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
Heijendaalseweg 135, 6525 AJ Nijmegen, The Netherlands
lionel@mamane.lu, herman@cs.ru.nl and james@cs.ru.nl

Abstract This report 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. $\bar{\lambda}\mu\tilde{\mu}$ is a proof term language for sequent calculus proofs; the pseudo-natural language rendering thus contributes a way to talk about these proofs in natural language.

1 Introduction

Effective Mathematical Knowledge Management requires languages for representation of mathematical documents and objects and discourse within them. We here focus on proofs, at a level more aware of the logic reasoning than efforts such as OMDoc. From our focus on proofs, follows a focus the part of the MKM toolchain that deals with proofs, namely proof assistants. We aim at a proof language that is general enough to capture different notions of proofs, that captures the structure of a proof in detail; a language that differentiates texts that code for 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, ...), ...

Because validity of proofs is more closely tied to the precise semantics than the informal meaning of an expression, we find that the best level to address interchange of proofs is not the content level (using the terminology of [7]), but something in between the content level and the semantic level.

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 regard, 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 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 propositional logic and only proofs where every step taken is an atomic step of reasoning, thus when seen from the viewpoint of proof assistants, proof where no automation is used or the proof that is constructed by the automation, as opposed to the proof written by the user. Naturally, the language will be extended in future work to address those limitations.

1.1 What Is a Proof?

The traditional answer from a logician's point of view is that it is a derivation in a formal system of rules, in the form of a DAG / tree / lambda-term. From a more general mathematician point of view, it is a text that convinces his peers, for example a text that convinces them that the only thing needed to obtain a fully formal derivation is spending the time and effort, not mathematical intuition or creativity.

With our view centred on proof assistants, we consider the user input to the proof assistant, that which gets stored in the library of the proof assistant, to be a good candidate for the right notion of proof. Indeed, if the automation changes and finds a different "logician's proof" for the same input, the user won't consider that the proof he has written changed. 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.* λ-term) 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 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: Given two texts, when do they 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, and these proofs among themselves.

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	H: A by B
hence C	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 $A \to C \wedge B$	we have $A \to C \wedge B$
a fortiori we have $A \to C$	that is, we have $A \to C$ (x)
we already established A in lemma 5	and we have B (y)
thus C	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 This leads to a concept in opposition with the traditional design of proof languages, which is to make them *minimal* at the logical level: with the possible exception of the cut rule, remove any deduction rule from the system and it is not complete anymore. E.g. for conjunction, if you have both projections, you don't need full decomposition and vice-versa. We 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.

– 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 (excluded

middle, Peirce's law, double negation law, ...). Others, such as PVS, use the reasoning with multiple goals of sequent calculus, where proving any of the goals (not all of them) finishes the proof. A proof in one style can be transformed into the other style mechanically, but that is an infidelity producing *another* proof of the same proposition. A language that would, for example, make the choice of classical logic by double negation would naturally get into difficulties when trying to represent PVS proofs faithfully: It would have no natural way to distinguish a PVS proof p essentially making use of the multiple goals from the double-negation-using proof p' it maps p to, done in PVS, using a double-negation theorem in PVS. If it does not distinguish between those, one of the bad consequences is that transforming the proof from PVS into the interchange language and back into PVS will, for at least one of the proofs p or p', not produce the same proof again.
- It makes the language particularly well suited to talk about proof manipulations. For example, to present an algorithm to transform a proof from one system into another system, because it has the concepts and rules from both systems, it gives a way to talk about them and relate them. For example, an algorithm to transform a multiple-goal proof into a double-negation law proof can be expressed as a transformation on terms of the language.
- It gives a language in which to study and characterise what kinds of proofs what system can handle, expressed as sublanguage of the 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, let's 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 (that multiple goals stuff is a bit mysterious, that's not how logic works in his mind), 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, a proof that he can present to his colleagues (which, being of a similar background, wouldn't be convinced by a goal-switching proof). 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, the fundamental proof checking, can (e.g. for De Bruijn criterion reasons) still happen in a minimal system, interpreting the other rules as lemma applications. And the 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 what choices the proof assistant one uses 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.

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 semantic and maps to a concept or a rule that the reader recognises.

Flexibility The language should capture different proof assistants' notion of a proof, but also a human's natural language view of a rigorous proof.

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

2.1 Syntax

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

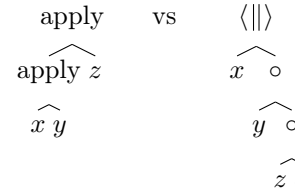
term an expression of the category v. A typing judgement for a term is of the form $\Gamma \vdash v : T \mid \Delta$. The $v : T$ part is called the *stoup*. By abuse of language, we say "v is of type T" when $\Gamma \vdash v : T \mid \Delta$ for some Γ, Δ that are clear from the context. The typing judgement should be understood as: 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. The top level of a term is where right introduction rules happen, where one works on the right side of the sequent.
A term variable will be denoted as $x, y, z, \ldots$

environment an expression of the category e. A typing judgement for it is of the form $\Gamma \mid e : T \vdash \Delta$. The conventions made for terms apply mutatis mutandis. The typing judgement should be understood as: 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. The top level of an environment is where left introduction rules happen, where one works on the hypotheses or a previously established proposition.
An environment variable will be denoted as $\alpha, \beta, \gamma, \delta, \ldots$

command an expression of the category $c ::= \langle v \| e \rangle$; it combines 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) and T does not necessarily appear in the typing of c; it is the construct that allows switching the current focus to another (related or not) goal. Switching to an unrelated goal is, in essence, a logical cut.

	syntax	type	natural language
$v ::=$	x	T	by x
	$\mid \lambda x{:}T.v$	$T \to T'$	assume T (x)↵ ...
	$\mid \mu\alpha{:}T.c$	T	thesis T (α)↵ ...
$e ::=$	α	T	done proving α
	$\mid v \circ e$	$T \to T'$	then ... ...
	$\mid \tilde{\mu}x{:}T.c$	T	we have proven T (x)↵ ...
$c ::=$	$\langle v \| e \rangle$		

Remark 1. Compare the λ-term xyz (the application of x to y, then to z) and its equivalent $\bar{\lambda}\mu\tilde{\mu}$ expression $\langle x \| y \circ z \circ \alpha \rangle$ as trees:



The λ-term buries the first thing one wants to mention in natural language deep in the structure of the term, while the $\bar{\lambda}\mu\tilde{\mu}$ expression is a convenient comb where structural recursion finds the information in the order it wants to use it. That is a central point of why a simple, strictly depth-0 structural rendering gives so much better results with the $\bar{\lambda}\mu\tilde{\mu}$ family than with λ-calculus. For a more detailed explanation of this point, see [1].

Definition 1. *A command whose environment ends in α is said to* conclude α. *It ultimately concludes α if it concludes α or ends in 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.

2.2 Typing

The typing rules are:

$$\frac{}{\Gamma, x{:}T, \Gamma' \vdash x : T \mid \Delta}(x{:}T') \notin \Gamma' \qquad \frac{}{\Gamma \mid \alpha : T \vdash \Delta, \alpha{:}T, \Delta'}(\alpha{:}T') \notin \Delta$$

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

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

Definition 2 ($\mathbf{LK}_{\mu\tilde{\mu}}$). *The sequent calculus formed by dropping the $\bar{\lambda}\mu\tilde{\mu}$ expressions in these rules (but not the stoup structure, nor the names from Γ and Δ) is called $LK_{\mu\tilde{\mu}}$.*

Intuitionistic Fragment Intuitionistic logic (eventually augmented with an axiom scheme like excluded middle or a double negation law, which is the more usual system for classical logic in the mathematical vernacular) is obtained by restricting the use of environment variables to only the most recently (innermost) bound one; equivalently, by restricting the universe of environment variables to just one name; in this case, we choose $\star$ for that unique name.

Definition 3. *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 leading to an occurrence of α traverses a μ.*

2.3 Structure of $\bar{\lambda}\mu\tilde{\mu}$

Definition 4. *$x{:}T \epsilon \Gamma$ if and only if $\Gamma = \Gamma', x{:}T, \Gamma''$ and $\nexists T', (x{:}T') \in \Gamma''$. Symmetrically, $\alpha \epsilon \Delta$ if and only if $\Delta = \Delta', \alpha{:}T, \Delta''$ and $\nexists T', (\alpha{:}T') \in \Delta'$. By abuse of notation, $x \epsilon \Gamma$ iff $T \epsilon \Gamma$ iff $x{:}T \epsilon \Gamma$ and the same for α instead of x.*

Stoup Structure

Remark 2. Note that one can put any hypothesis from Γ in the stoup with the pattern $\langle x \| \dots \rangle$; this amounts to a kind of contraction; if $x{:}A \epsilon \Gamma$ then this pattern types as:

$$\dfrac{\dfrac{}{\Gamma \vdash A \mid \Delta} \qquad \dfrac{\vdots}{\Gamma \mid A \vdash \Delta}}{\Gamma \vdash \Delta}$$

Actually, in order to left-introduce on $x \epsilon \Gamma$, one is forced to put it in the stoup, and thus to contract it in the way above. In our understanding of the intent of the proof that a $\bar{\lambda}\mu\tilde{\mu}$ term represents, we recognise that pattern as a mere invocation of x and will thus not render it as we render a cut in general.

Reduction Computationally, μ binds the environment it is combined with; dually, $\tilde{\mu}$ binds the term it is combined with, leading to these reduction rules, which form a critical pair:

$$\langle \mu\alpha{:}T.c \| e \rangle \rightsquigarrow c\{\alpha{:} = e\} \qquad\qquad \langle v \| \tilde{\mu}x{:}T.c \rangle \rightsquigarrow c\{x := v\}$$

In particular, $\langle v \| \tilde{\mu}x{:}T.c \rangle$ is better known as `let` $x{:}T := v$ `in` c. μ gives the current goal a name (so that it can be referred to later, e.g. switched back to) and gets the proof back in the "neutral" state where no goal is in focus. It can be used for the sake of either effect, or for both. The direct equivalent of β-reduction in λ-calculus is (x not free in v', e)

$$\langle \lambda x{:}T.v \| v' \circ e \rangle \rightsquigarrow \langle v' \| \tilde{\mu}x{:}T.\, \langle v \| e \rangle \rangle$$

The right hand-side then further reduces to $\langle v\{x := v'\} \| e \rangle$ or, if $v' = \mu\alpha{:}T.c$, also to $c\{\alpha{:} = \tilde{\mu}x{:}T.\, \langle v \| e \rangle\}$.

3 Natural Language Rendering

3.1 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 (doesn't improve on it).

The Syntactic Categories A term v typed by $\Gamma \vdash v : T \mid \Delta$ being a proof of $\Gamma \vdash T, \Delta$ with focus on T, intuitively its natural language rendering is some text that is a proof of $\Gamma \vdash T, \Delta$, with T being the focus (current thesis) at the beginning of the text.

$$[\![v]\!] := \text{some text}$$

An environment e typed by $\Gamma \mid e : T \vdash \Delta$ being the proof that, if provided a proof of T, one can conclude $\Gamma \vdash \Delta$, its natural language 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 $\Gamma \vdash \Delta$. The hole is denoted $\Box$. It follows intuitively that the natural language rendering of the command $\langle v \| e \rangle$, the combination of v and e into a closed whole, is the rendering of e with the placeholder filled in with the rendering of v.

$$[\![e]\!] := \text{some text } \Box \text{ some text} \qquad [\![\langle v \| e \rangle]\!] := [\![e]\!]\{\Box := [\![v]\!]\}$$

However, in the spirit of [1], we always place the hole in prefix position.

$$[\![e]\!] := \Box \text{ some text}$$

The rendering of a $\langle v \| e \rangle$ is thus always the concatenation of the text generated from v and the text generated from e. In order to lighten notations, we do not write out the hole, but write $[\![\langle v \| e \rangle]\!]$ as concatenation instead of as hole-filling:

$$[\![e]\!] := \text{some text} \qquad [\![\langle v \| e \rangle]\!] := [\![v]\!]\,[\![e]\!]$$

The rendering is based upon the rendering of [1]. We extended it to handle the whole calculus and not only the intuitionistic fragment. We also replaced the rendering of $\circ$ by "then" instead of "and"; the significance of this will be made clear later. $\boxed{\hookrightarrow}$ is an increase in indentation level and $\boxed{\hookleftarrow}$ a decrease.

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

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 then by y_Q then by y_S
  we have proven P (y_P)
  by x_R then by y_P
done proving α

The following term of the same type:

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

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 then thesis P (β)
    by x_P then by y_Q then 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.

3.2 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 and brief, we will in general not discuss the interaction between the enhancements explicitly, unless there is an interesting or problematic point; the interaction between two enhancements (if any) often amounts to a cartesian product of the rules.

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

$$\begin{aligned}\llbracket(\mu\alpha{:}T.c) \circ e\rrbracket &:= \text{the thesis is reduced to } T \qquad (\alpha)\\ &\quad \boxed{\hookrightarrow}\, \llbracket c\rrbracket\, \llbracket e\rrbracket\end{aligned}$$

Note that "then thesis" is already more clear than the "and thesis" of [1]. Read the previous example again while mentally doing the replacement.

Intuitionistic Logic Here, we recognise when single-goal logic (i.e. intuitionistic logic augmented or not with 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; $\hookleftarrow$ is a line break.

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

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. Announcing such changes at the start of the corresponding subproof, rather than at the end, leads to proofs easier to follow and more in the style of existing declarative proof languages; the latter typically requires it. 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 $\mathrm{LK}_{\mu\tilde{\mu}}$. This corresponds to the pattern $\mu\alpha{:}T.c$ where c does not ultimately conclude α. However, we delay the announcement until the command that effectively concludes (the command whose environment does not end in a binder, that is a $\tilde{\mu}$). As a solution, we use a subscript in the transformation to keep track of the thesis currently active in the natural language text.

$$\begin{aligned}\llbracket\, \langle v \| e\rangle \rrbracket_\beta &\text{ when } e \text{ does not conclude } \beta \text{ and concludes } \alpha\\ &:= \text{we now consider thesis } \alpha\\ &\quad \llbracket v\rrbracket_\alpha \llbracket e\rrbracket_\alpha\\ \llbracket\mu\beta{:}T.c\rrbracket_\alpha &:= \text{thesis } T\ (\beta)\\ &\quad \boxed{\hookrightarrow} \llbracket c\rrbracket_\beta\end{aligned}$$

This rendering keeps implicit in the natural language text that the active thesis is the most recently introduced one. If one wishes to follow the mechanics of sequent calculus more closely, e.g. if one wishes to be didactic about sequent calculus, one can instead use:

$$\begin{aligned}\llbracket \mu\beta{:}T.c \rrbracket_\alpha := {} & \text{thesis } T\ (\beta)\\ & \boxed{\hookrightarrow} \llbracket c \rrbracket_\perp\end{aligned}$$

$\perp$ is here used only as a convenient symbol that is not an environment variable, in order to always trigger the "e does not conclude β" condition in the $\langle \| \rangle$ rule.

– A cut. As already remarked, we will not consider the pattern $\langle x \| e \rangle$ as a cut for the purposes of the rendering; it is a cut that is forced upon us by the stoup structure, whose purpose is to emulate contraction, which is implicit by the scoping of the name of the hypothesis in the natural language. 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:

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

As to the pattern $\langle v \| e \rangle$, where v starts with a right-introduction rule (that is, a λ), it is dismissed as "bad style": It is a proof that does a thesis change, but refuses to announce it. 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.

– When the right child of a left introduction rule is neither a left introduction rule, nor an axiom rule. That is the pattern $\langle v \| e(\tilde{\mu}x{:}T.c) \rangle$ in $\bar{\lambda}\mu\tilde{\mu}$; where $e(\tilde{\mu}x{:}T.c)$ 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 $\cdot$.

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

This rule and the "detect a cut" rule form a critical pair; we resolve it 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 3. 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, what 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 a new syntax command being that the first $\cdot$ is instantiated by the first e, the second by the second, etc. The typing rules can be adapted by giving $\cdot$ the type $\cdot$ and when typing a command substituting the types of the es for the $\cdot$s in the type of the E. E containing several $\cdot$ is not possible in pure $\bar{\lambda}\mu\tilde{\mu}$, but will become possible in the extensions below.

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:

$$[\![x_0 \circ x_1 \circ \cdots \circ x_n \circ e]\!] := \text{then by } x_0, x_1, \ldots, x_{n-1}, x_n\ \ [\![e]\!]$$

$$\begin{aligned}
&[\![(\mu\alpha_0{:}T_0.c_0) \circ \cdots \circ x_j \circ \ldots \\
&\quad \cdots \circ (\mu\alpha_i{:}T_i.c_i) \circ \cdots \circ e]\!] := \text{the thesis is reduced to:} \\
&\qquad\qquad \bullet\ T_0\ (\alpha_0) \\
&\qquad\qquad\ \ \boxed{\hookrightarrow}\,[\![c_0]\!] \\
&\qquad\qquad \ldots \\
&\qquad\qquad \bullet \text{ assumption } x_j \\
&\qquad\qquad \ldots \\
&\qquad\qquad \bullet\ T_i\ (\alpha_i) \\
&\qquad\qquad\ \ \boxed{\hookrightarrow}\,[\![c_i]\!] \\
&\qquad\qquad \ldots \\
&\qquad\qquad [\![e]\!]
\end{aligned}$$

Note that any environment of the shape $v \circ e$, where the root of v is a recursive constructor other than μ (e.g. λ) tends to give a rather unintelligible rendering; this is not fixed on purpose. The view is that such an expression encodes a proof in bad style; fixing it crosses the line of showing a *better* proof than the one written, not the proof written. It is best to handle this kind of improvements at the calculus level, by η-expanding such a v into $\mu\alpha{:}T.\,\langle v \| \alpha \rangle$.

Simple Proofs One can avoid using a block/bullet/indent structure for "simple" proofs, for some notion of "simple" that implies the proof can be written in one line. For example ($e(e')$ is an environment that terminates in exactly one e'):

$$\begin{aligned}
[\![\mu\alpha{:}T.c(\alpha)]\!] &\text{ where } c \text{ is simple} \\
&:= \text{thesis } A\ (\alpha){:}\,[\![c(\alpha)]\!] \\
[\![\,\langle v \| e(\tilde{\mu}y{:}T.c)\rangle]\!] &\text{ where } \langle v \| e(\tilde{\mu}y{:}T.c)\rangle \text{ is simple} \qquad [\![\cdot]\!]^c := \\
&:= [\![v]\!]\,[\![e(\cdot)]\!]^c \text{ we have } T\ (x) \\
&\quad\ [\![c]\!]
\end{aligned}$$

"simple" can for example mean "made of only term variables, $\langle \| \rangle$, $\circ$ and an environment terminal". E.g. $\langle x \| x_o \circ \cdots \circ x_n \circ \tilde{\mu}y{:}T.c \rangle$ is simple even if c is not.

4 Other Constructors for Implication

There is a variety of alternative ways to handle implication in the calculus [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], in conflict with another constructor; we thus rename it to $\lambda\_{:}A$, with the convention that

_ is a special name for "do not bind to a name":

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

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

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

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

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:

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

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

For the purposes of intuitionistic logic, one should consider that this binds the $\star$ to a dummy value, so that the previous binding does not get exposed in c.

Furthermore, it 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↵ $[\![e]\!]$", which had to be combined with changing the translation for $\tilde{\mu}x{:}T.c$ to "we can now assume T↵ $[\![c]\!]$", 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)
...

See page 24 for further discussion of this ι_1.

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

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

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

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

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

$\boldsymbol{\mu\_{:}T.c}$ 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 then by y, we have proven T, but don't use that fact in the rest of the proof ...".

5 Link to proof assistants

We here succinctly treat transformation of intuitionistic-logic $\bar{\lambda}\mu\tilde{\mu}$ (not the extended calculus) terms typed by $\vdash v : T \mid$ into Isabelle/Isar/HOL with a focus on simplicity rather than completeness by way of example; transformation into Mizar is similar (except that Mizar can *only* do forward steps, no backward step); transformation into PVS is designed, but less direct and not included here for lack of place. The transformation is not much more than an alternate

syntax for a partially enhanced natural language rendering of $\bar{\lambda}\mu\tilde{\mu}$, showing that there is a good correspondence between $\bar{\lambda}\mu\tilde{\mu}$ and Isar proofs that do not use any automation. It should be noted that not all $\bar{\lambda}\mu\tilde{\mu}$ terms can be mapped faithfully to an Isar proof: there are $\bar{\lambda}\mu\tilde{\mu}$ proof fragments 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:

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

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. These rules identify the subcalculus of $\bar{\lambda}\mu\tilde{\mu}$ that Isar proofs map to.

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 use the alternate syntax of page 12 for clarity.

$$\begin{aligned} \texttt{()} &:= \texttt{-} & \mathrm{cl}(\lambda x{:}T.v) &:= \mathrm{cl}(v) + 1 \\ \mathrm{cl}(\mu\alpha{:}T.c) &:= 0 & \mathrm{cl}(\langle v|E|\tilde{\mu}x{:}T.c\rangle) &:= \mathrm{cl}(c) \\ \mathrm{cl}(\langle v|E|\alpha\rangle) &:= \mathrm{cl}(v) & \mathrm{cl}(x) &:= 0 \end{aligned}$$

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

$$\begin{aligned} [\![\langle v|\cdot|\alpha\rangle]\!] := \texttt{proof (rule impI, ...}&\mathrm{cl}(v)\text{ times}\texttt{..., rule impI)}\ \boxed{\hookrightarrow} \\ &[\![v]\!]\ \boxed{\hookleftarrow} \\ \texttt{qed}& \end{aligned}$$

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

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. Here is a variant that is closer to the basic natural language rendering:

$$
\begin{array}{rl}
[\![\mu\alpha{:}T_\alpha.\, \langle v|E|\tilde{\mu}x{:}T_x.c\rangle]\!] := & \texttt{have } T \;\; [\![\, \langle v|E|\alpha\rangle]\!] \\
& \texttt{hence } x{:}T\texttt{.} \\
& [\![\mu\alpha{:}T_\alpha.c]\!]
\end{array}
$$

6 Extension to Propositional Logic; Introduction Rules

6.1 Conjunction

We add conjunction introduction in the different reasonable ways:

$$
\begin{array}{l}
v ::= \ldots \mid (v, v) \\
e ::= \ldots \mid \pi_1[e] \mid \pi_2[e] \mid (x{:}A, y{:}B).c \mid \lambda_1 x{:}A.e \mid \lambda_2 x{:}A.e
\end{array}
$$

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

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

$$
\frac{\Gamma \mid e : T \vdash \Delta}{\Gamma \mid \pi_1[e] : T \wedge T' \vdash \Delta}
\qquad
\frac{\Gamma \mid e : T' \vdash \Delta}{\Gamma \mid \pi_2[e] : T \wedge T' \vdash \Delta}
$$

$$
\begin{array}{rlrl}
[\![(v, v')]\!] := & \bullet\, [\![v]\!] & [\![\pi_{1,2}[e]]\!] := & \text{a fortiori } [\![e]\!] \\
& \bullet\, [\![v']\!] & [\![\lambda_{1,2}x{:}T.e]\!] := & \text{we have proven } T\ (x) \text{ and } [\![e]\!] \\
& & [\![(x{:}A, y{:}B).c]\!] := & \text{we have proven } A\ (x) \text{ and } B\ (y) \\
& & & [\![c]\!]
\end{array}
$$

Example 1 We here show proofs of $A \wedge B \to (A \to B \to C) \to C$, using the various ways to destruct conjunction.

Proof 1

$$\lambda x{:}A \wedge B.\lambda y{:}A \to B \to C.\mu\alpha{:}C.\\ \langle y \| (\mu\beta{:}A.\, \langle x \| \pi_1[\beta]\rangle) \circ (\mu\beta{:}A.\, \langle x \| \pi_2[\beta]\rangle) \circ \alpha\rangle$$

Basic rendering:

assume $A \wedge B$ (x)
assume $A \to B \to C$ (y)
thesis C (α)
  by y
  then thesis A (β)
    by x a fortiori
  done proving β
  then thesis B (β)
    by x a fortiori
  done proving β
done proving α

Enhanced rendering:

assume $A \wedge B$ (x)
assume $A \to B \to C$ (y)
we have to prove C
  by y the thesis is reduced to
- A
    by x a fortiori
  done
- B
    by x a fortiori
  done

done

Proof 2

$$\lambda x{:}A \wedge B.\lambda y{:}A \to B \to C.\mu\alpha{:}C.\, \langle x \| (z{:}A, z'{:}B).\, \langle y \| z \circ z' \circ \alpha\rangle\rangle$$

Basic rendering:

assume $A \wedge B$ (x)
assume $A \to B \to C$ (y)
thesis C (α)
  by x we have A (z) and B (z') (z, z')
  by y then z then z'
done proving α

Proof 3

$$\lambda x{:}A \wedge B.\lambda y{:}A \to B \to C.\mu\alpha{:}C.\\ \langle y \| (\mu\beta{:}A.\, \langle x \| \lambda_1 z{:}A.\beta\rangle) \circ (\mu\beta{:}B.\, \langle x \| \lambda_2 z{:}B.\beta\rangle) \circ \alpha\rangle$$

Basic rendering:

assume $A \wedge B$ (x)
assume $A \to B \to C$ (y)
thesis C (α)
  by y
  then thesis A (β)
    by x we have A (z) and (z)
  done proving β
  then thesis B (β)
    by x we have B (z) and (z)
  done proving β
done proving α

Enhanced rendering:

assume $A \wedge B$ (x)
assume $A \to B \to C$ (y)
we have to prove C
  by y the thesis is reduced to
  • A
      by x we have A (z) and (z)
    done
  • B
      by x we have B (z) and (z)
    done
done

Example 2 In the previous example, the use of $\lambda_{1,2}$ gives a rather inelegant proof, but they can be quite useful, as in this proof of $A \wedge ((C \wedge D) \wedge B) \to T$:

$$\lambda x{:}A \wedge ((C \wedge D) \wedge B).\mu\alpha{:}T.\, \langle x \| \lambda_1 y_A{:}A.\lambda_2 y_B{:}B.(y_C{:}C, y_D{:}D).\ldots \rangle$$

Basic rendering:

assume $A \wedge ((C \wedge D) \wedge B)$ (x)
thesis T (α)
  by x
  we have A and (y_A)
  we have B and (y_B)
  we have C (y_C) and D (y_D) (y_C, y_D)
  ...

Enhanced Pseudo-Natural Language Many of the improvements of section 3.2 apply to conjunction mutadis mutandis. A few examples: $(x{:}A, y{:}B)$ is a $\tilde{\mu}$-like construct (it is a binder and a terminal environment constructor); the special handling applied to $\tilde{\mu}$ thus benefits $(x{:}A, y{:}B)$ in the same way. (v, v) benefits from block-structure avoidance if its subterms are "simple", too. Just as sequences

of $\circ$ can be collected in an emulation of n-ary implication, sequences of π_1, π_2, λ_1 can be collected in an emulation of n-ary conjunction:

$$[\![\pi_{1,2}[\pi_{1,2}[\pi_{1,2}[\ldots e \ldots]]]]\!] := \text{a fortiori } [\![e]\!]$$
$$[\![\pi_{1,2}[\pi_{1,2}[\pi_{1,2}[\ldots \lambda_{1,2}x{:}T.e \ldots]]]]\!] := \text{a fortiori we have } T\ (x) \text{ and } [\![e]\!]$$

It is actually even more useful for conjunction than it is for implication; the basic rendering renders $\pi_1[\pi_2[\lambda_1 x{:}A.e]]$ as "a fortiori a fortiori a fortiori we have A (x) ...", which is more ugly than the basic rendering of e.g. $x_0 \circ x_1 \circ \cdots$.

Here we also see a gain of using "then" for $\circ$, in deviation with [1]. With "and" for $\circ$, the rendering of e.g. $\lambda_1 x{:}T.(y \circ \alpha)$ is "we have proven T (x) and and by y done", which has two times the word "and" contiguously. While using "then" for $\circ$, the rendering is "we have proven T (x) and then by y done", which is nicer. The other gain, naturally, is that avoids overloading the word "and" to mean both application (as in "by x and by y done") and the natural language grammatical coordinating conjunction (as in "we have proven T (x) and P (y)"), making the produced pseudo-natural language better structured, less confusing.

6.2 Propositional Logic

In a similar way, we add the other connectives of propositional logic.

$$v ::= \ldots \mid \iota_{1,2}(v) \mid [\alpha{:}A, \beta{:}B].c \mid \lambda_{1,2}\alpha{:}A.v \mid \lambda_\neg x{:}A.v \mid \ltimes$$
$$e ::= \ldots \mid [e, e] \mid \neg[v] \mid \rtimes$$

$$\frac{\Gamma \vdash v : T \mid \Delta}{\Gamma \vdash \iota_1(v) : T \vee T' \mid \Delta} \qquad \frac{\Gamma \vdash v : T' \mid \Delta}{\Gamma \vdash \iota_2(v) : T \vee T' \mid \Delta} \qquad \frac{\Gamma, x{:}T \vdash v : \bot \mid \Delta}{\Gamma \vdash (\lambda_\neg x{:}T.v) : \neg T \mid \Delta}$$

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

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

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

$$[\![\iota_1(v)]\!] := \text{it suffices to prove the left part of the disjunction } [\![v]\!]$$
$$[\![\iota_2(v)]\!] := \text{it suffices to prove the right part of the disjunction } [\![v]\!]$$
$$[\![\lambda_{1,2}\alpha{:}A.v]\!] := \text{keeping in mind that we may prove } A\ (\alpha),\ [\![v]\!]$$

$$[\![[\alpha{:}A, \beta{:}B].c]\!] := \text{thesis } A\ (\alpha) \text{ or } B\ (\beta)\ \boxed{\hookrightarrow}\ [\![c]\!]$$
$$[\![[e, e']]\!] := \text{either}\ \boxed{\hookrightarrow}\ [\![e]\!]\ \text{or}\ \boxed{\hookrightarrow}\ [\![e']\!]\ \boxed{\hookleftarrow}$$
$$[\![\neg[v]]\!] := \text{then } [\![v]\!]\ \boxed{\hookleftarrow}\ \text{done } (ECQ)$$
$$[\![\ltimes]\!] := \text{true}$$
$$[\![\rtimes]\!] := \boxed{\hookleftarrow}\ \text{done } (EFQ)$$
$$[\![\lambda_\neg x{:}A.v]\!] := \text{assume } A\ (x)\ [\![v]\!]$$

ECQ is "ex contradictione (sequitur) quodlibet" and EFQ "ex falso (sequitur) quodlibet". Some of these constructs are inherently not intuitionistic; this needs to be taken into account when restricting oneself to intuitionistic logic. The terminal environment constructors that are not binders (α-like), have to be treated as such in the enhanced natural language rendering. $[\alpha{:}A, \beta{:}B]$, being a μ-like construct, needs to be treated accordingly in the enhanced rendering. Note that according to our definition, e.g. $\rtimes$ and $\neg[v]$ conclude α for any α. The rendering of $\lambda_{1,2}$, $\iota_{1,2}$ and $[e, e']$ particularly benefit from typing information, as in:

$$\begin{aligned}
[\![\iota_{1,2}(v)]\!] &:= \text{it suffices to prove } t(v) & [\![[e,e']]\!] &:= \text{either } t(e)\\
&\quad [\![v]\!] & &\quad \boxed{\hookrightarrow}[\![e]\!]\\
[\![\lambda_1\alpha{:}A.(v)]\!] &:= \text{keeping in mind we may prove } A\ (\alpha), & &\quad \text{or } t(e')\\
&\quad \text{we proceed with the proof of } t(v) & &\quad \boxed{\hookrightarrow}[\![e']\!]\boxed{\hookleftarrow}
\end{aligned}$$

The introduction of $[\cdot,\cdot]$ has interesting consequences for the enhanced rendering: there is not anymore unicity of the terminal environment constructor. For example, one branch can conclude α and the other branch finish in a $\tilde{\mu}x{:}T.c$. 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 the "announce thesis changes" enhancement, for example:

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

For the alternative syntax of page 12, it means a command needs to take a list of es whose length matches the number of $\cdot$ in the E.

There is another possible term constructor for negation, which corresponds to the usual sequent calculus rule:

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

But it takes an environment as an argument, which integrates badly with the rest of the rendering, so we don't keep it.

7 Classical logic

We remind the reader that sequent calculus handles classical logic by maintaining a set of goals throughout the reasoning; concluding any one of these goals concludes the whole proof. In a $\bar{\lambda}\mu\tilde{\mu}$ command, this set of goals is the set of all environment variables the command has a name for; that is Δ in the typing judgement. For a $\bar{\lambda}\mu\tilde{\mu}$ term, it is the environmental variables the term has a name for (the Δ in the typing judgement), augmented with the type of the term (the type in the stoup), which does not have a name. A $\mu\alpha{:}T$ is the binding of this unnamed goal, T to the name α.

We now consider, by way of example of a proof done in classical logic, the following term, of type $((A \wedge A) \to B) \to ((\neg A \vee B) \to A) \to B$:

$$\begin{array}{l}\lambda x{:}(A \wedge A) \to B.\lambda y{:}(\neg A \vee B) \to A.\mu\alpha{:}B.\\ \big\langle \mu\beta{:}A.\, \langle y \| (\mu\gamma{:}\neg A \vee B.\, \langle \iota_1(\lambda_\neg z'{:}A.\mu\_{:}\bot.\, \langle z' \| \beta\rangle) \| \gamma\rangle) \circ \beta\rangle \,\big|\\ \quad \tilde{\mu}z{:}A.\, \langle x \| (\mu\gamma{:}A \wedge A.\, \langle (z,z) \| \gamma\rangle) \circ \alpha\rangle \,\big\rangle\end{array}$$

The basic rendering of that term is:

assume $(A \wedge A) \to B$ (x)
assume $(\neg A \vee B) \to A$ (y)
thesis B (α)
  thesis A (β)
    by y then thesis $\neg A \vee B$ (γ)
      it suffices to prove the left of the disjunction
      assume A (z')
      we give up on the current thesis $\bot$
        by z'
      done proving β
    done proving γ
  done proving β
  we have proven A (z)
  by x then thesis $A \wedge A$ (γ)
    • by z
    • by z
  done proving γ
done proving α

It does not resemble a vernacular proof, because those are usually not written in sequent calculus multiple-goals style. Our purpose here is to propose a way to speak about sequent calculus proofs in a natural language kind of way. We could transform the proof into e.g. intuitionistic logic + excluded middle, and then only translate it to pseudo-natural language, but we argue that this is presenting a *different* proof than the one that was done; it can be useful for some purposes (and $\bar{\lambda}\mu\tilde{\mu}$ gives us a language to talk about that transformation), but showing the proof as it was done (e.g. in PVS) has its uses. An enhanced rendering is:

assume $(A \wedge A) \to B$ (x)
assume $(\neg A \vee B) \to A$ (y)
we have to prove B
  we have A (β, z)
  proof
    by y the thesis is reduced to $\neg A \vee B$
      it suffices to prove $\neg A$
      assume A (z')
      we now consider thesis β: by z' done
    done
  done
  by x the thesis is reduced to $A \wedge A$: by z and z done
done

For people not used at all to sequent calculus, this alternative rendering may help; it makes explicit at every point what the set of acceptable goals is, instead of leaving it implicit as "all $\alpha, \beta, \ldots$ names in scope", and what goal (thesis) one is currently considering:

assume $(A \wedge A) \to B$ (x)
assume $(\neg A \vee B) \to A$ (y)
left to prove: B (α)
  we now prove A (or B) (β, z)
    we now consider thesis β:A
    by y the thesis is reduced to $\neg A \vee B$ (or any of A, B) (γ)
      we now consider thesis γ:$\neg A \vee B$
      it suffices to prove $\neg A$
      assume A (z')
      left to prove: $\bot$ (or any of $\neg A \vee B$, A, B)
        we now consider thesis β:A
        by assumption z'
      done
    done
  done
  we now consider thesis α:B
  by x the thesis is reduced to $A \wedge A$: by z and z done
done

We want a saturated calculus. So there should also be constructs in it to handle classical logic the usual way, that is with a double negation law or excluded middle.

$$v ::= \ldots \mid \mathrm{TE}(T) \mid \mathrm{DN}(v) \qquad e ::= \ldots \mid \mathrm{DN}(e)$$
$$[\![\mathrm{DN}(v)]\!] := \text{proof by contradiction } [\![v]\!] \qquad [\![\mathrm{TE}(T)]\!] := \text{by TE}$$
$$[\![\mathrm{DN}(e)]\!] := \text{by double negation elimination } [\![e]\!]$$

TE stands for (principium) tertii exclusi.

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

For example $\langle \mathrm{TE}(P) \| [\cdot, \cdot] \rangle$ and $\mathrm{DN}(\lambda_{\neg} x{:}\neg T.v)$ render to, respectively:

by TE either P	proof by contradiction
...	assume $\neg T$ (x)
or $\neg P$	$[\![v]\!]$
...	

Let's also note that in a classical logic setting, the problematic implication ι_1 has a good variant:

$v ::= \ldots \mid \xi(v) \quad [\![\xi(v)]\!] :=$ it suffices to prove the antecedent does not hold

$$\frac{\Gamma \vdash v : \neg A \mid \Delta}{\Gamma \vdash \xi(v) : A \to B \mid \Delta}$$

Or, with access to typing information:

$$[\![\xi(v)]\!] := \text{it suffices to prove } t(v)$$

In a classical logic setting, we add alternative rules for some connectors:

$$\begin{array}{rr} v ::= \ldots \mid \psi_{1,2} x{:}A.v & e ::= \ldots \mid [e,e]_{\to} \mid [e,e]_{1,2} \\ [\![\psi_{1,2}(v)]\!] := \text{assume } A\ (x) & [\![[e,e]_{\to}]\!] := [\![[e,e]]\!] \\ [\![v]\!] & [\![[e,e]_{1,2}]\!] := [\![[e,e]]\!] \end{array}$$

$$\frac{\Gamma, x{:}\neg A \vdash v : B \mid \Delta}{\Gamma \vdash \psi_1 x{:}\neg A.v : A \vee B \mid \Delta} \qquad \frac{\Gamma, x{:}\neg B \vdash v : A \mid \Delta}{\Gamma \vdash \psi_2 x{:}\neg B.v : A \vee B \mid \Delta}$$

$$\frac{\Gamma \mid e : \neg A \vdash \Delta \qquad \Gamma \mid e' : B \vdash \Delta}{\Gamma \mid [e,e']_{\to} : A \to B \vdash \Delta}$$

$$\frac{\Gamma \mid e : A \vdash \Delta \qquad \Gamma \mid e' : \neg A \wedge B \vdash \Delta}{\Gamma \mid [e,e']_1 : A \vee B \vdash \Delta} \qquad \frac{\Gamma \mid e : A \wedge \neg B \vdash \Delta \qquad \Gamma \mid e' : B \vdash \Delta}{\Gamma \mid [e,e']_2 : A \vee B \vdash \Delta}$$

8 Predicate Logic

Universal and existential quantification are interpreted as dependent product and sum, respectively. We keep the traditional notations for the quantifiers ($\forall$ and $\exists$) rather than those for product and sum (Π and Σ). A quantification variable will be denoted $a, b, \ldots$, and an element of a quantification domain (be it first or second order) t. The notations of implication are recycled.

$$\begin{array}{rr} v ::= \ldots \mid \lambda a{:}A.v \mid (t,v) & e ::= \ldots \mid t \circ e \mid \lambda a{:}A.e \\ [\![\lambda a{:}A.v]\!] := \text{let } a{:}A\ [\![v]\!] & [\![t \circ e]\!] := \text{applied to } t\ [\![e]\!] \\ [\![\lambda a{:}A.e]\!] := \text{consider such an } a\ [\![e]\!] & [\![(t,v)]\!] := \text{take } t\ [\![v]\!] \end{array}$$

$$\frac{\Gamma \vdash v : B \mid \Delta}{\Gamma \vdash (\lambda a{:}A.v) : \forall a{:}A.B \mid \Delta}\ a \text{ not free in } \Gamma,\ \Delta \qquad \frac{\Gamma \mid e : B\{a := t\} \vdash \Delta}{\Gamma \mid t \circ e : \forall a{:}A.B \vdash \Delta}$$

$$\frac{\Gamma \mid e : B \vdash \Delta}{\Gamma \mid (\lambda a{:}A.e) : \exists a{:}A.B \vdash \Delta}\ a \text{ not free in } \Gamma,\ \Delta \qquad \frac{\Gamma \vdash v : B[a := t] \mid \Delta}{\Gamma \vdash (t, v) : \exists a{:}A.B \mid \Delta}$$

9 Administrativia

Proof assistant proofs often have proof steps like "delete assumption x". Vernacular proofs typically do not, they just don't use x. In the context of capturing proof assistant proofs, it may be judicious to add this kind of "anti-binder" to the calculus.

10 Future Work

There are several directions in which this can be taken further. The foremost glaring need is that the problem of capturing automation in theorem provers needs to be addressed for the language to be really functional in practice 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), ...).

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 language 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 $\mathrm{LK}_{\mu\tilde{\mu}}$. 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.: Fourth International Conference on Mathematical Knowledge Management (MKM2005). Volume 3863 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2006) 234–249
2. Sacerdoti Coen, C.: Declarative representation of proof terms. In: Programming Languages for Mechanized Mathematics Workshop. Volume 07-10 of RISC Report Series., University of Linz (Austria) (July 2007) 3–18
3. Kirchner, F.: Interoperable proof systems. PhD 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.: Fifth International Conference on Mathematical Knowledge Management (MKM2006). Volume 4108 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2006) 67–81
5. Curien, P.L., Herbelin, H.: The duality of computation. In: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000. SIGPLAN Notices 35(9), ACM (2000) 233–243
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. Asperti, A., Guidi, F., Padovani, L., Sacerdoti Coen, C., Schena, I.: Mathematical knowledge management in helm. Annals of Mathematics and Artifical Intelligence **38** (may 2003) 27–46
8. Corbineau, P.: A declarative proof language for the Coq proof assistant. In: TYPES 2007: Types for Proofs and Programs. Volume 4941 of Lecture Notes in Computer Science., Springer-Verlag (2007)
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.: Theorem Proving in Higher Order Logics (TPHOLs'99). Volume 1690 of Lecture Notes in Computer Science., Springer Verlag (1999)
10. Owre, S., Rushby, J.M., , Shankar, N.: PVS: A prototype verification system. In Kapur, D., ed.: 11th International Conference on Automated Deduction (CADE). Volume 607 of Lecture Notes in Artificial Intelligence., Saratoga, NY, Springer-Verlag (jun 1992) 748–752
11. Trybulec, A.: Mizar language `http://mizar.org/language/`.
12. Coscoy, Y.: Explication textuelles de preuves pour le calcul des constructions inductives. Thèse d'université, Université de Nice-Sophia-Antipolis (September 2000)