Computational Ludics

Kazushige Terui

National Institute of Informatics, Tokyo

Laboratoire d'Informatics de Paris Nord, CNRS

email: terui@nii.ac.jp

What is ludics?

- Ludics (Girard 01): pre-logical framework upon which logic is built and various phenomena are analyzed.
- Keywords: Monism, existentialism, interaction/orthogonality:

Game Semantics	$\Longleftrightarrow$	Ludics	$\Longleftrightarrow$	Proof Theory
strategies		designs		proofs
$\Uparrow$		$\Downarrow$orthogonality		$\Uparrow$
arenas		behaviours		types

- Goal: Logical reconstruction of computability and complexity theory based on ludics

Why ludics?

- Ingredients of computability theory

Alphabet	Σ
Words	$w \in \Sigma^*$
Languages	$L \subseteq \Sigma^*$
Language classes	$\mathcal{C} \subseteq 2^{\Sigma^*}$

Why ludics?

- Ingredients of computability theory correspond in logic to

Alphabet	Σ	Logical rules
Words	$w \in \Sigma^*$	Proofs
Languages	$L \subseteq \Sigma^*$	Sets of proofs
Language classes	$\mathcal{C} \subseteq 2^{\Sigma^*}$	(Restricted) proof systems

Why ludics?

- Ingredients of computability theory correspond in logic to

Alphabet	Σ	Logical rules
Words	$w \in \Sigma^*$	Proofs
Languages	$L \subseteq \Sigma^*$	Sets of proofs
Language classes	$\mathcal{C} \subseteq 2^{\Sigma^*}$	(Restricted) proof systems

- How to specify a language / a set of proofs?
  - Via typing: $\{\pi \; : \quad \vdash \pi : E\}$
    static, cf. regular expressions $a + ab^*a$
  - Via normalization: $\{\pi \; : \; \sigma(\pi) \Downarrow \pi_{\mathsf{accept}}\}$
    dynamic, cf. finite automata

Why ludics?

- Ludics is endowed with a canonical notion of acceptance:

  For any closed net π, either $\pi \Downarrow \maltese$ or $\pi \Uparrow$

- Orthogonality:

$$\sigma \perp \pi \iff \sigma(\pi) \Downarrow \maltese$$

- $\sigma^{\perp}$ = the language accepted by σ.

Alphabet	Σ	Actions
Words	$w \in \Sigma^*$	Designs
Languages	$L \subseteq \Sigma^*$	Behaviours
Language classes	$\mathcal{C} \subseteq 2^{\Sigma^*}$	Restriction on σ

- Regular expressions vs. Finite automata

  $\Longrightarrow$ Typing vs. Interaction

Computational ludics

- We reformulate the original ludics. Why?

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

  $\Rightarrow$ We introduce a term calculus (following Curien's concrete syntax)

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

    $\Rightarrow$ We introduce a term calculus (following Curien's concrete syntax)
  - Designs are infinite, while algorithms must be finitely presented.

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

    $\Rightarrow$ We introduce a term calculus (following Curien's concrete syntax)
  - Designs are infinite, while algorithms must be finitely presented.

    $\Rightarrow$ We introduce design generators that give finite descriptions to some infinite designs.

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

    $\Rightarrow$ We introduce a term calculus (following Curien's concrete syntax)
  - Designs are infinite, while algorithms must be finitely presented.

    $\Rightarrow$ We introduce design generators that give finite descriptions to some infinite designs.
  - Girard's designs are cut-free and identity-free. Lack of computational power.

Computational ludics

- We reformulate the original ludics. Why?
  - Working with absolute addresses (loci) is like programming with machine codes.

    ⇒ We introduce a term calculus (following Curien's concrete syntax)
  - Designs are infinite, while algorithms must be finitely presented.

    ⇒ We introduce design generators that give finite descriptions to some infinite designs.
  - Girard's designs are cut-free and identity-free. Lack of computational power.

    ⇒ We incorporate cuts and identities into designs.

Part I: Architecture of ludics

Behaviours:	semantic types, reducibility candidates,
$\Uparrow$	
Designs:	proofs, strategies, processes
$\Uparrow$	
Generators:	proof search instructions

Well-behaved frag. of simply typed λ-calculus

- Types: $\tau ::= \iota \mid \tau \to \tau$
- Positive terms P and negative terms N are defined by:

$$\begin{array}{rcl} P^{\iota} & ::= & (N_0^{\tau_1 \to \dots \tau_n \to \iota}) N_1^{\tau_1} \dots N_n^{\tau_n} \\ N^{\tau_1 \to \dots \tau_n \to \iota} & ::= & x \mid \lambda x_1^{\tau_1} \cdots x_n^{\tau_n}.P^{\iota} \end{array}$$

- Reduction: the arity n always agrees.

$$(\lambda x_1 \cdots x_n.P) N_1 \cdots N_n \longrightarrow P[N_1/x_1, \dots, N_n/x_n]$$

- $(N_0)N_1 \cdots N_n$ is a redex if N_0 is not a variable.
- $(N_0)N_1 \cdots N_n$ is η-expandible if some N_i (of non-atomic type) is a variable x.

Towards ludics

- Designs in ludics:
  - Type-free; arity agreement is ensured in another way.
  - Infinitary (coinduitive).
  - Various actions (rather than the single pair λ/@)
  - Daimon (immediate termination)
  - Additive superimposition: $N_1 + N_2 + N_3 + \cdots$

Towards ludics

- Girard/Curien's original designs:
  - Extensions of normal and η-long lambda terms.
  - Actions built from ramifications $I \in \mathcal{P}_f(\mathcal{N})$.
- Our designs:
  - Extensions of arbitrary terms
  - Actions built from a given signature.
- Signature $\mathcal{A} = (A, ar)$:

  A is a set of names,

  $ar : A \longrightarrow \mathcal{N}$ gives an arity to each name.

Computational designs

- The sets of c-designs are coinductively defined by:

$$
\begin{array}{rcll}
P & ::= & \maltese & \text{Daimon} \\
 & | & \Omega & \text{Divergence} \\
 & | & N_0|\overline{a}\langle N_1, \ldots, N_n\rangle & \text{Proper positive action} \\
N & ::= & x & \text{Variable} \\
 & | & \sum a(\vec{x}_a).P_a & \text{Proper negative action}
\end{array}
$$

  - where $ar(a) = n$, $\vec{x}_a = x_1, \ldots, x_n$
  - $\sum a(\vec{x}_a).P_a$ is built from $\{a(\vec{x}_a).P_a\}_{a \in A}$. Compare it with:

$$
\begin{array}{rcl}
P & ::= & (N_0)N_1 \ldots N_n \\
N & ::= & x \mid \lambda x_1 \cdots x_n.P
\end{array}
$$

Computational designs

- $N_0|\overline{a}\langle N_1, \ldots, N_n\rangle$ is a cut if N_0 is not a variable.
- x is an identity if it occurs as $N_0|\overline{a}\langle N_1, \ldots, x, \ldots, N_n\rangle$
- Reduction rule:

$$(\sum a(\vec{x}_a).P_a)\,|\overline{a}\langle N_1, \ldots, N_n\rangle \;\longrightarrow\; P_a[N_1/x_1, \ldots, N_n/x_n].$$

- Compare it with

$$(\lambda x_1 \cdots x_n.P)N_1 \cdots N_n \;\longrightarrow\; P[N_1/x_1, \ldots, N_n/x_n]$$

Standard c-designs

- C-designs have to be identifed up to α-equivalence.
- P is total if $P \neq \Omega$.
- T is linear if for any subterm $N_0|a\langle N_1, \ldots, N_n\rangle$, $fv(N_0)$, …, $fv(N_n)$ are pairwise disjoint.
- T is standard if it is linear, total, cut-free, identity-free and has finitely many free variables.
- Fact: If P is standard, then $P = \maltese$ or $P = x|\overline{a}\langle N_1, \ldots, N_n\rangle$ where none of $N_1, \ldots, N_n$ is a variable.
- Fact: The standard c-designs over the signature $(\mathcal{P}_f(\mathcal{N}), |\ \ |)$ exactly correspond to Girard's original designs.

Architecture of ludics: computation

Behaviours:	Orthogonality
$\Uparrow$	
Designs:	Reduction-based normalization
$\Uparrow$	
Generators:	Krivine's abstract machines

Normalization

- Reduction rule: $(\sum a(\vec{x}_a).P_a) \,|\overline{a}\langle \vec{N}_a \rangle \longrightarrow P_a[\vec{N}_a/\vec{x}]$.
- $P \Downarrow Q$ if $P \longrightarrow^* Q$ and Q is neither a cut nor Ω.
- By corecursion, it can be extended to $[\![\ ]\!]$:

$$
\begin{array}{rcll}
[\![P]\!] & = & \maltese & \text{if } P \Downarrow \maltese; \\
 & = & x|\overline{a}\langle [\![N_1]\!], \ldots, [\![N_n]\!] \rangle & \text{if } P \Downarrow x|\overline{a}\langle N_1, \ldots, N_n \rangle; \\
 & = & \Omega & \text{if } P \Uparrow; \\
[\![x]\!] & = & x; & \\
[\![\sum a(\vec{x}_a).P_a]\!] & = & \sum a(\vec{x}_a).[\![P_a]\!]. &
\end{array}
$$

- Non-effective: it works on infinite designs; renaming and substitution involved.

What are data?

- Examples: integers, words, trees, lists, records, etc.
- Data must be:
  - structured (eg. list = head + tail)
  - linearly duplicable (“linear” = “machine-like”)
  - compressable (eg. binary int. $\rightarrow$ hexadecimal int.)
- Fix a unary name $\uparrow \in A$.
- The set of data designs is coinductively defined by

$$d \; ::= \uparrow(x).x|\overline{a}\langle d, \ldots, d\rangle, \qquad a \in A.$$

- Notation: $\downarrow = \overline{\uparrow}$, $\qquad \uparrow\overline{a}\langle\vec{N}\rangle = \uparrow(x).x|\overline{a}\langle\vec{N}\rangle$

Data: examples

- Natural numbers

$$\begin{aligned} 0^\star &= \uparrow\overline{\mathsf{zero}} \\ n+1^\star &= \uparrow\overline{\mathsf{suc}}\langle n^\star\rangle \end{aligned}$$

- Ordinals

$$\omega^\star = \uparrow\overline{\mathsf{suc}}\langle\omega^\star\rangle.$$

- Words, labelled binary trees, and lists:

$$\begin{aligned} \epsilon^\star &= \uparrow\overline{\mathsf{nil}}, & \mathtt{leaf}_i^\star &= \uparrow\overline{\mathsf{leaf}}_i, \\ (iw)^\star &= \uparrow\overline{\mathsf{suc}}_i\langle w^\star\rangle, & (\mathtt{node}_i(t,u))^\star &= \uparrow\overline{\mathsf{node}}_i\langle t^\star, u^\star\rangle, \\ []^\star &= \uparrow\overline{\mathsf{nil}}; \\ (d :: l)^\star &= \uparrow\overline{\mathsf{cons}}\langle d, l^\star\rangle. \end{aligned}$$

Functions on Data

- Discriminators. Given N_a (a variable $x_i \in \{\vec{x}_a\}$ or $\uparrow(y).P_a$) for each $a \in K$,

$$[x] \sum_K a(\vec{x}_a) \rhd N_a \; = \uparrow(y).x|\downarrow\langle \sum_K a(\vec{x}_a).N_a|\downarrow\langle y\rangle\rangle.$$

- Given $d = \uparrow\overline{a}\langle d_1, \ldots, d_n\rangle$ with $a \in K$,

$$\begin{array}{rcl}[d] \sum_K a(\vec{x}_a) \rhd N_a & = & \uparrow(y).d|\downarrow\langle \sum_K a(\vec{x}_a).N_a|\downarrow\langle y\rangle\rangle \\ & \longrightarrow & \uparrow(y).\left(\sum_K a(\vec{x}_a).N_a|\downarrow\langle y\rangle\right)|\overline{a}\langle d_1, \ldots, d_n\rangle \\ & \longrightarrow & \uparrow(y).(N_a[d_1/x_1, \ldots, d_n/x_n]|\downarrow\langle y\rangle) \\ & \longrightarrow & N_a[d_1/x_1, \ldots, d_n/x_n],\end{array}$$

- Predecessor: $Pred[x] = [x](\text{zero} \rhd 0^\star + \text{suc}(z) \rhd z)$.

Functions on Data

- Duplicator. A linear c-design $Dup[x]$ s.t. for any finite datum d,

$$[\![Dup[d]]\!] = \uparrow \overline{\mathsf{pair}}(d, d).$$

- Cf. Linear duplicator in λ-calculus

$$\begin{array}{llllll}
Dup_{\mathsf{B}}(x) & = & case & x = \mathsf{true} & \Rightarrow & \mathsf{true} \otimes \mathsf{true} \\
 & & & x = \mathsf{false} & \Rightarrow & \mathsf{false} \otimes \mathsf{false} \\
Dup_{\mathsf{N}}(x) & = & case & x = \mathsf{zero} & \Rightarrow & \mathsf{zero} \otimes \mathsf{zero} \\
 & & & x = \mathsf{suc}(y) & \Rightarrow & let\ z_1 \otimes z_2 = Dup_{\mathsf{N}}(y) \\
 & & & & & \quad in\ \ \mathsf{suc}(z_1) \otimes \mathsf{suc}(z_2)
\end{array}$$

- Cut is essential for finite generation.
- Q: Does Dup duplicate $\omega^\star$?

Functions on Data

- General recursion scheme: Given a linear c-design H_a for each $a \in K$, there exists a linear c-design F:

$$[\![F[\uparrow\overline{a}\langle\vec{d}\rangle, \vec{e}]]\!] = [\![H_a[F[d_1, \vec{e}], \ldots, F[d_n, \vec{e}], \vec{e}]]\!]$$

for every $a \in K$ and finite data $\vec{e}$.

Architecture of ludics: computation

Behaviours:	Orthogonality
$\Uparrow$	
Designs:	Reduction-based normalization
$\Uparrow$	
Generators:	Krivine's abstract machines

Design generators

- Generator: $G = (S^+, S^-, \ell)$, where S^+ and S^- are disjoint sets of states, ℓ is a function on $S = S^+ \cup S^-$:
  - $\ell(s^+)$ = ✠, Ω or $s_0^- | \overline{a}\langle s_1^-, \ldots, s_n^- \rangle$
  - $\ell(s^-)$ = x or $\sum a(\vec{x}_a).s_a^+$
- Generators are like designs, but may contain loops. A design is obtained by unfolding a generator.
- Krivine's abstract machine directly works on generators
  $\Longrightarrow$ Effective computation

Architecture of ludics: computation

Behaviours:	Orthogonality
$\Uparrow$	
Designs:	Reduction-based normalization
$\Uparrow$	
Generators:	Krivine's abstract machines

Orthogonality

- From now on, we considery only linear c-designs.
- P is closed if it has no free variables: either $\maltese$ or a cut.
- P is atomic if $FV(P) \subseteq \{x_0\}$. N is atomic if $FV(N) = \emptyset$.
- For atomic P and N, $P \perp N$ if $P[N/x_0] \Downarrow \maltese$.
- Behaviour: a set $\mathbf{T}$ of total linear c-designs of the same polarity such that

$$\mathbf{T}^{\perp\perp} = \mathbf{T}.$$

Analytical theorems

- Associativity:
  $[\![T[N_1/y_1,\ldots,N_n/y_n]\,]\!] = [\![\,[\![T]\!][\,[\![N_1]\!]/y_1,\ldots,[\![N_n]\!]/y_n]\,]\!]$.
- Stability: If $\{T_i\}_{i\in\Lambda}$ are compatible, then $[\![\bigcap_{i\in\Lambda} T_i]\!] = \bigcap_{i\in\Lambda}[\![T_i]\!]$.
- η-long normal form: $[\![T]\!]_\eta$ is $[\![T]\!]$ with identity x replaced by fax

$$\eta(x) = \sum a(y_1,\ldots,y_n).x|\overline{a}\langle\eta(y_1),\ldots,\eta(y_n)\rangle.$$

- Separation: $[\![T]\!]_\eta \preceq [\![U]\!]_\eta$ if and only if $T^\perp \subseteq U^\perp$.
- Only up to "$\beta\eta$-equivalence."

Architecture of ludics: computation

Behaviours:	Orthogonality
$\Uparrow$	
Designs:	Reduction-based normalization
$\Uparrow$	
Generators:	Krivine's abstract machines

What are logical connectives?

- Requirements for logical connectives:
  - Have a dual
  - Admit internal completeness
  - Induce behaviour constructors (semantics) and logical inference rules (syntax)
- n-ary logical connective $\alpha = \{a(\vec{x}_a)\}_{a \in K}$ such that K is finite and $\{\vec{x}_a\} \subseteq \{x_1, \ldots, x_n\}$.
- Examples: $\& = \{\pi_1(x_1), \pi_2(x_2)\}$, $⅋ = \{\wp(x_1, x_2)\}$.

Logical connectives

- Behaviours built by logical connectives:

$$
\begin{aligned}
\overline{\alpha}\langle \mathbf{N}_1, \ldots, \mathbf{N}_n \rangle &= \Big(\bigcup_{a(\vec{x}_a)\in\alpha} \overline{a}\langle \mathbf{N}_{i_1}, \ldots, \mathbf{N}_{i_m} \rangle \Big)^{\perp\perp}; \\
\alpha(\mathbf{P}_1, \ldots, \mathbf{P}_n) &= (\overline{\alpha}\langle \mathbf{P}_1^{\perp}, \ldots, \mathbf{P}_n^{\perp} \rangle)^{\perp};
\end{aligned}
$$

where $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$,

$$
\overline{a}\langle \mathbf{M}_1, \ldots, \mathbf{M}_m \rangle = \{ x_0 | \overline{a}\langle M_1, \ldots, M_m \rangle : M_k \in \mathbf{M}_k, 1 \leq k \leq m \}
$$

Logical connectives

- Examples: $\& = \{\pi_1(x_1), \pi_2(x_2)\}$, $\parr = \{\wp(x_1, x_2)\}$.
- Let $\oplus = \overline{\&}$, $\iota_i = \overline{\pi}_i$, $\otimes = \overline{\parr}$, and $\bullet = \overline{\wp}$.

$$
\begin{array}{rcl}
\oplus\langle \mathbf{N}, \mathbf{M} \rangle & = & (\iota_1\langle \mathbf{N} \rangle \cup \iota_2\langle \mathbf{M} \rangle)^{\perp\perp} \\
\otimes\langle \mathbf{N}, \mathbf{M} \rangle & = & \bullet\langle \mathbf{N}, \mathbf{M} \rangle^{\perp\perp} \\
\&(\mathbf{P}, \mathbf{Q}) & = & (\iota_1\langle \mathbf{P}^\perp \rangle)^\perp \cap (\iota_2\langle \mathbf{Q}^\perp \rangle)^\perp \\
\parr(\mathbf{P}, \mathbf{Q}) & = & (\bullet\langle \mathbf{P}^\perp, \mathbf{Q}^\perp \rangle)^\perp
\end{array}
$$

Internal completeness

- Surface incarnation.
  $|\mathbf{P}|_h$ consists of 'head normal' I-designs $x_0|\overline{a}\langle\vec{M}\rangle$ in $\mathbf{P}$
  $|\mathbf{N}|_\alpha$ consists of 'head optimal' I-designs $\sum_\alpha a(\vec{x}_a).P_a$ in $\mathbf{N}$.
- Internal completeness theorem:
  1. $|\overline{\alpha}\langle\mathbf{N}_1,\ldots,\mathbf{N}_n\rangle|_h = \bigcup_{a(\vec{x}_a)\in\alpha}\overline{a}\langle\mathbf{N}_{i_1},\ldots,\mathbf{N}_{i_m}\rangle$.
  2. $|\alpha(\mathbf{P}_1,\ldots,\mathbf{P}_n)|_\alpha = \sum_\alpha a(\vec{x}_a).[\mathbf{P}_{i_1}^\perp/x_{i_1},\ldots,\mathbf{P}_{i_m}^\perp/x_{i_m}]^\perp$.
- Examples:

$$
\begin{array}{rcl}
|\oplus\langle\mathbf{N},\mathbf{M}\rangle|_h & = & \iota_1\langle\mathbf{N}\rangle\cup\iota_2\langle\mathbf{M}\rangle \\
|\otimes\langle\mathbf{N},\mathbf{M}\rangle|_h & = & \bullet\langle\mathbf{N},\mathbf{M}\rangle \\
|\&\langle\mathbf{P},\mathbf{Q}\rangle|_\& & = & \pi_1(x_0).\mathbf{P}+\pi_2(x_0).\mathbf{Q} \\
|\invamp\langle\mathbf{P},\mathbf{Q}\rangle|_\invamp & = & \invamp(x_1,x_2).[\mathbf{P}^\perp/x_1,\mathbf{Q}^\perp/x_2]^\perp
\end{array}
$$

Part II: Some topics

1. Data specification
2. Designs and automata
3. Types and completeness
4. WIP: Unique interpretation
5. WIP: Nondeterminism in ludics
6. WIP: Focalization

Data specification via interaction

- Given a set $\mathbf{D}$ of data designs, there are two ways to specify $\mathbf{D}$
  1. Via typing: Find a syntactic type D s.t.

  $$\vdash d : \mathsf{D} \iff d \in \mathbf{D},$$

  2. Via interaction: Find a c-design P s.t.

  $$P \perp d \iff d \in \mathbf{D},$$

     i.e., $||P^{\perp}|| = \mathbf{D}$.
     ($||P^{\perp}||$ consists of material ✠-free designs in $P^{\perp}$)
- In the latter case, we say $\mathbf{D}$ is accepted by P.
- Theorem: Any (possibly infinite) set of finite data can be accepted.

Data specification via interaction

- Theorem: Any set of finite data can be accepted.
- Proof idea:
  - Given a datum d, build a counter design d^c such that

$$d^c \perp e \iff e = d$$

    for every datum e (Cf. Faggian 05).
  - Any d_1^c and d_2^c are compatible.
  - $P = \bigcup \{d^c : d \in \mathbf{D}\}$ accepts $\mathbf{D}$:

$$P \perp e \iff e \in \mathbf{D}.$$

- P is not finitely generated.

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$
\begin{aligned}
Q_I &= x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\
Q_F &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\
Q_\Omega &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle
\end{aligned}
$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$
\begin{aligned}
Q_I &= x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\
Q_F &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\
Q_\Omega &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle
\end{aligned}
$$

$$
Q_I[(aba)^\star/x] = Q_I[{\uparrow}\overline{a}\langle(ba)^\star\rangle/x]
$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$
\begin{aligned}
Q_I &= x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\
Q_F &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\
Q_\Omega &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle
\end{aligned}
$$

$$
\begin{aligned}
Q_I[(aba)^\star/x] &= Q_I[{\uparrow}\overline{a}\langle (ba)^\star\rangle/x] \\
&\longrightarrow^* Q_F[(ba)^\star/x]
\end{aligned}
$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$\begin{array}{rcl} Q_I & = & x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\ Q_F & = & x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\ Q_\Omega & = & x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \end{array}$$

$$\begin{array}{rcl} Q_I[(aba)^\star/x] & = & Q_I[\uparrow\overline{a}\langle(ba)^\star\rangle/x] \\ & \longrightarrow^* & Q_F[(ba)^\star/x] \\ & \longrightarrow^* & Q_I[(a)^\star/x] \end{array}$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$
\begin{array}{rcl}
Q_I & = & x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\
Q_F & = & x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\
Q_\Omega & = & x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle
\end{array}
$$

$$
\begin{array}{rcl}
Q_I[(aba)^\star/x] & = & Q_I[{\uparrow}\overline{a}\langle(ba)^\star\rangle/x] \\
 & \longrightarrow^* & Q_F[(ba)^\star/x] \\
 & \longrightarrow^* & Q_I[(a)^\star/x] \\
 & \longrightarrow^* & Q_F[{\uparrow}\overline{\mathsf{nil}}/x]
\end{array}
$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$
\begin{aligned}
Q_I &= x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\
Q_F &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\
Q_\Omega &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle
\end{aligned}
$$

$$
\begin{aligned}
Q_I[(aba)^\star/x] &= Q_I[\uparrow\overline{a}\langle(ba)^\star\rangle/x] \\
&\longrightarrow^* Q_F[(ba)^\star/x] \\
&\longrightarrow^* Q_I[(a)^\star/x] \\
&\longrightarrow^* Q_F[\uparrow\overline{\mathsf{nil}}/x] \\
&\longrightarrow^* \maltese
\end{aligned}
$$

Designs and automata

- Finite automaton M corresponds to design Q_I:

$$\begin{aligned} Q_I &= x|{\downarrow}\langle a(x).Q_F + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \\ Q_F &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_I + \mathsf{nil}.\maltese\rangle \\ Q_\Omega &= x|{\downarrow}\langle a(x).Q_\Omega + b(x).Q_\Omega + \mathsf{nil}.\Omega\rangle \end{aligned}$$

$$\begin{aligned} Q_I[(aba)^\star/x] &= Q_I[{\uparrow}\overline{a}\langle(ba)^\star\rangle/x] \\ &\longrightarrow^* Q_F[(ba)^\star/x] \\ &\longrightarrow^* Q_I[(a)^\star/x] \\ &\longrightarrow^* Q_F[{\uparrow}\overline{\mathsf{nil}}/x] \\ &\longrightarrow^* \maltese \end{aligned}$$

- $Q_I \perp d \iff d = (ab)^n a, \qquad n \geq 0.$

Acceptance via finitely generated designs

- Theorem: For any language $\mathbf{L}$,
  1. $\mathbf{L}$ is accepted by a finitely generated standard design $\iff \mathbf{L}$ is regular.
  2. $\mathbf{L}$ is accepted by a finitely generated design $\iff \mathbf{L}$ is r.e.
- Proof idea of 1: Cut-free designs $\sim$ finite automata
  Restriction to languages is essential.
- Proof idea of 2.
  $\Rightarrow$ Krivine's machine works effectively on finite generators.
  $\Leftarrow$ Constructors, discriminators, general recursion schema are available.

Types and completeness

- The types are coinductively defined by:

$$\begin{aligned} \mathsf{P} &::= \overline{\alpha}\langle \mathsf{N}_1, \ldots, \mathsf{N}_n \rangle \\ \mathsf{N} &::= \alpha(\mathsf{P}_1, \ldots, \mathsf{P}_n) \end{aligned}$$

$$\begin{tikzcd} & \text{Type T} \arrow[dl, "\text{interpretation}"'] \arrow[dr, "\text{derivation}"] & \\ \text{Behaviour T}^\bullet & \overset{\text{completeness}}{\Longleftrightarrow} & \text{Proof set T}^\circ \end{tikzcd}$$

- $\overline{\alpha}\langle \mathsf{N}_1, \ldots, \mathsf{N}_n \rangle^\bullet = \overline{\alpha}\langle \mathsf{N}_1^\bullet, \ldots, \mathsf{N}_n^\bullet \rangle$
- $\mathsf{T}^\circ = \{P \mid\ \vdash P : \mathsf{T} \text{ is derivable}\}$

Proof system

- Positive sequent: $P \vdash x_1 : \mathsf{P}_1, \ldots, x_n : \mathsf{P}_n$
- Negative sequent: $N \vdash x_1 : \mathsf{P}_1, \ldots, x_n : \mathsf{P}_n, \mathsf{N}$
- Inference rules:

$$\frac{}{\maltese \vdash}\ (\maltese) \qquad \frac{T \vdash \Theta}{T \vdash \Theta, \Gamma}\ (w)$$

$$\frac{M_1 \vdash \Gamma_1, \mathsf{N}_{i_1} \quad \ldots \quad M_m \vdash \Gamma_1, \mathsf{N}_{i_m}}{z|\overline{a}\langle M_1, \ldots, M_m\rangle \vdash \Gamma_1, \ldots, \Gamma_n, z : \overline{\alpha}\langle \mathsf{N}_1, \ldots, \mathsf{N}_n\rangle}\ (\overline{\alpha}, \overline{a}(\vec{x}_a))$$

$$\frac{\{P_a \vdash \Gamma, \vec{x}_a : \vec{\mathsf{P}}_a\}_{a(\vec{x}_a)\in\alpha}}{\sum_\alpha a(\vec{x}_a).P_a \vdash \Gamma, \alpha(\mathsf{P}_1, \ldots, \mathsf{P}_n)}\ (\alpha)$$

where $\vec{x}_a = x_{i_1}, \ldots, x_{i_m}$, $\vec{\mathsf{P}}_a = \mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_m}$.

- Variable-free polarized MALL can be embedded.

Types and completeness

- Interpretation is compositional: $(\mathsf{P} \parr \mathsf{Q})^\bullet = \mathsf{P}^\bullet \parr \mathsf{Q}^\bullet$
- Derivation is continuous: $Dv(\bigcup^\uparrow \mathcal{S}_i) = \bigcup^\uparrow Dv(\mathcal{S}_i)$, where

  $Dv(\mathcal{S}) = \mathcal{S} \cup \{S_0 : S_0$ is immediately derivable from sequents in $\mathcal{S}\}$.
- Completeness: Continuous construction meets compositional one.
- Interpretations and proof sets are not unique.
  - Least behaviours/proof sets: $\mathsf{T}^\bullet_L$, T°_L
  - Greatest behaviours/proof sets: $\mathsf{T}^\bullet_G$, T°_G
- Full completeness theorem: for every type T,

$$|\mathsf{T}^\bullet_G| = \mathsf{T}^\circ_G, \qquad |\mathsf{T}^\bullet_L| = \mathsf{T}^\circ_L.$$

Types and completeness

- Key observations:
  - Internal completeness implies continuity of logical connectives: For any chain $\mathbf{P}_0 \subseteq \mathbf{P}_1 \subseteq \mathbf{P_2} \cdots$

$$\left(\bigcup_n \mathbf{P}_n\right)^{\perp\perp} \parr \mathbf{Q} = \left(\bigcup_n \mathbf{P}_n \parr \mathbf{Q}\right)^{\perp\perp}$$

  - Syntactic completeness:

$$P \vdash'_L x_0 : \mathsf{P} \iff P \perp N \text{ for all } N \vdash_G \mathsf{P}^\perp$$

  - Syntactic completeness implies "compositionality" of proof sets:

$$(\vdash x : \mathsf{P}, y : \mathsf{Q})^\circ_G \subseteq [\mathsf{P}^{\circ\perp}_G / x, \mathsf{Q}^{\circ\perp}_G / y]^\perp$$

WIP: Unique interpretation?

- Our $\perp$ is defined based on termination:

$$P \perp N \iff P[N/x_0] \longrightarrow \longrightarrow \cdots \text{terminates}$$

- What happens if it is defined on safety?

$$P \perp N \iff P[N/x_0] \longrightarrow \longrightarrow \cdots \text{doesn't deadlock}$$

- Melliès and Vouillon (LICS05) observed (in the context of lambda calculus): if $\perp$ is defined based on safety, then recursive types admit unique interpretation.
- Question: what happens if one does the same on ludics?
- A crucial step towards logical understanding of infinitary automata theory.

WIP: Nondeterminism in ludics

- One can also consider a nondeterministic generator, which generates a set of designs.
- It provides a means of controlled proof search.
- Our slogan: designs are deterministic (as proofs), while generators can be nondeterministic (as proof search).

Behaviours: stable theory

$\Uparrow$

Designs: deterministic

$\Uparrow$

Generators: nondeterministic

WIP: Focalization

- Focalization: $L \otimes (M \oplus N)$ can be considered as a single connective $\oslash(L, M, N)$.
- In ludics: Internal completeness implies

$$|\otimes\langle \mathbf{L}, \uparrow \oplus\langle \mathbf{M}, \mathbf{N}\rangle\rangle| \cong |\oslash\langle \mathbf{L}, \mathbf{M}, \mathbf{N}\rangle| \, .$$

  for any negative behaviours $\mathbf{L}, \mathbf{M}, \mathbf{N},$.
- More generally,

$$\left|\overline{\alpha}\langle \uparrow \overline{\beta}_1\langle \vec{\mathbf{N}}\rangle, \ldots, \uparrow \overline{\beta}_n\langle \vec{\mathbf{N}}\rangle\rangle\right| \cong \left|\overline{\alpha\beta_1 \cdots \beta_n}\langle \vec{\mathbf{N}}\rangle\right|$$

- Together with full completeness, does it lead to another proof of the focalization theorem?

Conclusion

- Our contribution:
  - A handy syntax with cuts and identities. Cuts are important for expressive power, while identities are for efficiency.
  - A full completeness theorem in an infinitary setting.
  - An analysis of data in ludics.
  - An exact characterization of cut-freely acceptable languages.
- It is now time to analyze computability/complexity properties in ludics.