

The $\bar{\lambda}\mu\tilde{\mu}$ -Calculus as a Setting for (Formalist) Argumentation

Max Rapp

November 13, 2024

About me

I am a PhD student/scientific research assistant in the KWARC-group (aka Professur für Wissensrepräsentation u. -verarbeitung) under the supervision of Michael Kohlhase.

- ▶ Originally came on board for the ALMANAC-project funded by the RATIO-SPP (Robust Argumentation Machines);
- ▶ Background in Philosophy of Science (M.Sc.) and Logic (M.Sc.);
- ▶ My interests are in knowledge representation for formal argumentation and legal AI ..
- ▶ as well as what I call “formalist argumentation” (this talk).
- ▶ Currently, I am working with Merlin on the DIREGA-project.
- ▶ Find me at my desk (11.125), or via email, Matrix etc. (myfirstname.mylastname[at]fau.de)

What even is an Argument (Computationally Speaking)?

According to the Stanford Encyclopedia of Philosophy “[formalism in mathematics] is a position which it is fair to say most philosophers of mathematics still think is hopeless.” (Weir 2024)

- ▶ So let’s be much more formalist, not just about mathematics but about everything else!
- ▶ In Curry-Howard formalism, proofs can be understood as programs.
- ▶ Can we come up with a similar correspondence for arguments?

Intuitively, an (sound) argument provides evidence for its conclusion using certain commonly accepted inference rules to progress from assumptions (to be disambiguated) to a conclusion (like a proof). But arguments are ampliative (unlike proofs) and defeasible, in that they may be challenged by other arguments (unlike proofs).

Four Requirements for a Formalist Notion of Argument

This intuition gives rise to a list of requirements for a CH-formalist/computational interpretation of arguments:

- ▶ A model of gaps (enthymemes and defaults);
- ▶ A model of conflict (what is “attack” between arguments?)
- ▶ A model of defeasibility (when should an argument be retracted and what does this means?)
- ▶ A model of dialogue (how can arguments be exchanged and interact with each other?)

The $\bar{\lambda}\mu\tilde{\mu}$ -Calculus (Curien and Herbelin 2000) - My Flavor

v	$::=$	x	$ $	$\lambda x : \sigma.v$	$ $	$v \circ E$	$ $	$\mu\alpha : \sigma.c$	<i>Terms</i>
E	$::=$	α	$ $	$\alpha : \sigma\lambda.E$	$ $	$v \circ E$	$ $	$\tilde{\mu}x : \sigma.c$	<i>Environments</i>
c	$::=$	$\langle v \prec E \rangle$							<i>Commands</i>
σ, δ	$::=$	a	$ $	$\sigma \rightarrow \delta$	$ $	$\sigma - \delta$			<i>Programms</i>
$\bar{\sigma}, \bar{\delta}$	$::=$	a	$ $	$\bar{\sigma} \rightarrow \bar{\delta}$	$ $	$\bar{\sigma} - \bar{\delta}$			<i>Continuations</i>
σ, δ	$::=$	a	$ $	$\sigma \rightarrow \delta$	$ $	$\sigma - \delta$			<i>Halt</i>

Grammar: Language of the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus (adapted from Curien and Herbelin 2000)

Intuition I: Terms and Environments

- ▶ Terms are expressions with *outer* holes representing an environment to which they can be passed. The type of the term determines which environments it can be passed to.
- ▶ Environments are expressions with *inner* holes representing a term which can be passed to them. The type of the environment determines which terms can be passed to it.
- ▶ Commands non-deterministically assign terms to environments.
- ▶ The μ and $\tilde{\mu}$ -operators bind and name a term/environment respectively to continue with a different term/environment in the environment and (possibly) return to the original term/environment later .

Intuition I: Rendering Semantics

To capture this intuition we can define natural language rendering semantics:

$$\llbracket x \rrbracket := x$$

$$\llbracket \alpha \rrbracket := \text{to } \alpha$$

$$\llbracket \lambda x : \sigma.v \rrbracket := \text{Given } \sigma(x)$$

$$\llbracket v \rrbracket$$

$$\llbracket v \circ E \rrbracket := \text{with } \llbracket v \rrbracket$$

$$\llbracket \mu \alpha : \delta.c \rrbracket := \text{with current continuation } \alpha : \delta$$

$$\llbracket E \rrbracket$$

$$\llbracket c \rrbracket$$

$$\llbracket \langle v \prec E \rangle \rrbracket := \text{return } \llbracket v \rrbracket$$

$$\llbracket \tilde{\mu} x : \sigma.c \rrbracket := \text{with current term } x :$$

$$\llbracket E \rrbracket$$

$$\llbracket c \rrbracket$$

Example: Application

Function application is recovered via the expression
 $\lambda x : \sigma. \lambda y : \sigma \rightarrow \delta. \mu \alpha : \delta. \langle y \prec x \circ \alpha \rangle$ which is rendered as:

Rendering

Given $x : \sigma$

Given $y : \sigma \rightarrow \delta$

With current continuation $\alpha : \delta$

return y with x

to α

Peirce's Law rendered computationally

The $\bar{\lambda}\mu\tilde{\mu}$ -Calculus Curry-Howard corresponds to classical logic. This can be seen through the following expression corresponding to a proof of Peirce's law $((\sigma \rightarrow \delta) \rightarrow \sigma) \rightarrow \sigma$:

$$\lambda f : (\sigma \rightarrow \delta) \rightarrow \sigma. \mu \alpha : \sigma. \langle f \prec \lambda x : \sigma. \mu \beta : \delta. \langle c \prec \alpha \rangle \circ \alpha \rangle$$

Computationally speaking, Peirce's law corresponds to Call/CC:

Computational Rendering of Peirce's Law

Given $f : (\sigma \rightarrow \delta) \rightarrow \sigma$

with current continuation $\alpha : \sigma$

return f

with given $x : \sigma$

with current continuation $\beta : \delta$

return x to α .

to α .

Intuition II: From Programs to Proofs.

Via the proofs-as-programs correspondence, we can think of

- ▶ a term (in a term variable context A_1, \dots, A_n) of type T as a proof of T from hypotheses A_1, \dots, A_n ;
- ▶ Dually, an environment of type T can be thought of as a proof of a given thesis from T .

Note the asymmetry in this intuition: multiple hypotheses, only one thesis.

Intuitionistic Rendering Semantics (Sacerdoti Coen 2006)

This intuition can be captured by the following rendering semantics, restricting the set of environment variables to a singleton $\{\star\}$ (Sacerdoti-Coen 2006):

$$\begin{array}{ll} \llbracket x \rrbracket := \text{by } x & \llbracket \alpha \rrbracket := \\ \llbracket \lambda x : \sigma.v \rrbracket := \text{Assume } \sigma(x) & \text{done} \\ \llbracket v \rrbracket & \llbracket v \circ E \rrbracket := \text{and} \llbracket v \rrbracket \\ \llbracket \mu \star : \delta.c \rrbracket := \text{we need to prove } \delta & \llbracket E \rrbracket \\ \llbracket c \rrbracket & \llbracket \tilde{\mu} x : \sigma.c \rrbracket := \text{we have proved } \sigma(x) \\ \llbracket \langle v \prec E \rangle \rrbracket := \llbracket v \rrbracket \llbracket E \rrbracket & \llbracket c \rrbracket \end{array}$$

Example: Application

The application example $(\lambda x : \sigma. \lambda y : \sigma \rightarrow \delta. \mu \alpha : \delta. \langle y \prec x \circ \alpha \rangle)$ becomes:

Application/Implication Elimination: Intuitionistic Rendering

Assume $x : \sigma$

Assume $y : \sigma \rightarrow \delta$

We need to prove δ

By y and by x

done

Example: Peirce's Law

But intuitionistic rendering semantics fail for the classical case as can be seen in the following rendering of Peirce's Law:

$$\lambda f : (\sigma \rightarrow \delta) \rightarrow \sigma. \mu \alpha : \sigma. \langle f \prec \lambda x : \sigma. \mu \beta : \delta. \langle c \prec \alpha \rangle \circ \alpha \rangle$$

Peirce's Law: Failed Intuitionistic Rendering

Assume $(\sigma \rightarrow \delta) \rightarrow \sigma$ (f)

we need to prove σ

by f and assume σ (x)

we need to prove δ

by x

done (???)

done

The $\bar{\lambda}\mu\tilde{\mu}$ -Calculus Natural Deduction Style

To remedy this problem, we will provide a third intuition. First we define a notational variant of the calculus.

- ▶ The typing system of the calculus can be transposed into a natural deduction system using the method due to Lovas and Crary 2006.
- ▶ Recall that antecedents are conjunctive whereas succedents are disjunctive in sequent calculus. Transpose the calculus like so:
 - ▶ Add a dedicated type for continuations (denoted by greek-letter shaped holes).
 - ▶ Mirror the succedent to the left side of the turnstile (it becomes a conjunctive cedent of continuation-typed environments).
 - ▶ The formulae in the stoups become the single-conclusion succedent which can either hold a term or an environment (corresponding to a right-hand and left-hand stoup correspondingly).

The $\bar{\lambda}\mu\tilde{\mu}$ -Calculus: Typing rules

$$\begin{array}{c}
 \frac{}{\Sigma, x : \sigma; \Delta \vdash x : \sigma} \\
 \\
 \frac{\Sigma, x : \sigma; \Delta \vdash v : \delta}{\Sigma; \Delta \vdash \lambda x : \sigma. v : \sigma \rightarrow \delta} \\
 \\
 \frac{\Sigma; \Delta \vdash E : \delta \quad \Sigma; \Delta \vdash v : \sigma}{\Sigma; \Delta \vdash E \circ v : \sigma - \delta} \\
 \\
 \frac{\Sigma; \Delta, \alpha : \delta \vdash c : \perp}{\Sigma; \Delta \vdash \mu \alpha : \delta. c : \delta} \\
 \\
 \frac{\Sigma; \Delta \vdash v : \sigma \quad \Sigma; \Delta \vdash E : \sigma}{\Sigma; \Delta \vdash \langle v \prec E \rangle : \sigma} \\
 \\
 \frac{\Sigma; \Delta, \alpha : \delta \vdash \alpha : \delta}{\Sigma; \Delta, \alpha : \sigma \vdash F : \delta} \\
 \\
 \frac{\Sigma; \Delta \vdash v : \sigma \quad \Sigma; \Delta \vdash E : \delta}{\Sigma; \Delta \vdash v \circ E : \sigma \rightarrow \delta} \\
 \\
 \frac{\Sigma; \Delta \vdash \alpha \lambda. F : \sigma - \delta}{}
 \end{array}$$

Intuition III: Proofs and Refutations

Logically speaking ...

- ▶ ...terms can be thought of as (conditional) proofs;
- ▶ ...environments can be thought of as (conditional) refutations;
- ▶ ...commands can be thought of as contradictions.

We obtain a dialectical (proof-by-contradiction driven) natural deduction system.

Dialectical Rendering Semantics

$\llbracket x \rrbracket :=$ proved by x

$\llbracket \lambda x : \sigma. v \rrbracket :=$ Assume $x : \sigma$

$\llbracket v \rrbracket$

$\llbracket \mu \alpha : \delta. c \rrbracket :=$ then $\delta :$

for assume a ref. $\alpha : \delta$

$\llbracket c \rrbracket$

$\llbracket \tilde{\mu} x : \sigma. c \rrbracket :=$ then not $\sigma :$

for assume a proof $x : \sigma$

$\llbracket c \rrbracket$

$\llbracket \alpha \rrbracket :=$ but refuted by α
contradiction

$\llbracket v \circ E \rrbracket :=$ and $\llbracket v \rrbracket$

$\llbracket E \rrbracket$

$\llbracket \langle v \prec E \rangle \rrbracket :=$ return $\llbracket v \rrbracket \llbracket E \rrbracket$

Application Rendered Dialectically

Application (or implication elimination) becomes the following somewhat more indirect proof

$(\lambda x : \sigma. \lambda y : \sigma \rightarrow \delta. \mu \alpha : \delta. \langle y \prec x \circ \alpha \rangle)$:

Rendering

Assume $x : \sigma$

Assume $y : \sigma \rightarrow \delta$

then δ : for assume a refutation $\alpha : \delta$

proved by y and x

but refuted by α , contradiction

Peirce's Law Rendered Dialectically

But crucially, the new rendering semantics can handle Peirce's law:

$$\lambda f : (\sigma \rightarrow \delta) \rightarrow \sigma. \mu \alpha : \sigma. \langle f \prec \lambda x : \sigma. \mu \beta : \delta. \langle c \prec \alpha \rangle \circ \alpha \rangle$$

Classical Rendering

Assume $f : (\sigma \rightarrow \delta) \rightarrow \sigma$

then σ : for assume a refutation $\alpha : \sigma$

proved by f and assume $x : \sigma$

then δ : for assume a refutation of δ (β)

proved by x but refuted by α , contradiction.

but refuted by α , contradiction.

What even is an Argument - First Approximation

An argument is simply an open $\bar{\lambda}\mu\tilde{\mu}$ -expression:

- ▶ its type corresponds to the argument's conclusion and determines its polarity (for against the thesis);
- ▶ its free variables correspond to its assumptions (not hypotheses!);
- ▶ its body corresponds to its chain of reasoning.

Why this Model of Argument? The Reasons so Far.

So far we have seen that the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus (rendered as a natural deduction system) provides the following advantages:

- ▶ Single conclusion system that preserves the nice proof-theoretical properties of the sequent-calculus (see Lovas and Crary 2006)
- ▶ Term language provides structure to abstract arguments (sequent-based argumentation identifies all sequents with same typing judgments! But argumentation is highly proof-relevant!)
- ▶ Compact representation of arguments in terms of expressions.
- ▶ Natural deduction captures human reasoning well.
- ▶ Rendering semantics enable natural language explanations.

But crucially, arguments are exchanged in *argumentation* processes. Therefore it is necessary to capture the fundamental relations between arguments.

From Arguments to Argumentation: Three Modes of Attack

In most approaches to formal argumentation the fundamental relation between argument is attack.

- ▶ **Rebut** (or sometimes Rebuttal): Two arguments reach *contrary* conclusions.
- ▶ **Undermine** (unfortunately, in logical argumentation this is a type of undercut): The attacking argument's conclusion is contrary to an assumption of the attacked argument.
- ▶ **Undercut**: The attacking argument's conclusion indicates the inapplicability of a defeasible inference rule (weak implication) used in the attacked argument's body.

From Arguments to Argumentation: Support

Another relation between arguments is one of **support**.

For our purposes, an argument A supports another argument B if A 's conclusion is an assumption of B . (There are also other kinds of support relations which we omit for now).

A Striking Correspondence: Rebut and Cut

Compare the (compact) rebuttal rule from sequent-based argumentation (Arieli and Straßer 2015) with the Cut-rule of the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus:

$$\frac{\Gamma_1 \vdash \varphi \quad \Gamma_2 \vdash \neg\varphi}{\Gamma_2 \not\vdash \varphi}$$

$$\frac{\Sigma; \Delta \vdash v : \sigma \quad \Sigma; \Delta \vdash E : \sigma}{\Sigma; \Delta \vdash \langle v \prec E \rangle : \sigma}$$

- ▶ If we identify $\neg\varphi$ with a refutation of φ the conditions of the rules are isomorphic!
- ▶ Thus given a classical base logic, either cut or rebuttal could be applied.

“Discharging” Derivations

- ▶ In logic, a contradiction is handled by (non-deterministically) discharging and negating an assumption.
- ▶ Argumentation is more conservative: since we don't know which assumption (or set of assumptions) is the culprit, we merely quarantine one side of the offending contradiction.
- ▶ **Idea:** capture rebuttal in the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus by restricting the contradiction elimination rule (or computationally, restrict non-deterministic choice).

The Power of the Command Syntax

- ▶ At a state $\langle v \prec E \rangle : \sigma$ both opposing arguments are recorded in the proof term.
- ▶ Decide (non-deterministically) which argument wins (say v).
- ▶ Allow the reinstatement of v ...

Using the command syntax, we can encode rebut with the following term:

$$\mathit{rebut}_{\mathit{left}}(t_1, t_2) = \mu * : \sigma. \langle \mu _ : \sigma. \langle t_1 : \sigma \prec * : \sigma \rangle \prec \tilde{\mu} * : \sigma. \langle * : \sigma \prec t_2 : \sigma \rangle \rangle$$

Computational Rendering of Rebut

Note that the affine continuation $_$ never gets returned to, instead t_1 is returned to the outer continuation. Meanwhile, t_2 is stored using a dummy value aborting the program if the right hand side is evaluated before the left-hand side (call-by-name).

Rebut

With current continuation $* : \sigma$

return With current continuation $_ : \sigma$

return $t_1 : \sigma$ to $*$

With current term $\star : \sigma$

return $\star : \sigma$ to t_2

Argumentative Rendering Semantics for Rebut

For an argumentative rendering of rebut we just add a case for the rebut pattern (which we make sure is syntactically unique by availing ourselves of a reserved continuation assumption “rebut”):

$$\llbracket \text{rebut}_{left}(t_1, t_2) \rrbracket = \text{Rebutting } t_2 \text{ with } t_1$$

Undermine

Recall the compact direct defeat rule from sequent-based argumentation

$$\frac{\Sigma_1 \vdash \neg\varphi \quad \Sigma_2, \varphi \vdash \psi}{\Sigma_2 \not\vdash \psi}$$

Unlike in the cut rule, here the assumption, not the consequent is attacked.

$$\frac{\Sigma; \Delta \vdash v : \sigma \quad \Sigma; \Delta \vdash E : \sigma}{\Sigma; \Delta \vdash \langle v \prec E \rangle : \sigma}$$

How to Return to an Earlier Proof State: Proposed Type for a Stash/Pop-Operator

In order to capture undermine, we have to somehow return to the assumption of the attacked argument. The type of such an operator would be given by the following tautology:

$$\sigma \rightarrow (\sigma \rightarrow \delta) \rightarrow \sigma$$

This bears a certain similarity to Peirce's law but can easily be seen to be an intuitionistic tautology.

Stash/Pop in the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus

We can use the command syntax to return to an earlier proof state while “stashing” the present proof state as a continuation for the earlier state:

$$\frac{\frac{\frac{\frac{\Sigma, x : \sigma, _ : \delta; \Delta, \text{stash} : \sigma \vdash x : \sigma}{\Sigma, x : \sigma, _ : \delta; \Delta, \text{stash} : \sigma \vdash \langle x \prec \text{stash} \rangle : \sigma}}{\Sigma, x : \sigma; \Delta, \text{stash} : \sigma \vdash \tilde{\mu}_ : \delta. \langle x \prec \text{stash} \rangle : \delta}}{\Sigma, x : \sigma; \Delta, \text{stash} : \sigma \vdash x \circ \tilde{\mu}_ : \delta. \langle x \prec \text{stash} \rangle : \sigma \rightarrow \delta} \quad \frac{\frac{\Sigma, x : \sigma; \Delta, \text{stash} : \sigma \vdash x : \sigma}{\Sigma; \Delta \vdash \lambda x. t : \sigma \rightarrow \delta}}{\Sigma; \Delta \vdash \lambda x. t : \sigma \rightarrow \delta}}{\frac{\Sigma, x : \sigma; \Delta, \text{stash} : \sigma \vdash \langle \lambda x : \sigma. t \prec x \circ \tilde{\mu}_ : \delta. \langle x \prec \text{stash} \rangle \rangle : \sigma \rightarrow \delta}{\Sigma, x : \sigma; \Delta \vdash \mu \text{stash} : \sigma. \langle \lambda x : \sigma. t \prec x \circ \tilde{\mu}_ : \delta. \langle x \prec \text{stash} \rangle \rangle : \sigma}}$$

We will call t the *stashed term*, $\lambda x : \sigma. t$ the *stashed sequent*, σ the *return state* and the entire term starting with μ stash ... a *stash term*. Further we will use $\langle t' : \varphi \rangle$ as a shorthand for $\langle \lambda x : \sigma. t' \prec x \circ \tilde{\mu}_ : \varphi. \langle x \prec \text{stash} \rangle \rangle$.

Computational Rendering of Stash/Pop

The Stash tactic binds the current continuation, calls it with a given value x but then immediately binds and throws away the resulting term t and returns x instead.

Computational Rendering of Stash (Tactic form)

With current continuation $stash : \sigma$

return given $x : \sigma$

t

with x with current term $_ : \delta$

return x to $stash$.

Or if one reverses the flow of computation and thinks of x as a *default term*: the continuation $\lambda x.t$ is allowed to make use of x until another overriding (outer) continuation demands x . Then the computation done by $\lambda x.t$ is rewound and x is passed to the outer continuation.

Undermine via Cut and Stash

Given this we can treat Compact Direct Defeat as a special case of cut where the right-hand argument is a stash term.

$$\frac{\Sigma; \Delta \vdash t : \sigma \quad \Sigma, x : \sigma; \Delta \vdash \mu \text{ stash} : \sigma. \langle t' : \varphi \rangle : \sigma}{\Sigma, x : \sigma; \Delta \vdash \langle \mu \text{ stash} : \sigma. \langle t' : \varphi \rangle \prec t \rangle : \sigma}$$

corresponds to the introduction part of:

$$\frac{\Sigma_1 \vdash \neg \sigma \quad \Sigma_2, \sigma \vdash \varphi}{\Sigma_2 \not\vdash \varphi}$$

while preserving the concrete argument term corresponding to the sequent $\Sigma_2, \sigma \vdash \varphi$.

Eliminating Undermine

Instead of a single sequent elimination rule, we split Compact Direct Defeat into an introduction and an elimination form so as to not break the continuity of the derivation.

- ▶ Using the normal cut elimination form unrestrictedly would yield the “wrong” result: there should not be nondeterminism about which assumption to reject.
- ▶ So we use another restricted version of cut elimination:

$$\frac{\Sigma, x : \sigma; \Delta \vdash \langle \mu \text{ stash} : \sigma.c \prec t \rangle : \perp}{\Sigma; \Delta \vdash \tilde{\mu}x : \sigma. \langle \mu \text{ stash} : \sigma.c \prec t \rangle : \sigma}$$

Undermine: Computational Interpretation

Following our earlier interpretation of the free variable x as a default, we can interpret `undermine` computationally as follows:

- ▶ t is an outer continuation for σ ;
- ▶ the stashed term is a an inner continuation for σ which may locally use x ;
- ▶ `Undermine` takes the current term x and extracts it from the stashed term, undoing any computation carried out using x ;
- ▶ it then passes x to t and we are left with another continuation for x .

You might ask: “Why would one ever compute something only to undo it later?” but think of the stashed term as a computation that is only safe in certain environments and should fail if passed to others. *And that is exactly what happens when arguments are exchanged.*

Encoding Support: Why not Function Application?

The “obvious” way to capture undercut would seem to just use function application, but this yields the wrong result:

- ▶ Using function application, the original argument is lost and replaced by an argument with different assumptions;
- ▶ If the supporting argument rests on several assumptions itself, the “attack surface” of the original argument actually increases (but support should make the supported argument stronger!)

Luckily, we can use the command syntax to implement a better way.

Support and Alternative Proofs

Autexier and Sacerdoti Coen 2006 use the affine fragment of the $\bar{\lambda}\mu\tilde{\mu}$ -calculus to gain a representation of *alternative proofs* of a proposition:

$$\text{alt}^T(t_1, t_2) := \mu\star : T.\langle \mu_ : T.\langle t_1 \prec \star \rangle \prec \tilde{\mu}_ : T.\langle t_2 \prec \star \rangle \rangle$$

Support can be characterized recursively as follows:

$\text{support}^T(t_1, a)$ if $\text{alt}^T(t_1, a)$ where a is an assumption.

$\text{support}^T(t_1, t_2)$ if $t_2 = \text{support}^T(t_3, t_4)$

This notion of support captures the crucial property, that an argument A which is supported on some assumption a by some argument B can still be attacked on a . B can however be used as a defender of A meaning that any argument attacking A on a must defeat B .

Ongoing work: Extending Fellowship

Fellowship (Kirchner 2007) is the (afaik) only existing interactive theorem prover for the $\bar{\lambda}\mu\tilde{\mu}$ -calculus.

- ▶ I wrote a wrapper for Fellowship (with a lot of help from ChatGPT);
- ▶ so far \sim 1500 lines of (horrible) Python code;
- ▶ implements arguments, support, undermine, reinstatement/pop (so far);
- ▶ enables interactive construction of arguments, attacks/support and *scrutable* explanations in natural language.

Future Work: an actual interactive argumentation assistant.

Ongoing Work: Labelling

When an assumption/argument has been defeated, argumentation semantics demand that it cannot be used anymore (and hence, in particular, the assumptions/arguments it in turn defeated become usable again).

- ▶ Label assumptions in a given $\bar{\lambda}\mu\tilde{\mu}$ derivation (argument) as IN, OUT or UNDEC (following Wu and Caminada 2010).
- ▶ Adapt the algorithm from Arieli and Straßer 2019.
- ▶ Can this be done within the rewriting system of the calculus itself?

Ongoing Work: Proper Default Assumptions

Currently free variables do double duty as hypotheses and (default) assumptions. This is how it is done in logical argumentation but obviously not ideal (need to define some subset of variables that count as assumptions, ...).

- ▶ Adapt work by Borg et al. on integrating assumptions in sequent-based argumentation.
- ▶ Assumptions behave much like prover goals:
- ▶ Arguments become expressions with open goals instead of free variables (that is already how it is implemented in the Fellowship extension).

Future Work: Rewriting

The $\bar{\lambda}\mu\tilde{\mu}$ comes with a weakly normalizing rewriting system (Lovas and Crary 2006).

- ▶ What is a normal form for arguments?
- ▶ What is the role of different orders-of-execution (Call-by-name, Call-by-value, Call-by-X)?

Future Work: Undercut and Subaltern Calculi

We have not covered the undercut attack scheme here. The reason is that this requires something *weaker* than contrary:

Example

One can infer that an object is red if it appears red. But an undercutter to this is the information that the room is currently lit by red light.

The undercutter does not permit to infer that the object is not red but only that one cannot conclude that it is. So

$\not\vdash \text{appears_red} \rightarrow \text{red}$ instead of $\vdash \text{appears_red} \rightarrow \text{red}$

Future Work: Integrate a non-provability calculus in the vein of Bonatti and Olivetti 2002 to model undercut and enable completeness in decidable domains.

Future Work: Categorical Semantics

Melliès 2016:




In this paper, we consider a two-sided notion of dialogue category which we call dialogue chirality and which we formulate as an adjunction between a monoidal category \mathcal{A} of proofs and a monoidal category \mathcal{B} of counter-proofs equivalent to its opposite category $\mathcal{A}^{op(01)}$. The two-sided formulation of dialogue categories is compared to the original one-sided formulation by exhibiting a 2-dimensional equivalence between a 2-category of dialogue categories and a 2-category of dialogue chiralities. The resulting coherence theorem clarifies in what sense every dialogue chirality may be strictified to an equivalent dialogue category.

Conclusion

With the goal of finding a formalist interpretation of argument, we have investigated the $\bar{\lambda}\mu\tilde{\mu}$ -Calculus as a setting for argumentation.

- ▶ The calculus can be interpreted as a system of proofs and refutations modelling conflict
- ▶ We can use the command syntax to encode common argument relations including rebuttal, undermine and support and gain computational interpretations of the same.
- ▶ The command syntax also enables the interaction of arguments all in the same formal language
- ▶ **Future Work:** Address gaps using proper assumptions and defeasibility using labelling semantics.





References I

-  Arieli, Ofer and Christian Straßer (2015). “Sequent-based logical argumentation”. In: *Argument & Computation* 6.1, pp. 73–99. DOI: [10.1080/19462166.2014.1002536](https://doi.org/10.1080/19462166.2014.1002536). eprint: <https://doi.org/10.1080/19462166.2014.1002536>. URL: <https://doi.org/10.1080/19462166.2014.1002536>.
-  — (2019). “Logical argumentation by dynamic proof systems”. In: *Theoretical Computer Science* 781. Logical and Semantic Frameworks with Applications, pp. 63–91. ISSN: 0304-3975. DOI: <https://doi.org/10.1016/j.tcs.2019.02.019>. URL: <https://www.sciencedirect.com/science/article/pii/S0304397519301252>.
-  Autexier, Serge and Claudio Sacerdoti Coen (Aug. 2006). “A Formal Correspondence Between OMDoc with Alternative Proofs and the $\lambda\mu$ -Calculus.”. In: pp. 67–81. ISBN: 978-3-540-37104-5. DOI: [10.1007/11812289_7](https://doi.org/10.1007/11812289_7).

References II

-  Bonatti, Piero Andrea and Nicola Olivetti (Apr. 2002). “Sequent calculi for propositional nonmonotonic logics”. In: *ACM Trans. Comput. Logic* 3.2, pp. 226–278. ISSN: 1529-3785. DOI: 10.1145/505372.505374. URL: <https://doi.org/10.1145/505372.505374>.
-  Curien, Pierre-Louis and Hugo Herbelin (2000). “The duality of computation”. In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming. ICFP '00*. New York, NY, USA: Association for Computing Machinery, pp. 233–243. ISBN: 1581132026. DOI: 10.1145/351240.351262. URL: <https://doi.org/10.1145/351240.351262>.
-  Kirchner, Florent (2007). “Systèmes de preuve interoperables. (Interoperable proof systems)”. PhD thesis. École Polytechnique, Palaiseau, France. URL: <https://tel.archives-ouvertes.fr/pastel-00003192>.

References III

-  Lovas, William and Karl Crary (Jan. 2006). “Structural normalization for classical natural deduction”. In: URL: <https://www.cs.cmu.edu/~wlovas/papers/clnorm.pdf>.
-  Melliès, Paul-André (Nov. 2016). “Dialogue Categories and Chiralities”. In: *Publications of the Research Institute for Mathematical Sciences* 52. DOI: 10.4171/PRIMS/185.
-  Sacerdoti Coen, Claudio (2006). “Explanation in Natural Language of \neg -Terms”. In: *Mathematical knowledge management: 4th international conference, MKM 2005, Bremen, Germany, July 15-17, 2005: revised selected papers*, pp. 234–249. ISBN: 978-3-540-31430-1. DOI: 10.1007/11618027_16.
-  Weir, Alan (2024). “Formalism in the Philosophy of Mathematics”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Spring 2024. Metaphysics Research Lab, Stanford University.

References IV



Wu, Yining and Martin Caminada (Jan. 2010). “A labelling based justification status of arguments”. In: *Studies in Logic* 3, pp. 12–29.