Classical Realisability and Focalisation

Guillaume Munch–Maccagnoni[1]

Université Paris 7
Partially funded by *INRIA Saclay* and *U. Penn*.

Réalisabilité à Chambéry
June 4th, 2009

[1] Guillaume.Munch@pps...

Introduction:
Sequent calculi in computer science

Code v vs. environments e:

$$\frac{\Gamma \vdash v : A \mid \Delta \qquad \Gamma' \mid e : A \vdash \Delta'}{\langle v \| e \rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')} \text{(cut)}$$

Introduction: Sequent calculi in computer science

Code v vs. environments e:

$$\frac{\Gamma \vdash v : A \mid \Delta \qquad \Gamma' \mid e : A \vdash \Delta'}{\langle v \parallel e \rangle : (\Gamma, \Gamma' \vdash \Delta, \Delta')} \text{(cut)}$$

Reduction defined on commands:

$$\langle v \parallel e \rangle \longrightarrow \langle v' \parallel e' \rangle$$

Sequent calculi in computer science

Example

Krivine's weak head reduction machine: (Call-by-name)

$$\begin{array}{rcll} \langle v\ v' \parallel E \rangle & \rightarrow & \langle v \parallel v' \cdot E \rangle & \text{"push"} \\ \langle \lambda\alpha.v \parallel v' \cdot E \rangle & \rightarrow & \langle v[v'/\alpha] \parallel E \rangle & \text{"pop"} \end{array}$$

(using notations to come)

Sequent calculi in computer science

Example

Krivine's weak head reduction machine: (Call-by-name)

$$
\begin{array}{rcll}
\langle v\, v' \parallel E\rangle & \to & \langle v \parallel v' \cdot E\rangle & \text{"push"} \\
\langle \lambda\alpha.v \parallel v' \cdot E\rangle & \to & \langle v[v'/\alpha] \parallel E\rangle & \text{"pop"}
\end{array}
$$

(using notations to come)

Example

Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}_v$ [Curien-Herbelin 2000]. (Call-by-value)

$$
\begin{array}{rcll}
\langle v\, v' \parallel e\rangle & \to & \langle v \parallel v' \cdot e\rangle & \\
\langle \lambda x.v \parallel v' \cdot e\rangle & \to & \langle v' \parallel \mu x.\langle v \parallel e\rangle\rangle & \\
\langle V \parallel \mu x.c\rangle & \to & c[V/x] & (V \text{ value})
\end{array}
$$

Sequent calculi in computer science

- "Abstract" abstract machines (commands $\langle v \parallel e \rangle$) as a way to define operational semantics.
  - i.e. operational semantics as *syntax* with semantically fine-grained constructs.
  - programming languages (i.e. natural deduction with *useful* connectives (*abstraction*, *application*)) defined afterwards.

Sequent calculi in computer science

- "Abstract" abstract machines (commands $\langle v \parallel e \rangle$) as a way to define operational semantics.
  - i.e. operational semantics as *syntax* with semantically fine-grained constructs.
  - programming languages (i.e. natural deduction with *useful* connectives (*abstraction*, *application*)) defined afterwards.

- Back to the roots of computer science: computation as the interaction of a *program* with *data*.
  - e.g. for words $w \dots$ and states $s, s' \dots$, a finite automata (NFA) can be represented by the interaction:

$$\langle a.w \parallel s \rangle \to \langle w \parallel s' \rangle$$

Sequent calculi in computer science

- "*System* L": Syntax *à la* Curien-Herbelin (2000) for a representation of generic sequent calculi, with an "abstract machine" flavor.
- Semantical investigations [Girard's LC, Danos-Joinet-Schellinx LK^{η}_{pol}, Laurent's LLP] clarified concepts around computation in classical logic (*polarities, focalization*).

Sequent calculi in computer science

- "*System* L": Syntax *à la* Curien-Herbelin (2000) for a representation of generic sequent calculi, with an "abstract machine" flavor.
- Semantical investigations [Girard's LC, Danos-Joinet-Schellinx LK^{η}_{pol}, Laurent's LLP] clarified concepts around computation in classical logic (*polarities, focalization*).
- Here: design a "system L" that expresses into the syntax what we know about the semantics.

Focalising System L

Focalising System L (L_{foc}): A syntax for sequent calculi whose reduction rules correspond to the semantics of Girard's classical logic LC (1992).

- It is a term syntax for LK_{pol} (focalised classical logic with the 4 connectives from LL), linear logic LL...
- It has a *low technicality* and the *readability/writeability* of the λ calculus.

It naturally extends Krivine's classical realizability.

Focalising System L: Ideas

Focalisation: Four connectives (in addition to *classical* negation): $\otimes$, $\parr$, &, $\oplus$. Positive constructs are *strict*; negative constructs are *lazy*. [independently underlined by Zeilberger, 2008]

Values: The *stoup* [Girard, 1992]: same notion as the *values* [Plotkin, 1975]. [Remark due to Curien-Herbelin, 2000, might have appeared implicitly before]

Indeed,compare:

$$\frac{\Gamma \vdash A; \Delta \qquad \Gamma' \vdash B; \Delta'}{\Gamma, \Gamma' \vdash A \otimes B; \Delta, \Delta'} (\vdash \otimes) \quad \text{and} \quad V ::= (V, V) \mid \ldots$$

Focalising System L: Ideas

Two polarities: "Strict" and "Lazy" qualify connectives instead of the strategy of reduction. Code can mix constructs of the two polarities.

- Compare with e.g. $\bar{\lambda}\mu\tilde{\mu}$ or Wadler's "Dual Calculus": One non-confluent calculus with two confluent restrictions, CBV and CBN (corresponds to the "pre-1987" negations that necessarily coincide with a modality).
- Duality of connectives $\neq$ duality of constructs.

Pattern-matching: Invertible constructs are represented with an *informal* pattern-matching.

Focalising System L: Types

Polarisation: positive and negative formulae:

$$\begin{array}{rcl} A & ::= & P \mid N \\ P & ::= & X \mid A \otimes A \mid A \oplus A \mid \mathbf{1} \mid \mathbf{0} \\ N & ::= & X^{\perp} \mid A \parr A \mid A \& A \mid \bot \mid \top \end{array}$$

Positive variables $x, y \ldots$; negative variables $\alpha, \beta \ldots$.
Contexts Γ have elements of the form $\alpha : P$ or $x : N$.

Focalising System L: Constructs

- Patterns (positive constructs) and pattern-matching (negative constructs). [Link with focalisation underlined independently from Zeilberger, 2008]

$$\begin{array}{llll} \otimes: & (t, u) & \parr: & \mu(x, y).c \\ \&: & \mu(\imath_1(x).c \mid \imath_2(y).c') & \oplus: & \imath_i(t) \end{array}$$

Focalising System L: Constructs

- Patterns (positive constructs) and pattern-matching (negative constructs). [Link with focalisation underlined independently from Zeilberger, 2008]

$$\begin{array}{llll} \otimes: & (t,u) & \parr: & \mu(x,y).c \\ \&: & \mu(\imath_1(x).c \mid \imath_2(y).c') & \oplus: & \imath_i(t) \end{array}$$

- Computational interpretation:
  - $\langle(t,u)|$: constructor of the strict conjunction (OCaml's pair).
  - $|(t,u)\rangle$: destructor of the lazy disjunction (its dual for the duality of computation).
  - $|\mu(x,y).c\rangle$: destructor of the strict conjunction (dual for the duality of constructs).

Focalising System L: Constructs

$\otimes/\parr$:

$$\frac{\vdash t : A \mid \Gamma \qquad \vdash u : B \mid \Delta}{\vdash (t,u) : A \otimes B \mid \Gamma, \Delta}\ (\otimes)$$

$$\frac{c : (\vdash x : A, y : B, \Gamma)}{\vdash \mu(x,y).c : A \parr B \mid \Gamma}\ (\parr)$$

$\oplus/\&$:

$$\frac{\vdash t : A_i \mid \Gamma}{\vdash \imath_i(t) : A_1 \oplus A_2 \mid \Gamma}\ (\oplus_i)$$

$$\frac{c : (\vdash x : A, \Gamma) \qquad c' : (\vdash y : B, \Gamma)}{\vdash \mu(\imath_1(x).c \mid \imath_2(y).c') : A \,\&\, B \mid \Gamma}\ (\&)$$

One-sided vs. Two-sided sequents

Two traditions in sequent calculus:

- Gentzen two-sided sequents: $\Gamma \vdash \Delta$
  - Input/output symmetry ("duality of computation").
  - i.e. $\langle t| \neq |t\rangle$.

One-sided vs. Two-sided sequents

Two traditions in sequent calculus:

- Gentzen two-sided sequents: $\Gamma \vdash \Delta$
  - Input/output symmetry ("duality of computation").
  - i.e. $\langle t| \neq |t\rangle$.
- Girard's one sided sequents: $\vdash \Gamma$
  - Amounts to quotienting with $\langle t \parallel u\rangle \equiv \langle u \parallel t\rangle$.
  - Reasoning modulo the duality of computation.
  - No real meaning in terms of "abstract machines".

In the following: we have one sided-sequents and $\langle t \parallel u\rangle \equiv \langle u \parallel t\rangle$ for simplification. (To retrieve the two-sided version one has to add a connective for classical negation)

Focalising System L: Syntax

$$
\begin{array}{rlll}
\kappa ::= & \alpha \mid x & & \\
t_+ ::= & x & \mid \mu\alpha.c & \\
& \mid (t, t) & \mid \imath_i(t) & (\otimes, \oplus_i) \\
& \mid () & & (\mathbf{1}, \mathbf{0}) \\
t_- ::= & \alpha & \mid \mu x.c & \\
& \mid \mu(\kappa, \kappa).c & \mid \mu(\imath_1(\kappa).c \mid \imath_2(\kappa).c) & (\invamp, \&) \\
& \mid \mu().c & \mid \mathsf{tp} & (\bot, \top) \\
c ::= & \langle t_+ \parallel t_- \rangle & \mid \langle t_- \parallel t_+ \rangle &
\end{array}
$$

- Negation left implicit because we are one-sided ($\langle t \parallel u \rangle \equiv \langle u \parallel t \rangle$). (Two sided-sequents recover classical negation and resemblance to abstract machines)
- Shifts of polarities left implicit: we do not add constraints of polarity to formulae. (Constructs are their own shifts)

Focalising System L: Reduction

- Define values:

$$V ::= t_- \mid x \mid (V, V) \mid \imath_i(V) \mid ()$$

- Head reduction $\rightarrow_h$:

$$\langle \mu\alpha.c \parallel t_- \rangle \rightarrow_h c\,[t_-/\alpha]$$
$$\langle \mu p.c \parallel V_+ \rangle \rightarrow_h c\,[V_+/p]$$

  with p a *pattern* (very informal).

- Plus conventional "ς" rules:

$$\langle (t, u) \parallel v_- \rangle \rightarrow_h \left\langle t \,\middle\|\, \mu\kappa.\langle u \parallel \mu\kappa'.\langle (\kappa, \kappa') \parallel v_- \rangle \rangle \right\rangle$$

$$\ldots$$

- Reduction is deterministic.

Realisability: Observation $\perp\!\!\!\perp$, definition

Realisability: Observation $\perp\!\!\!\perp$, definition

- The *observation* $\perp\!\!\!\perp$: a set of *closed* commands which is $\rightarrow_h$-saturated:

$$c \rightarrow_h c', c' \in \perp\!\!\!\perp \implies c \in \perp\!\!\!\perp$$

- If $\langle t \parallel u \rangle \in \perp\!\!\!\perp$ then one writes $t \perp\!\!\!\perp u$.

Definition

- $T^{\perp} \stackrel{\text{def.}}{=} \{ t \mid \forall u \in T, t \perp\!\!\!\perp u \}$.
- *Behaviours* are sets of the form $T^{\perp}$.

Realisability: Observation $\perp\!\!\!\perp$, definition

- The *observation* $\perp\!\!\!\perp$: a set of *closed* commands which is $\to_h$-saturated:

$$c \to_h c', c' \in \perp\!\!\!\perp \implies c \in \perp\!\!\!\perp$$

- If $\langle t \parallel u \rangle \in \perp\!\!\!\perp$ then one writes $t \perp\!\!\!\perp u$.

Definition

- $T^{\perp} \stackrel{\text{def.}}{=} \{ t \mid \forall u \in T, t \perp\!\!\!\perp u \}$.
- *Behaviours* are sets of the form $T^{\perp}$.

Fact (Basic properties of the orthogonal)

- *If* $U \subseteq V$ *then* $V^{\perp} \subseteq U^{\perp}$.
- $U \subseteq U^{\perp\perp}$.
- $U^{\perp} = U^{\perp\perp\perp}$.
- U *is a behaviour iff* $U = U^{\perp\perp}$.

Analogy with NFAs

Take $\perp\!\!\!\perp$ a set of $\langle w \parallel s \rangle$ which is saturated for the reduction of NFAs.

- Then the $\mathcal{S}^{\perp}$ with $\mathcal{S}$ a set of states are regular languages.

Analogy with NFAs

Take $\perp\!\!\!\perp$ a set of $\langle w \parallel s \rangle$ which is saturated for the reduction of NFAs.

- Then the $\mathcal{S}^{\perp}$ with $\mathcal{S}$ a set of states are regular languages.

In particular, take $\perp\!\!\!\perp$ the smallest observation that contains $\langle \varepsilon \parallel s_F \rangle$ for each final state s_F. Then:

- $\{s_0\}^{\perp}$ is the language the automaton accepts;
- Colinearity, i.e. $\{s\}^{\perp} = \{s'\}^{\perp}$ is the Nerode equivalence.

An analogy:

- useful to introduce classical realizability,
- that shows that the interaction between the two sides of the cut is like the interaction between a *program* and *data* that lies at the roots of computer science.

Realisability: Behaviours

Back to L_{foc}. Instead of regular expressions that define regular languages: we have closed logical formulae A that define behaviours $|A|$.

- Formulas extended with parameters $R \in \Pi = \wp(\mathcal{T}_+^0 \cap \mathbb{V})$ (sets of closed positive terms).
  - Ex.: $\{V_1, V_2\}^\perp \otimes (\{V_3\} \oplus X)$ is a formula.
- For A a closed formula one defines a behaviour $|A|$.

Realisability: Behaviours

Back to L_{foc}. Instead of regular expressions that define regular languages: we have closed logical formulae A that define behaviours $|A|$.

- Formulas extended with parameters $R \in \Pi = \wp(\mathcal{T}_+^0 \cap \mathbb{V})$ (sets of closed positive terms).
  - Ex.: $\{V_1, V_2\}^\perp \otimes (\{V_3\} \oplus X)$ is a formula.
- For A a closed formula one defines a behaviour $|A|$.
- Definition such that $|A|^\perp = |A^\perp|$.
- Base cases of the definition:
  - $|R| \stackrel{\text{def.}}{=} R^{\perp\perp}$
  - $|R^\perp| \stackrel{\text{def.}}{=} R^\perp$

Realisability: Behaviour of $\otimes$, $\oplus$, definition

- Case of $\otimes / \parr$:
  - $|A \otimes B| \stackrel{\text{def.}}{=} (|A| \times |B|)^{\perp\perp}$
  - $|A \parr B| \stackrel{\text{def.}}{=} \left(|A^{\perp}| \times |B^{\perp}|\right)^{\perp}$
- Case of $\oplus / \&$:
  - $|A \oplus B| \stackrel{\text{def.}}{=} (|A| + |B|)^{\perp\perp}$
  - $|A \& B| \stackrel{\text{def.}}{=} \left(|A^{\perp}| + |B^{\perp}|\right)^{\perp}$

Realisability: Adequacy lemma

"Proof systems build terms that belong to the behaviours of their types."

Theorem

Suppose c *typable in* $\{\mathsf{LK}_{pol}, \mathsf{LL}\ldots\}$ *of type* $\vdash \kappa_1 : A_1, \ldots, \kappa_n : A_n$.

Then

$$\forall i,\ t_i \in |A_1{}^{\perp}| \implies c\left[\overrightarrow{t_i}\middle/\overrightarrow{\kappa_i}\right] \in \perp\!\!\!\perp$$

In particular

$$\vdash t : A \implies t \in |A|$$

(formulae are all closed)

Realisability: Adequacy lemma

Proof.
(Positive case).
Case $\vdash \mu x.c : N; \Gamma$. This comes from $c : (x : N; \Gamma)$. One has:

$$\langle \mu x.c \parallel t_+ \rangle \to c\,[t_+/x]$$

only when t_+ is a value! $\square$

Realisability: Adequacy lemma

Proof.

(Positive case).

Case $\vdash \mu x.c : N; \Gamma$. This comes from $c : (x : N; \Gamma)$. One has:

$$\langle \mu x.c \parallel t_+ \rangle \to c\,[t_+/x]$$

only when t_+ is a value! $\square$

We need the:

Fact

(Generation) Behaviours are generated by the set of their values.

$$|A|_{\mathbb{V}}{}^{\perp\perp} = |A|$$

Application 1: Head normalisation

Theorem

If $\vdash t : P$ *then* $\langle t \parallel tp \rangle \longrightarrow_h^* \langle V \parallel tp \rangle$ *for some value* V.

Application 1: Head normalisation

Theorem

If $\vdash t : P$ *then* $\langle t \parallel tp \rangle \rightarrow_h^* \langle V \parallel tp \rangle$ *for some value* V.

Proof.

- Take $\perp\!\!\!\perp = \{\, c \mid \exists V \text{ value}, c \rightarrow_h^* \langle V \parallel \mathrm{tp} \rangle \,\}$

Application 1: Head normalisation

Theorem

If $\vdash t : P$ *then* $\langle t \parallel tp \rangle \rightarrow_h^* \langle V \parallel tp \rangle$ *for some value* V.

Proof.

- Take $\perp\!\!\!\perp = \{ c \mid \exists V \text{ value}, c \rightarrow_h^* \langle V \parallel \mathrm{tp} \rangle \}$
- One has $t \in |P|$ (Adequacy Lemma).

Application 1: Head normalisation

Theorem

If $\vdash t : P$ *then* $\langle t \parallel tp \rangle \to_h^* \langle V \parallel tp \rangle$ *for some value* V.

Proof.

- Take $\perp\!\!\!\perp = \{ c \mid \exists V \text{ value}, c \to_h^* \langle V \parallel \mathrm{tp} \rangle \}$
- One has $t \in |P|$ (Adequacy Lemma).
- One has $\mathrm{tp} \in {|P|_{\mathbb{V}}}^{\perp} = |P^{\perp}|$ (Generation theorem).

Hence $\langle t \parallel \mathrm{tp} \rangle \in \perp\!\!\!\perp$.

$\square$

Application 2: Disjunction property

Theorem

If $\vdash t : A \oplus B$ *then* $\langle t \parallel tp \rangle \longrightarrow_h^* \langle \imath_i(V) \parallel tp \rangle$ *for some* $i \in \{1, 2\}$ *and some value* V.

Application 2: Disjunction property

Theorem

If $\vdash t : A \oplus B$ *then* $\langle t \,\|\, tp \rangle \to_h^* \langle \imath_i(V) \,\|\, tp \rangle$ *for some* $i \in \{1,2\}$ *and some value* V.

Proof.

- Take $\perp\!\!\!\perp = \{\, c \mid \exists i, \exists V \text{ value}, c \to_h^* \langle \imath_i(V) \,\|\, \mathtt{tp} \rangle \,\}$.

Application 2: Disjunction property

Theorem

If $\vdash t : A \oplus B$ *then* $\langle t \,\|\, tp \rangle \rightarrow_h^* \langle \imath_i(V) \,\|\, tp \rangle$ *for some* $i \in \{1, 2\}$ *and some value* V.

Proof.

- Take $⫫ = \{ c \mid \exists i, \exists V \text{ value}, c \rightarrow_h^* \langle \imath_i(V) \,\|\, \mathrm{tp} \rangle \}$.
- One has $t \in |A \oplus B|$ (Adequacy Lemma).
- One has $\mathrm{tp} \in |A^\perp \,\&\, B^\perp|$ (Generation theorem).

Hence $\langle t \,\|\, \mathrm{tp} \rangle \in ⫫$.

$\square$

Application 2: Disjunction property

Theorem

If $\vdash t : A \oplus B$ *then* $\langle t \parallel tp \rangle \to_h^* \langle \imath_i(V) \parallel tp \rangle$ *for some* $i \in \{1, 2\}$ *and some value* V.

Proof.

- Take $\perp\!\!\!\perp = \{\, c \mid \exists i, \exists V \text{ value}, c \to_h^* \langle \imath_i(V) \parallel \mathrm{tp} \rangle \,\}$.
- One has $t \in |A \oplus B|$ (Adequacy Lemma).
- One has $\mathrm{tp} \in |A^\perp \,\&\, B^\perp|$ (Generation theorem).

Hence $\langle t \parallel \mathrm{tp} \rangle \in \perp\!\!\!\perp$.

$\square$

- Generalises to any positive type.
- A form of type safety without resorting to subject reduction!

Datatypes

- Suppose:

$$\langle t \,\|\, \alpha \rangle \to_h^* \langle \imath_i(V) \,\|\, \alpha \rangle$$

  Not a real disjunction property if $\alpha \in \mathcal{FV}(V)$!

Datatypes

- Suppose:
$$\langle t \parallel \alpha \rangle \to_h^* \langle \imath_i(V) \parallel \alpha \rangle$$
Not a real disjunction property if $\alpha \in \mathcal{FV}(V)$!

- Particular case of constructivity: Hereditarily positive formulae [Girard, 1992], e.g.:
$$\mathrm{Bool} \stackrel{\text{def.}}{=} \mathbf{1} \oplus \mathbf{1}$$
One has:
$$|\mathbf{1} \oplus \mathbf{1}| = \{\imath_1(), \imath_2()\}^{\perp\perp}$$

Datatypes

- Suppose:
$$\langle t \parallel \alpha \rangle \rightarrow_h^* \langle \imath_i(V) \parallel \alpha \rangle$$
Not a real disjunction property if $\alpha \in \mathcal{FV}(V)$!

- Particular case of constructivity: Hereditarily positive formulae [Girard, 1992], e.g.:
$$\text{Bool} \stackrel{\text{def.}}{=} \mathbf{1} \oplus \mathbf{1}$$
One has:
$$|\mathbf{1} \oplus \mathbf{1}| = \{\imath_1(), \imath_2()\}^{\perp\perp}$$

- A kind of storage theorem without the need for storage operators.
Indeed, one has:
$$\lambda x.x \Vdash (\{\imath_1(), \imath_2()\} \rightarrow A) \rightarrow (\text{Bool} \rightarrow A)$$

Application: Issues with quantification

Simple method to add a new feature in the language:

- Caracterise the feature in terms of behaviours
- Ensure these behaviours are generated by their values.

Application: Issues with quantification

Simple method to add a new feature in the language:

- Caracterise the feature in terms of behaviours
- Ensure these behaviours are generated by their values.

Both steps are modular: we only have to check if the new feature is compatible with a generic notion of computation in LK_{pol}.

- Example of universal quantification / polymorphism.

Application 3: Issues with quantification

- If we try to define the behaviour of quantification like this:

$$|\forall X\ A| = \bigcap_{R \in \Pi} |A[R/X]|$$

then the generation theorem fails as the above behaviour is not generated by its values.

Application 3: Issues with quantification

- If we try to define the behaviour of quantification like this:

$$|\forall X\ A| = \bigcap_{R \in \Pi} |A[R/X]|$$

then the generation theorem fails as the above behaviour is not generated by its values.

2 solutions:

- Introduce a shift (Girard's method)
- Introduce an explicit *value* restriction (Polymorphism *à la* ML):

$$|\mathbb{A}X\ A| = \left(\bigcap_{R \in \Pi} |A[R/X]|_{\mathbb{V}} \right)^{\perp\perp}$$

Application 4: Parametricity

Example

If $\vdash t : \forall X (X \otimes X \to X \otimes X)$
then $\langle t \parallel \{(V_1, V_2) \cdot \mathtt{tp}\}\rangle \to_h^* \langle (V_i, V_j) \parallel \mathtt{tp}\rangle$
for any positive values V_i, V_j and for some $i, j \in \{1, 2\}$.

Application 4: Parametricity

Example

If $\vdash t : \forall X(X \otimes X \to X \otimes X)$
then $\langle t \parallel \{(V_1, V_2) \cdot \mathrm{tp}\}\rangle \to_h^* \langle (V_i, V_j) \parallel \mathrm{tp}\rangle$
for any positive values V_i, V_j and for some $i, j \in \{1, 2\}$.

Proof.

Let V_1, V_2 be positive values.

- $\perp\!\!\!\perp = \left\{ c \;\middle|\; \exists i, j \in \{1, 2\}, c \to_h^* \langle (V_i, V_j) \parallel \mathrm{tp}\rangle \right\}$.
- With $R = \{V_1, V_2\}$ as a parameter one derives $\langle t \parallel \{(V_1, V_2) \cdot \alpha\}\rangle : (\vdash \alpha : R \otimes R)$.
- One has $\mathrm{tp} \in |R \otimes R|^\perp$.

Hence $\langle t \parallel \{(V_1, V_2) \cdot \mathrm{tp}\}\rangle \in \perp\!\!\!\perp$ by the adequacy lemma.

$\square$

Conclusion

Proximity of Classical Realizability with Ludics:

- There is a daimon:

$$\frac{}{c_0 : (\vdash \_ : A_1, \ldots, \_ : A_n)} \maltese \text{ (when } c_0 \in \perp\!\!\!\perp)$$

- Daimon implies internal completeness of the connectives:

$$|A \otimes B|_{\mathbb{V}} = |A|_{\mathbb{V}} \times |B|_{\mathbb{V}}$$

  under some generic conditions.
- Decomposition of the universal quantification under the form $\mathbb{A}X \uparrow A = \forall X\ A$ where $\mathbb{A}$ enjoys "shocking equalities":

$$|\mathbb{A}X\ (A \oplus B)| = |(\mathbb{A}X\ A) \oplus (\mathbb{A}X\ B)|$$
$$|\mathbb{A}X\ (A \otimes B)| = |(\mathbb{A}X\ A) \otimes (\mathbb{A}X\ B)|$$

Conclusion

L_{foc} gives a very simple and non-bureaucratic account of various trends of proof theory:

- Syntaxes for sequent calculi [Curien-Herbelin] and the duality of computation
- Focalisation [Andreoli, Girard]
- Classical realisability [Krivine]

...very close to CS (values, distinction strict/lazy, analogy with automata).

End

Thanks to Pierre-Louis Curien, Hugo Herbelin, Stephen Zdancewic, Jeffrey Vaughan & the anonymous referees for comments on this work.

Guillaume Munch–Maccagnoni. *Focalisation and classical realisability.* To appear in the proceedings of CSL'09.