned}
v &::= \ldots \mid \iota_{1,2}(v) \mid [\alpha : A, \beta : B].c \mid \lambda_{1,2}\alpha : A.v \\
e &::= \ldots \mid [e, e]
\end{aligned}
$$

$$
\frac{\Gamma \vdash v : T \mid \Delta}{\Gamma \vdash \iota_1(v) : T \vee T' \mid \Delta} \qquad \frac{\Gamma \vdash v : T' \mid \Delta}{\Gamma \vdash \iota_2(v) : T \vee T' \mid \Delta}
$$

$$
\frac{c : (\Gamma \vdash \beta : T', \alpha : T, \Delta)}{\Gamma \vdash ([\alpha : T, \beta : T'].c) : T \vee T' \mid \Delta} \qquad \frac{\Gamma \mid e : T \vdash \Delta \qquad \Gamma \mid e' : T' \vdash \Delta}{\Gamma \mid [e, e'] : T \vee T' \vdash \Delta}
$$

$$
\frac{\Gamma \vdash v : T' \mid \alpha : T, \Delta}{\Gamma \vdash (\lambda_1\alpha : T.v) : T \vee T' \mid \Delta} \qquad \frac{\Gamma \vdash v : T \mid \alpha : T', \Delta}{\Gamma \vdash (\lambda_2\alpha : T'.v) : T \vee T' \mid \Delta}
$$

$[\![\iota_1(v)]\!]$:= it suffices to prove the left part of the disjunction $[\![v]\!]$

$[\![\iota_2(v)]\!]$:= it suffices to prove the right part of the disjunction $[\![v]\!]$

$[\![\lambda_{1,2}\alpha : A.v]\!]$:= keeping in mind that we may prove A (α), $[\![v]\!]$

$[\![[\alpha : A, \beta : B].c]\!]$:= thesis A (α) or B (β)
$\boxed{\hookrightarrow}[\![c]\!]$

$[\![[e, e']\!]$:= either
$\boxed{\hookrightarrow}[\![e]\!]$
or
$\boxed{\hookrightarrow}[\![e']\!]\boxed{\hookleftarrow}$

Many of the improvements of section 4 apply to disjunction mutadis mutandis. This is further detailed in [7], but here are a few examples:

- $[\alpha : A, \beta : B]$ is a μ-like construct; the definitions and extended rules that deal with μ should thus deal also with it, suitably adapted. For example, the "backwards proofs" enhancement:

  $[\![([\alpha : A, \beta : B].c) \circ e]\!]$:= the thesis is reduced to T (α) or T' (β)
  $\boxed{\hookrightarrow}[\![c]\!]\,[\![e]\!]$

  Similarly, the definition of "α is used intuitionistically in c" has to be changed to "no path from the μ that binds α to an occurrence of α traverses a μ or a $[\cdot, \cdot]$". In particular, if α is bound by a $[\cdot, \cdot]$, it is not used intuitionistically; $[\cdot, \cdot]$ is an inherently non-intuitionistic construct.
- $\lambda_{1,2}$, $\iota_{1,2}$ and $[e, e']$ benefit particularly strongly from typing information:

  $[\![\iota_{1,2}(v)]\!]$:= it suffices to prove $t(v)$
  $[\![v]\!]$

  $[\![\lambda_1\alpha : A.(v)]\!]$:= keeping in mind we may prove A (α),
  we proceed with the proof of $t(v)$

  $[\![[e, e']\!]$:= either $t(e)$
  $\boxed{\hookrightarrow}[\![e]\!]$
  or $t(e')$
  $\boxed{\hookrightarrow}[\![e']\!]\boxed{\hookleftarrow}$
- Just as sequences of $\circ$ can be collected in an emulation of n-ary implication, sequences of ι_1, ι_2 can be collected in an emulation of n-ary disjunction.

The introduction of $[\cdot, \cdot]$ has interesting consequences for the enhanced rendering: there is not anymore unicity of the terminal environment constructor (the first environment constructor that does not syntactically recurse back into category e). For example, $[\ldots\alpha, \ldots\tilde{\mu}x : T.c]$.

Definition 3. *If all terminal environment constructors of an environment e are the same (the* same α *or* $\tilde{\mu}x : T.c$ *with the* same x *and the* same T*), then e is said to terminate* uniformly *in that constructor. Similarly, e (ultimately)* uniformly *concludes* α *if all its branches (ultimately) conclude* α*.*

The adaptation to that situation is to then defer the decisions (on commands) that depend on the terminals of the environment, when these are not uniform. For example, the "announce thesis changes" enhancement:

$[\![[e, e']]\!]_\beta :=$ either $t(e)\boxed{\hookrightarrow}$
if e does not conclude β *and uniformly concludes* α
we now consider thesis α
$[\![e]\!]_\alpha$
else if e terminates uniformly in $\tilde{\mu}x : T.c$
we now prove T (x)
$\boxed{\hookrightarrow}[\![e(\cdot)]\!]_\beta$
$[\![c]\!]_\beta$
else
$[\![e]\!]_\beta$
end if
or $t(e')\boxed{\hookrightarrow}$
the same for e'
$\boxed{\hookleftarrow}$

Similarly, in the alternative syntax of remark 1 page 413, a command now needs to take a list of es whose length matches the number of $\cdot$s in the E.

8 Future Work

There are several directions in which this can be taken further. The most obvious is extending, with the same concern for saturation, to some predicate logic. The second glaring need is that the problem of capturing automation in theorem provers needs to be addressed for the language to be really functional in practise as a proof interchange language. A natural idea is to store as part of the term a witness provided by the automation, which would be used to produce a proof in a system whose automation is weaker. Alas, it may not be practical or possible to get a witness from some provers' automation (no access to its source code, prover not structured in De Bruijn criterion conformant way (that is the automation is not already forced to provide a witness to a kernel), ...).

Also, our calculus is not completely saturated in its treatment of classical logic; one can convincingly argue that e.g. De Morgan laws and classical decomposition of disjunction tend to be considered as valid atomic deduction steps in the vernacular; a saturated calculus should thus have constructs for them.

The proof of the pudding being in the eating, we will concretely implement the transformations from and to various proof languages (various proof assistants, but also e.g. a standard sequent calculus); this would find a natural place as part of a proof assistant.

On a more theoretical side, our natural language rendering has an underlying concept of structure of a natural language proof, which (when restricted to single-goal logic) is quite close to the structure of declarative proof languages like Mizar or Isar. But, in particular in its notion of active thesis, and when a change of it needs to be explicitly announced (i.e. always), it is not in complete agreement with the structure of proofs in $\mathrm{LK}_{\mu\tilde{\mu}}$ (sequent calculus with 'stoup', basically just $\bar{\lambda}\mu\tilde{\mu}$ without term/expression/command information). One would like changes in the active thesis to be characterised as either a cut, or an explicit active thesis change. We will thus define a sequent calculus whose notion of cut matches the notion of introducing an arbitrary new thesis (a forward step) in our natural language, and that matches the natural language's need for an explicit switch action between the theses. Restricting the hypothesis-creating children of a left introduction rule to a left introduction rule or an axiom may be the main thing needed to achieve the former. It would then be interesting to study proof intent conserving transformations between that calculus, $\mathrm{LK}_{\mu\tilde{\mu}}$, standard stoup-free sequent calculus and other proof formats.

References

1. Sacerdoti Coen, C.: Explanation in natural language of $\bar{\lambda}\mu\tilde{\mu}$ terms. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 234–249. Springer, Heidelberg (2006)
2. Sacerdoti Coen, C.: Declarative representation of proof terms. In: Prog. Lang. for Mechanized Math., University of Linz. RISC Reports, vol. 07-10, pp. 3–18 (2007)
3. Kirchner, F.: Interoperable proof systems. Ph.D thesis, École Polytechnique (2007)
4. Autexier, S., Sacerdoti Coen, C.: A formal correspondence between OMDoc with alternative proofs and the $\bar{\lambda}\mu\tilde{\mu}$-calculus. In: Borwein, J.M., Farmer, W.M. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 67–81. Springer, Heidelberg (2006)
5. Curien, P.L., Herbelin, H.: The duality of computation. In: ICFP 2000, vol. 35(9), pp. 233–243. ACM, New York (2000)
6. Herbelin, H.: C'est maintenant qu'on calcule; au cœur de la dualité. Habilitation à diriger des recherches, Université Paris 11 (December 2005)
7. Mamane, L.E., Geuvers, H., McKinna, J.: A logically saturated extension of $\bar{\lambda}\mu\tilde{\mu}$ to propositional logic, `http://www.mamane.lu/science/lbmmt_extension/`
8. Corbineau, P.: A declarative proof language for the Coq proof assistant. In: Miculan, M., Scagnetto, I., Honsell, F. (eds.) TYPES 2007. LNCS, vol. 4941, pp. 69–84. Springer, Heidelberg (2008)
9. Wenzel, M.: Isar - a generic interpretative approach to readable formal proof documents. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, p. 167. Springer, Heidelberg (1999)
10. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS (LNAI), vol. 607, pp. 748–752. Springer, Heidelberg (1992)
11. Trybulec, A.: Mizar language, `http://mizar.org/language/`
12. Asperti, A., Guidi, F., Padovani, L., Sacerdoti Coen, C., Schena, I.: Mathematical knowledge management in HELM. AMAI 38(1), 27–46 (2003)
13. Coscoy, Y.: Explication textuelles de preuves pour le calcul des constructions inductives. Ph.D thesis, Université de Nice-Sophia-Antipolis (September 2000)

A Basic Propositional Logic

$$v ::= \ldots \mid (v, v) \mid \lambda_{\neg} x : A.v \mid \ltimes \mid \mathrm{TE}(T) \mid \mathrm{DN}(v)$$
$$e ::= \ldots \mid \pi_{1,2}[e] \mid (x : A, y : B).c \mid \lambda_{1,2} x : A.e \mid \neg[v] \mid \rtimes \mid \mathrm{DN}(e)$$

$$\frac{\Gamma \vdash v : T \mid \Delta \qquad \Gamma \vdash v' : T' \mid \Delta}{\Gamma \vdash (v, v') : T \wedge T' \mid \Delta} \qquad \frac{c : (\Gamma, x : T, x' : T' \vdash \Delta)}{\Gamma \mid ((x : T, x' : T').c) : T \wedge T' \vdash \Delta}$$

$$\frac{\Gamma, x : T \mid e : T' \vdash \Delta}{\Gamma \mid (\lambda_1 x : T.e) : T \wedge T' \vdash \Delta} \qquad \frac{\Gamma, x : T' \mid e : T \vdash \Delta}{\Gamma \mid (\lambda_2 x : T'.e) : T \wedge T' \vdash \Delta}$$

$$\frac{\Gamma \mid e : T \vdash \Delta}{\Gamma \mid \pi_1[e] : T \wedge T' \vdash \Delta} \qquad \frac{\Gamma \mid e : T' \vdash \Delta}{\Gamma \mid \pi_2[e] : T \wedge T' \vdash \Delta} \qquad \frac{\Gamma, x : T \vdash v : \bot \mid \Delta}{\Gamma \vdash (\lambda_{\neg} x : T.v) : \neg T \mid \Delta}$$

$$\frac{}{\Gamma \vdash \ltimes : \top \mid \Delta} \qquad \frac{\Gamma \vdash v : T \mid \Delta}{\Gamma \mid \neg[v] : \neg T \vdash \Delta} \qquad \frac{}{\Gamma \mid \rtimes : \bot \vdash \Delta}$$

$$\frac{}{\Gamma \vdash \mathrm{TE}(P) : P \vee \neg P \mid \Delta} \qquad \frac{\Gamma \vdash v : \neg\neg T \mid \Delta}{\Gamma \vdash \mathrm{DN}(v) : T \mid \Delta} \qquad \frac{\Gamma \mid e : T \vdash \Delta}{\Gamma \mid \mathrm{DN}(e) : \neg\neg T \vdash \Delta}$$

$[\![(x : A, y : B).c]\!]$:= we have proven A (x) and B (y)
$[\![c]\!]$

$[\![\lambda_{1,2} x : T.e]\!]$:= we have proven T (x) and $[\![e]\!]$

$[\![\mathrm{DN}(v)]\!]$:= proof by contradiction

$[\![\mathrm{DN}(e)]\!]$:= and by double negation elimination

$[\![(v, v')]\!]$:= • $[\![v]\!]$
• $[\![v']\!]$

$[\![\pi_{1,2}[e]]\!]$:= in particular $[\![e]\!]$

$[\![[\alpha : A, \beta : B].c]\!]$:= thesis A (α) or B (β)
$\boxed{\hookrightarrow}$ $[\![c]\!]$

$[\![\neg[v]]\!]$:= and $[\![v]\!]$
$\boxed{\hookleftarrow}$ done (ECQ)

$[\![[e, e']]\!]$:= either
$\boxed{\hookrightarrow}$ $[\![e]\!]$
or
$\boxed{\hookrightarrow}$ $[\![e']\!]$ $\boxed{\hookleftarrow}$

$[\![\ltimes]\!]$:= true

$[\![\rtimes]\!]$:= $\boxed{\hookleftarrow}$ done (EFQ)

$[\![\lambda_{\neg} x : A.v]\!]$:= assume A (x) $[\![v]\!]$

$[\![\mathrm{TE}(T)]\!]$:= by TE

ECQ is an abbreviation for "ex contradictione (sequitur) quodlibet" (from a contradiction, anything follows), EFQ for "ex falso (sequitur) quodlibet" (from falsehood, anything follows) and TE for "(principium) tertii exclusi" (principle of excluded middle).

Again, some natural language rendering enhancements apply mutatis mutandis; others need to be adapted. For example, dually to what happens with disjunction, $(x : A, y : B).c$ is a $\tilde{\mu}$-like construct and the enhancements that deal with $\tilde{\mu}$ need to similarly deal with it. Also, the definition of "e concludes α" has to be changed so that *e.g.* $\rtimes$ and $\neg[v]$ conclude α for any α.