Higher-order parity automata

Paul-André Melliès

Institut de Recherche en Informatique Fondamentale (IRIF)
CNRS – Université de Paris

Cyclic Syntax and Semantics
University of Gothenburg 20 → 22 November 2019
Linear logic

Seen through the lens of game semantics
Starting point: game semantics

Every proof of formula $A$ initiates a dialogue where

Proponent tries to convince Opponent

Opponent tries to refute Proponent

An interactive approach to logic and programming languages
The formal proof of the drinker’s formula

\[
\begin{align*}
A(x_0) & \vdash A(x_0) & \text{Axiom} \\
A(x_0) & \vdash A(x_0), \forall x. A(x) & \\
\vdash A(x_0), A(x_0) \Rightarrow \forall x. A(x) & \\
\vdash A(x_0), \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
\vdash \forall x. A(x), \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
A(y_0) & \vdash \forall x. A(x), \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
\vdash A(y_0) \Rightarrow \forall x. A(x), \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
\vdash \exists y. \{A(y) \Rightarrow \forall x. A(x)\}, \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
\vdash \exists y. \{A(y) \Rightarrow \forall x. A(x)\} & \\
\end{align*}
\]
Duality

Proponent Program plays the game $A$

Opponent Environment plays the game $\neg A$

Negation permutes the rôles of Proponent and Opponent
Duality

Opponent
Environment
plays the game

¬ A

Proponent
Program
plays the game

A

Negation permutes the rôles of Opponent and Proponent
Proponent selects the board which will be played
A form of constructive disjunction
Opponent selects the board which will be played
Product

A form of constructive conjunction
Tensor product

The two games are played in parallel. **Opponent** is allowed to switch board but not **Player**.
Tensor product

A form of classical conjunction
Parallel product

The two games are played in parallel. **Player** is allowed to switch board but not **Opponent**.
Parallel product

A form of classical disjunction
The law of excluded middle

Karpov

Korchnoi

Player wins by playing Karpov against Korchnoi
The exponential modality

\[ \odot \odot \odot \cdots \]

Opponent opens as many copies as necessary to beat Proponent but is not allowed to open an infinite number of copies.

Hence, the modality is \( \begin{cases} \text{coinductive} & \text{from the point of view of Player,} \\ \text{inductive} & \text{from the point of view of Opponent.} \end{cases} \)
A beautiful isomorphism of linear logic

For every pair of formulas $A$ and $B$ of linear logic

$$! A \otimes ! B \cong !(A \& B)$$

reminiscent of the isomorphism

$$\emptyset A \times \emptyset B \cong \emptyset (A + B)$$

This isomorphism is the origin for the name of exponential modality
The functorial approach to proof invariants

Cartesian closed categories
Brouwer - Heyting - Kolmogorov interpretation

A proof of the formula

\[ A \land B \]

is a pair

\( (\varphi, \psi) \)

consisting of a proof

\[ \varphi \]

of the formula \( A \) and of a proof

\[ \psi \]

of the formula \( B \).
Brouwer-Heyting-Kolmogorov interpretation

A proof of the formula

\[ A \Rightarrow B \]

is an algorithm

\[ \psi \]

which transforms every proof

\[ \varphi \]

of the formula \( A \) into a proof

\[ \psi(\varphi) \]

of the formula \( B \).
Cartesian closed categories

A cartesian category $\mathcal{C}$ is closed when there exists a functor

$$\Rightarrow : \mathcal{C}^{\text{op}} \times \mathcal{C} \to \mathcal{C}$$

and a natural bijection

$$\varphi_{A,B,C} : \mathcal{C}(A \times B, C) \cong \mathcal{C}(B, A \Rightarrow C)$$
The free cartesian closed category

The objects of the category \( \text{free-ccc}(\mathcal{C}) \) are the formulas

\[
A, B ::= X \mid A \times B \mid A \Rightarrow B \mid \mathbf{1}
\]

where \( X \) is an object of the category \( \mathcal{C} \).

The morphisms are the simply-typed \( \lambda \)-terms, modulo \( \beta\eta \)-conversion.

In particular, the \( \beta\eta \)-normal forms provide a “basis” of the free ccc.
### The simply-typed $\lambda$-calculus

<table>
<thead>
<tr>
<th>Rule</th>
<th>Judgments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variable</strong></td>
<td>$x : A \vdash x : A$</td>
</tr>
<tr>
<td><strong>Abstraction</strong></td>
<td>$\Gamma, x : A \vdash P : B$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma \vdash \lambda x . P : A \Rightarrow B$</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>$\Gamma \vdash P : A \Rightarrow B$</td>
</tr>
<tr>
<td></td>
<td>$\Delta \vdash Q : A$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, \Delta \vdash PQ : B$</td>
</tr>
<tr>
<td><strong>Weakening</strong></td>
<td>$\Gamma \vdash P : B$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, x : A \vdash P : B$</td>
</tr>
<tr>
<td><strong>Contraction</strong></td>
<td>$\Gamma, x : A, y : A \vdash P : B$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, z : A \vdash P[x, y \leftarrow z] : B$</td>
</tr>
<tr>
<td><strong>Exchange</strong></td>
<td>$\Gamma, x : A, y : B, \Delta \vdash P : C$</td>
</tr>
<tr>
<td></td>
<td>$\Gamma, y : B, x : A, \Delta \vdash P : C$</td>
</tr>
</tbody>
</table>
The simply-typed $\lambda$-calculus [with products]

Pairing
\[
\Gamma \vdash P : A \quad \Gamma \vdash Q : B \\
\Gamma \vdash \langle P, Q \rangle : A \times B
\]

Left projection
\[
\Gamma \vdash P : A \times B \\
\Gamma \vdash \pi_1 P : A
\]

Right projection
\[
\Gamma \vdash P : A \times B \\
\Gamma \vdash \pi_2 P : B
\]

Unit
\[
\Gamma \vdash \ast : 1
\]
Execution of $\lambda$-terms

In order to compute a $\lambda$-term, one applies the $\beta$-rule

\[
(\lambda x. P) Q \rightarrow_\beta P [x := Q]
\]

which substitutes the argument $Q$ for every instance of the variable $x$ in the body $P$ of the function. One may also apply the $\eta$-rule:

\[
P \rightarrow_\eta \lambda x. (Px)
\]
Proof invariants

Every ccc $\mathcal{D}$ induces a proof invariant $[-]$ modulo execution

A purely syntactic and type-theoretic construction
Knot invariants

Every ribbon category $\mathcal{D}$ induces a knot invariant $[-]$ modulo execution

A topological and algebraic construction
An analogy with knot invariants

Every ribbon category $\mathcal{D}$ induces a knot invariant $[\cdot]$ modulo deformation

The free ribbon category is the category of framed tangles
The free ribbon category

A typical morphism in the category \textbf{free-ribbon}(\mathcal{C})

\[(A^+) \longrightarrow (B^+, C^-, D^+)\]

looks like this:

where \( f : A \rightarrow B \) and \( g : C \rightarrow D \) are morphisms in the category \( \mathcal{C} \).
The Jones polynomial invariant

\[ \frac{2}{x^2} + \frac{1}{x^4} + \frac{y^2}{x^2} \]

\[ 2x^2 - x^4 + x^2y^2 \]
Proofs as 3-dimensional string diagrams

The left-to-right proof of the sequent

$$\neg\neg A \otimes \neg\neg B \vdash \neg\neg (A \otimes B)$$

is depicted as the flow of negation below
Higher-order recursion schemes

Seen through the lens of linear logic
Higher-order recursion schemes

The infinite tree

is generated by the higher-order recursion scheme

\[
\begin{align*}
S & \mapsto \rightarrow F \ a \ b \ c \\
F \ x \ y \ z & \mapsto \rightarrow x \ (y \ z) \ (F \ x \ y \ (y \ z))
\end{align*}
\]
Higher-order recursion schemes

Signature

\[ a : \circ \Rightarrow \circ \Rightarrow \circ \]
\[ b : \circ \Rightarrow \circ \]
\[ c : \circ \]

Non terminals

\[ S : \circ \]
\[ F : \circ \Rightarrow \circ \]

Rewrite rules

\[ S \rightarrow Fc \]
\[ F \rightarrow \lambda x. ax(F(bx)) \]

\[
S \rightarrow Fc \rightarrow ac(F(bc)) \rightarrow ac(a(bc)F(b(bc)))
\]
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme

\[
\begin{align*}
S & \mapsto F \ a \ b \ c \\
F \ x \ y \ z & \mapsto x \ (y \ z) \ (F \ x \ y \ (y \ z))
\end{align*}
\]

may be seen as a $\lambda$-term of type

\[(\circ \Rightarrow \circ \Rightarrow \circ) \Rightarrow (\circ \Rightarrow \circ) \Rightarrow \circ \Rightarrow \circ\]

in the simply-typed $\lambda$-calculus extended with a recursion operator $Y$.

Here, each tree-constructor $a$, $b$ and $c$ is of type:

\[
\begin{align*}
a & : \circ \Rightarrow \circ \Rightarrow \circ \\
b & : \circ \Rightarrow \circ \\
c & : \circ
\end{align*}
\]
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme

\[
\begin{align*}
S & \mapsto \rightarrow F\ a\ b\ c \\
F\ x\ y\ z & \mapsto \rightarrow x\ (y\ z)\ (F\ x\ y\ (y\ z))
\end{align*}
\]

may be seen as a $\lambda$-term of type

\[( ((\mathbb{O} \times \mathbb{O}) \Rightarrow \mathbb{O}) \times (\mathbb{O} \Rightarrow \mathbb{O}) \times \mathbb{O} ) \Rightarrow \mathbb{O} \]

in the simply-typed $\lambda$-calculus extended with a recursion operator $Y$.

Here, each tree-constructor $a$, $b$ and $c$ is of type:

\[
a : (\mathbb{O} \times \mathbb{O}) \Rightarrow \mathbb{O} \quad b : \mathbb{O} \Rightarrow \mathbb{O} \quad c : \mathbb{O}
\]
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme is translated as

$$M = (Y [\lambda F.\lambda x.\lambda y.\lambda z. x z (F x y (y z))] ) a b c$$

where the functional $F$ has type

$$((\circ \times \circ) \Rightarrow \circ) \times (\circ \Rightarrow \circ) \times \circ \Rightarrow \circ$$

Recall that the fixpoint operator $Y$ behaves in the following way:

$$Y M \mapsto M (YM).$$
Church encoding in linear logic

The formula

\[(\text{o} \Rightarrow \text{o} \Rightarrow \text{o}) \Rightarrow (\text{o} \Rightarrow \text{o}) \Rightarrow \text{t} \Rightarrow \text{t}\]

traditionally translated in linear logic as

\[A = !((!\text{o} \Rightarrow !\text{o} \Rightarrow \text{o}) \Rightarrow (!\text{o} \Rightarrow \text{o}) \Rightarrow !\text{t} \Rightarrow \text{t}).\]

may be also translated as

\[B = !((\text{o} \Rightarrow \text{o} \Rightarrow \text{t}) \Rightarrow (!\text{t} \Rightarrow \text{t}) \Rightarrow !\text{t} \Rightarrow \text{t}).\]
Church encoding in linear logic

So, the same tree may be seen as a term of type

$$A = !(!\odot \rightarrow !\odot \rightarrow \odot) \rightarrow !(!\odot \rightarrow \odot) \rightarrow !\odot \rightarrow \odot$$

with tree-constructors $a$, $b$ and $c$ of type

$$a : !\odot \rightarrow !\odot \rightarrow \odot \quad b : !\odot \rightarrow \odot \quad c : \odot$$

or as a term of type

$$B = !(\odot \rightarrow \odot \rightarrow \odot) \rightarrow !(!\odot \rightarrow \odot) \rightarrow !\odot \rightarrow \odot$$

with tree-constructors $a$, $b$ and $c$ of type

$$a : \odot \rightarrow \odot \rightarrow \odot \quad b : \odot \rightarrow \odot \quad c : \odot$$
Principle of duality

Proponent Program
plays the formula

$A$

Opponent Environment
plays the formula

$A^\perp$

Negation permutes the rôles of Proponent and Opponent
Principle of duality

Opponent
Environment
plays the formula
$A^\perp$

Proponent
Program
plays the formula
$A$

Negation permutes the rôles of Opponent and Proponent
Duality applied to the Church encoding

Question: So, what is the dual of a tree?

Answer: Well, it should be a tree automaton!
Duality applied to the Church encoding

The formulas $A$ and $B$ have counter-formulas:

$$A^\perp = ! ( ! \circ \rightarrow ! \circ \rightarrow \circ ) \otimes ! ( ! \circ \rightarrow \circ ) \otimes ! \circ \otimes \circ \perp$$

$$B^\perp = ! ( \circ \rightarrow \circ \rightarrow \circ ) \otimes ! ( \circ \rightarrow \circ ) \otimes \circ \otimes \circ \perp$$

Claim:

- the counter-formula $B^\perp$ is the type of **tree automata**
- the counter-formula $A^\perp$ is the type of **alternating tree automata**
Finite higher-order automata

Seen though the lens of linear logic
Higher-order recognizability

Suppose given a set $L$ of simply-typed $\lambda$-terms of same type $A$.

Question:
When should one consider the set $L$ as a recognizable language?
Higher-order recognizability

Suppose given a set $\mathcal{L}$ of simply-typed $\lambda$-terms of same type $A$.

**Question:**

When should one consider the set $\mathcal{L}$ as a recognizable language?

**Tentative answer:**

Interpret the simple type $A$ as a **finite** Scott domain.
Higher-order recognizability

Every finite Scott domain (= ordered set with a least element \( \perp \))

\[ D = (D, \leq) \]

induces an interpretation of \( A \) as a finite Scott domain:

\[
\begin{align*}
[\varnothing] & := D \\
[A \times B] & := [A] \times [B] \\
[A \Rightarrow B] & := [A] \Rightarrow [B]
\end{align*}
\]

Every \( \lambda \)-term \( M \) of type \( A \) is interpreted as an element

\[ [M] \in [A] \]

of the Scott domain \([A]\).
Higher-order recognizability

Now, every finite subset $\varphi \subseteq [A]$ induces a set

$$L_\varphi = \{ M \mid [M] \in \varphi \}$$

of $\lambda$-terms of type $A$.

**Notation:** We write $\models M : \varphi$ to mean that $[M] \in \varphi$.

**Definition.** [adapted from Salvati 2009]

A set of $\lambda$-terms $L$ is **recognizable** when it is of the form $L_\varphi$. 
The Scott semantics of linear logic

Well-known principle.

Every preorder \((A, \leq)\) induces a Scott domain

\[(\text{Dom}(A), \subseteq)\]

defined as follows:

\(\blacktriangleright\) its elements are the lower sets of the preorder,

\(\blacktriangleright\) the lower sets are ordered by inclusion.

Recall that a subset \(X \subseteq A\) is a **lower set** of the preorder \((A, \leq)\) when

\[\forall a \in A, \forall x \in X, \quad a \leq x \Rightarrow a \in X.\]
The Scott semantics of linear logic

Key observation.

Suppose that the base type $\emptyset$ is interpreted as the domain of lower sets

$$[\emptyset] = Dom(Q, \leq)$$

generated by a preorder $Q$ of atomic states.

In that case, the interpretation of every type $A$ is a domain of lower sets

$$[A] := Dom(Q_A, \leq_A)$$

generated by a specific preorder $Q_A$ of higher-order states.
The Scott semantics of linear logic

This induces a family of logical connectives on preorders:

\[ A^\perp := A^{op} \]
\[ A \& B := (A + B, \leq_A + \leq_B) \]
\[ A \otimes B := (A \times B, \leq_A \times \leq_B) \]
\[ !A := \wp_{fin}(A) \]

where the finite sets of elements of \( A \) are ordered as:

\[ \{ a_1, \ldots, a_p \} \leq_A \{ b_1, \ldots, b_q \} \iff \forall i \in \{1,\ldots,p\} \exists j \in \{1,\ldots,q\} \ a_i \leq_A b_j \]
The Scott semantics of linear logic

Given a preorder of atomic states for the base type $\circ$

$$Q_\circ = (Q, \leq)$$

the preorder $Q_A$ of higher-order states is defined by induction:

$$Q_A \times B = Q_A \& Q_B$$

$$Q_A \Rightarrow B = !Q_A \rightarrow Q_B$$

In particular, a state of the simple type $A \Rightarrow B$ is of the form

$$\{q_1, \ldots, q_n\} \rightarrow q$$

where $q_1, \ldots, q_n$ are states of $A$ and $q$ is a state of $B$. 
What is a higher-order automaton?

Methodological question.

Given a simple type $A$, a finite preorder $(Q, \leq)$ and a subset

$$\varphi \subseteq [A]$$

can we describe the $\lambda$-terms of the associated language

$$\mathcal{L}_\varphi = \{ M \mid [M] \in \varphi \} = \{ M \mid \models M : \varphi \}$$

in a more direct and automata-theoretic fashion?
What is a higher-order automaton?

Methodological question.

Given a simple type \( A \), a finite preorder \((Q, \leq)\) and an element

\[
q \in Q_A
\]

can we describe the \( \lambda \)-terms of the associated language

\[
\mathcal{L}_q = \{ M \mid q \in \llbracket M \rrbracket \}
\]
in a more direct and automata-theoretic fashion?
Higher-order alphabet

**Definition.** A higher-order alphabet

\[ \Sigma = a_1 : A_1, \ldots, a_n : A_n \]

is a finite set \( \Sigma \) of letters equipped with a function

\[ \Sigma : \Sigma \rightarrow Type \]

which maps every letter

\[ a \in \Sigma \]

to its higher-order arity

\[ \Sigma(a) \in Type \]

defined as a simple type of the \( \lambda \)-calculus.
What is a higher-order automaton?

**Definition.** A higher-order automaton

$$\mathcal{A} = \langle \Sigma, A, Q, \delta, q_0 \rangle$$

consists of:

- a higher-order alphabet
  $$\Sigma = a_1 : A_1, \ldots, a_n : A_n$$
- a simple type
  $$A$$
- a finite preordered set of states
  $$Q$$
- a family of transition functions
  $$\delta(a_i) \subseteq Q_{A_i}$$
- a higher-order initial state
  $$q_0 \in Q_A$$

where the interpretation of types is induced by the preorder $$Q_o = Q$$. 
Run-trees

**Definition** A run-tree $\mathcal{R}$ is a derivation tree of the judgement

$$\langle \Sigma \vdash M : A \mid \delta, q \rangle$$

in the deduction system defined by the rules

**Variable**

$$\frac{q \leq_A q' \quad q' \in \delta(a)}{\langle \Sigma, a : A \vdash a : A \mid \delta, q \rangle}$$

**Abstraction**

$$\frac{\langle \Sigma, a : A \vdash M : B \mid \delta + a \mapsto \{q_1, \ldots, q_n\}, q \rangle}{\langle \Sigma \vdash \lambda a. M : A \Rightarrow B \mid \delta, \{q_1, \ldots, q_n\} \leadsto q \rangle}$$

**Application**

$$\frac{\langle \Sigma \vdash M : A \Rightarrow B \mid \delta, u \leadsto q \rangle \quad \langle \Sigma \vdash N : A \mid \delta, u \rangle}{\langle \Sigma \vdash \text{App}(M, N) : B \mid \delta, q \rangle}$$

**Bag**

$$\frac{\langle \Sigma \vdash M : A \mid \delta, q_1 \rangle \quad \ldots \quad \langle \Sigma \vdash M : A \mid \delta, q_n \rangle}{\langle \Sigma \vdash M : A \mid \delta, \{q_1, \ldots, q_n\} \rangle}$$
Illustration

The higher-order automaton

\[ \mathcal{A} = \langle \Sigma, A \Rightarrow B, Q, \delta, q \rangle \]

with higher-order state

\[ q = \{q_1, \ldots, q_n\} \leadsto q_0 \in Q_{A \Rightarrow B} \]

confronted to the simply-typed \( \lambda \)-term

\[ \Sigma \vdash \lambda a. M : A \Rightarrow B \]

becomes the higher-order automaton

\[ \mathcal{A}' = \langle \Sigma \cup \{a : A\}, Q, \delta, a \mapsto \{q_1, \ldots, q_n\}, q_0 \rangle \]

confronted to the simply-typed \( \lambda \)-term

\[ \Sigma, a : A \vdash M : B. \]
Illustration of a run-tree

\[ \delta(a) = \{q_1 \rightarrow q_0 \rightarrow q_0\} \]

\[ \delta(b) = \{q_1 \rightarrow q_1\} \]

\[ \delta(c) = \{q_1\} \]
An adequacy theorem for the $\lambda$-calculus

Suppose given a finite preorder $(Q, \leq)$.

Adequacy Theorem  [Salvati 2009]

The interpretation $[M]$ of a simply-typed $\lambda$-term $M$ of type $A$ is the set of its accepting states.

In other words, for every higher-order state $q \in Q_A$,

$$ q \in [M] \iff M \text{ is accepted by the automaton } \langle \emptyset, Q, A, \emptyset, q \rangle $$
Higher-order recursion schemes

Moving to an infinitary situation
Higher-order recursion schemes

The infinite tree

\[ S \mapsto F a b c \]
\[ F x y z \mapsto x (y z) (F x y (y z)) \]

is generated by the higher-order recursion scheme
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme

\[
\begin{align*}
S & \mapsto F a b c \\
F x y z & \mapsto x (y z) (F x y (y z))
\end{align*}
\]

may be seen as a $\lambda$-term of type

\[(\circ \Rightarrow \circ \Rightarrow \circ) \Rightarrow (\circ \Rightarrow \circ) \Rightarrow \circ \Rightarrow \circ\]

in the simply-typed $\lambda$-calculus extended with a recursion operator $Y$.

Here, each tree-constructor $a$, $b$ and $c$ is of type:

\[
a : \circ \Rightarrow \circ \Rightarrow \circ \quad b : \circ \Rightarrow \circ \quad c : \circ
\]
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme

\[
\begin{align*}
S &\mapsto \rightarrow F \ a \ b \ c \\
F \ x \ y \ z &\mapsto \rightarrow x \ (y \ z) \ (F \ x \ y \ (y \ z))
\end{align*}
\]

may be seen as a $\lambda$-term of type

\[
( (((\circ \times \circ) \Rightarrow \circ) \times (\circ \Rightarrow \circ) \times \circ) \Rightarrow \circ
\]

in the simply-typed $\lambda$-calculus extended with a recursion operator $Y$.

Here, each tree-constructor $a$, $b$ and $c$ is of type:

\[
a : (\circ \times \circ) \Rightarrow \circ \quad b : \circ \Rightarrow \circ \quad c : \circ
\]
Church encoding in the $\lambda Y$-calculus

The higher-order recursion scheme is translated as

$$M = (Y[\lambda F.\lambda x.\lambda y.\lambda z.x z(Fxy(yz))] )a b c$$

where the functional $F$ has type

$$((\Box \times \Box) \Rightarrow \Box) \times (\Box \Rightarrow \Box) \times \Box \Rightarrow \Box$$

Recall that the fixpoint operator $Y$ behaves in the following way:

$$YM \mapsto M(YM).$$
Church encoding in the $\lambda Y$-calculus

This alternative (and somewhat simpler)

\[ M = (Y [ \lambda F. \lambda z. a z (F(bz))] ) \ c \]

produces the infinitary $\lambda$-term $[M]_{\infty}$ obtained by plugging the context into itself, coinductively...
\[ [M]_{\infty} = \]
Generation by infinitary $\beta$-rewriting

The $\lambda$-term $[M]_\infty$ is then rewritten by an infinite sequence of $\beta$-redexes

$$[M]_\infty \longrightarrow M_1 \longrightarrow \cdots \longrightarrow M_p \longrightarrow \cdots$$

into the expected infinite tree

$$N =$$

\[ a \]
\[ c \]
\[ a \]
\[ b \]
\[ a \]
\[ c \]
\[ b \]
\[ b \]
\[ b \]
\[ c \]
Generation by infinitary $\beta$-rewriting

The $\lambda$-term $[M]_\infty$ is then rewritten by an infinite sequence of $\beta$-redexes

$$[M]_\infty \rightarrow M_1 \rightarrow \cdots \rightarrow M_p \rightarrow \cdots$$

into the expected infinite tree (along the Church encoding)

\[ N = \text{tree diagram} \]
\[ M_1 = \]
\[ M_2 = \]
\[ N = \]
Generation by infinitary $\beta$-rewriting

The infinitary sequence of $\beta$-redexes

$$[M]_\infty \quad \xrightarrow{} \quad N$$

which turns $[M]_\infty$ into the infinite tree $N$ plays a central role...

**Key observation:**

The sequence may be chosen «**strongly Cauchy convergent**» in the sense of the Dutch school in infinitary rewriting.
Invariance theorem

More generally, consider an infinite sequence of $\beta$-redexes

$$M \longrightarrow N$$

which is strongly Cauchy convergent.

We establish that for every higher-order automaton $\mathcal{A}$, the following invariance property is satisfied by the rewriting path:

**Invariance theorem.**

The ho-automaton $\mathcal{A}$ recognizes the infinitary $\lambda$-term $M$ $\iff$ the ho-automaton $\mathcal{A}$ recognizes the infinitary $\lambda$-term $N$. 
An important message here...

This invariance property is apparently easy to establish using the traditional tools of denotational semantics:

- Scott semantics
- continuity
- Böhm trees

However, this semantic approach only works for automata with purely inductive acceptance conditions.

One thus needs to revisit the foundations entirely for more sophisticated notions of higher-order automata mixing inductive and coinductive acceptance conditions.
Higher-order automata

Shifting to the infinitary $\lambda$-calculus
The $\lambda Y_{\mu\nu}$-calculus

The $\lambda Y_{\mu\nu}$-calculus is defined as the simply-typed $\lambda$-calculus equipped with a least and greatest fixpoint operators:

$$Y_{\mu} : (A \Rightarrow A) \Rightarrow A$$
$$Y_{\nu} : (A \Rightarrow A) \Rightarrow A.$$

The two operators behave in the same way syntactically:

$$Y_{\mu} M \rightarrow M (Y_{\mu} M)$$
$$Y_{\nu} M \rightarrow M (Y_{\nu} M)$$

but they are interpreted differently in the Scott semantics $[\cdot]_{\mu\nu}$. 
**Infinite $\lambda$-terms with boundary**

**Definition**  A boundary $\mathcal{P}$ of a simply-typed infinitary $\lambda$-term $M$ is a set

\[ \mathcal{P} \subseteq \infty\text{-path}(M) \]

of infinite paths of $M$.

A simply-typed infinitary $\lambda$-term with **boundary** is a pair

\[ (M, \mathcal{P}_M) \]

consisting of a simply-typed infinitary $\lambda$-term $M$ together with a boundary $\mathcal{P}_M$.

Inspired by the definition of Borelian games in descriptive set theory
The adequacy theorem with boundary

Suppose given a finite preorder \((Q, \leq)\).

**Adequacy Theorem**

The interpretation \([M]_{\mu\nu}\) of a simply-typed \(\lambda Y_{\mu\nu}\)-term \(M\) of type \(A\) coincides with the set of its accepting states.

In other words, for every higher-order state \(q \in Q_A\),

\[
q \in [M] \iff M \text{ is accepted by the automaton } \mathcal{A} = \langle \emptyset, Q, A, \emptyset, q \rangle
\]

where the acceptance condition on the run-trees of the automaton \(\mathcal{A}\) reflects the **inductive** and **coinductive** status of the fixpoints.
Back to our illustration

The translation

\[ M = (Y [ \lambda F. \lambda z. a z (F (b z))] ) c \]

produces the infinitary \( \lambda \)-term \([M]_\infty\) obtained by plugging the context into itself

\[ \text{inductively or coinductively} \] depending on the definition of the boundary...
Traditional definition of the fixpoint operator $Y$

$[Y P]_\infty = \ldots$
Inductive definition of the fixpoint operator $Y_\mu$

\[ [Y_\mu P]_\infty = \text{infinite path not in the boundary} \]
Coinductive definition of the fixpoint operator $Y_\nu$

$$[Y_\nu P]_\infty = \text{infinite path in the boundary}$$
Generation by infinitary $\beta$-rewriting

The $\lambda$-term $[M]_\infty$ is then **rewritten** by an infinite sequence of $\beta$-redexes

$$[M]_\infty \longrightarrow M_1 \longrightarrow \cdots \longrightarrow M_p \longrightarrow \cdots$$

into the expected infinite tree

$$N = \quad \cdots$$
Generation by infinitary $\beta$-rewriting

The $\lambda$-term $[M]_\infty$ is then rewritten by an infinite sequence of $\beta$-redexes

$$[M]_\infty \rightarrow M_1 \rightarrow \cdots \rightarrow M_p \rightarrow \cdots$$

into the expected infinite tree (along the Church encoding)

$$N =$$

![Diagram of the infinite tree](image-url)
The need for an invariance theorem

Consider an infinite sequence of $\beta$-redexes

\[ M \rightarrow N \]

which is strongly Cauchy convergent.

We establish that for every higher-order automaton $A$, the following invariance property is satisfied by the rewriting path:

Invariance theorem.

\[
\begin{array}{c}
\text{the ho-automaton } A \\
\text{recognizes} \\
\text{the infinitary } \lambda\text{-term } M \\
\text{with boundary}
\end{array} \iff \begin{array}{c}
\text{the ho-automaton } A \\
\text{recognizes} \\
\text{the infinitary } \lambda\text{-term } N \\
\text{with boundary.}
\end{array}
\]
A key tool: diffraction patterns

Key idea:
The occurrence $D$ of a $\beta$-redex $R$ is turned into a diffraction pattern

$$E = \{ D_{A,i} \mid i \in I \} \rightarrow D_B$$

by the reduction of the $\beta$-redex:

\[
\begin{align*}
R &\rightarrow_{\beta} \\
\end{align*}
\]
\[ [M]_\infty = \]
$M_1 = \ldots$
$M_2 = \ldots$
$N = $
A modal translation of higher-order parity games

The S4 construction at work
A colour modality for Scott domains

Suppose given a specific number \( n \) of colours.

**Definition.** The colour modality on preorders is defined as

\[
\square A := \underbrace{A \& \cdots \& A}_n
\]

As a consequence, note that

\[
Dom(\square A) := Dom(A) \times \cdots \times Dom(A)
\]
The colour modality

Two observations

▷ The modality $\Box$ defines a comonad.

\[
\begin{align*}
\varepsilon_A &: \quad \Box A & \rightarrow & & A \\
(1, q) & \mapsto & & q \\
\delta_A &: \quad \Box A & \rightarrow & & \Box \Box A \\
(\max (m_1, m_2), q) & \mapsto & & (m_1, (m_2, q))
\end{align*}
\]

▷ The comonad $\Box$ commutes with finite products:

\[
\begin{align*}
\Box (A & \& B) & \cong & \Box A & \& \Box B \\
\Box \top & \cong & \top
\end{align*}
\]
A colour modality

**An important consequence:** The composite modality

\[ ! \Box : \text{Scott} \to \text{Scott} \]

defines an exponential modality of linear logic.

From this follows that the Kleisli category

\[ \mathcal{D} := \text{Kleisli}(\text{Scott}, ! \Box) \]

is a cartesian closed category.
An inductive-coinductive fixpoint

For simplicity, let us assume that the number $n$ of colours is even.

Given an infinitary $\lambda$-term

$$M : A^n \Rightarrow A$$

one defines the fixpoint as

$$Y(M) = \nu x_n \mu x_{n-1} \nu x_{n-2} \cdots \nu x_2 \mu x_1 M(x_1, \cdots, x_n)$$

**Theorem.** This defines an interpretation in the $\lambda Y_{\mu \nu}$-calculus.
Conclusion and future works

Higher-order automata generalising and explaining higher-order model checking

A modal $\lambda Y_{\mu \nu}$-calculus with boundaries refining the usual $\lambda Y$-calculus

A neat proof of decidability based on:

- Scott semantics of linear logic in the French style
- Infinitary rewriting theory in the Dutch style

New automata-theoretic foundations to the lambda-calculus

New features: higher-order, compositionality, which need to be explored
Thank you !
Modal reformulation

Collecting colours works in the same way as collecting levels of copies
A colour modality for intersection types

Definition. A parametric modality is a family of functors

\[ \square_m : \mathcal{C} \to \mathcal{C} \quad m \in \mathbb{N} \]

each of them lax monoidal:

\[ \square_m A \otimes \square_m B \to \square_m (A \otimes B) \]
\[ 1 \to \square_m 1 \]

and defining together a parametric comonad

\[ \square_{max(m,m')} A \to \square_m \square_{m'} A \]
\[ \square_0 A \to A \]

The structure of copy management in linear logic
The exponential modality

\[ !A \otimes !B \quad \rightarrow \quad !(A \otimes B) \]

\[ !A \quad \rightarrow \quad ! !A \]

\[ !A \quad \rightarrow \quad A \]

The structure of \textit{copy management} in linear logic
Translation

\[
\Delta \vdash t : (\theta_1, m_1) \land \ldots \land (\theta_k, m_k) \Rightarrow \theta \\
\Delta, \Delta_1 \uparrow m_1, \ldots, \Delta_k \uparrow m_k \vdash t u : \theta
\]

where

\[
\Delta \uparrow m = \{ F : (\theta, \max(m, m')) \mid F : (\theta, m) \in \Delta \}
\]

is translated as

\[
\Delta \vdash t : \Box_{m_1} \theta_1 \land \ldots \land \Box_{m_k} \theta_k \Rightarrow \theta \\
\Delta, \Box_{m_1} \Delta_1, \ldots, \Box_{m_k} \Delta_k \vdash t u : \theta
\]

\[
\Delta_i \vdash u : \theta_i \\
\Box_{m_i} \Delta_i \vdash u : \Box_{m_i} \theta_i
\]
A domain-theoretic formulation

The category $\mathcal{D}$ has

- finite prime algebraic domains as objects
- continuous functions $f : D^n \to E$ as morphisms.

Two morphisms of the category $\mathcal{D}$

$$f : D^n \to E \quad g : E^n \to F$$

are composed as follows:

$$D^n \xrightarrow{D^{\text{max}}} D^{n \times n} \xrightarrow{f^n} E^n \xrightarrow{g} E$$
A domain-theoretic formulation

In the case $n = 2$

$$g \circ f : (x_1, x_2) \mapsto g(f(x_1, x_2), (x_2, x_2))$$

In the case $n = 3$

$$g \circ f : (x_1, x_2, x_3) \mapsto g(f(x_1, x_2, x_3), f(x_2, x_2, x_3), f(x_3, x_3, x_3))$$

More generally:

$$\begin{pmatrix}
1 & 2 \\
2 & 2 \\
3 & 3 & 3
\end{pmatrix} \quad \begin{pmatrix}
1 & 2 & 3 \\
2 & 2 & 3 \\
3 & 3 & 3
\end{pmatrix} \quad \begin{pmatrix}
1 & 2 & 3 & 4 \\
2 & 2 & 3 & 4 \\
3 & 3 & 3 & 4
\end{pmatrix} \quad \begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
2 & 2 & 3 & 4 & 5 \\
3 & 3 & 3 & 4 & 5 \\
4 & 4 & 4 & 4 & 5 \\
5 & 5 & 5 & 5 & 5
\end{pmatrix}$$