Verifying properties of functional programs using modal extensions of linear logic

Charles Grellois
(partly joint with Melliès)

FOCUS Team – INRIA & University of Bologna

Présentation à l'équipe LIRICA
June 12, 2017

Charles Grellois — Parcours

Emplois

Jan 2016 –	Postdoc INRIA Bologne
2e sem 2015	Research Assistant Dundee
2012-2015	Doctorant Moniteur PPS & LIAFA, Paris 7
2008-2012	Fonctionnaire Stagiaire ENS Cachan

Formation

M2 Informatique Théorique MPRI

M2 Mathématiques Fondamentales, Paris 6

Mobilité

Bologne (20 mois)

Dundee (5 mois)

Oxford (3 + 1 mois)

Turku (3 mois)

Thèmes étudiés en lien avec l'équipe

- Logique
  - connaissances de base en théorie de la preuve (calcul des séquents, élimination des coupures, un peu de méthode des tableaux. . .)
  - en particulier logique linéaire et ses sémantiques (dénotationelles, modèles de jeux. . .)
  - aussi, preuves circulaires
- Logique et automates (MSO, μ-calcul modal, automates à parité. . .)
- Théorie des catégories, notamment en lien avec la sémantique
- Actuellement : programmation probabiliste et liens avec l'IA (machine learning)

Aujourd'hui : on va parler de logique (linéaire, modale) et automates.

Functional programs, Higher-order models

Imperative vs. functional programs

- Imperative programs: built on finite state machines (like Turing machines).

  Notion of state, global memory.

- Functional programs: built on functions that are composed together (like in Lambda-calculus).

  No state (except in impure languages), higher-order: functions can manipulate functions.

(recall that Turing machines and λ-terms are equivalent in expressive power)

Imperative vs. functional programs

- Imperative programs: built on finite state machines (like Turing machines).

  Notion of state, global memory.

- Functional programs: built on functions that are composed together (like in Lambda-calculus).

  No state (except in impure languages), higher-order: functions can manipulate functions.

(recall that Turing machines and λ-terms are equivalent in expressive power)

Example: imperative factorial

```
int fact(int n) {
  int res = 1;
  for i from 1 to n do {
    res = res * i;
    }
  }
  return res;
}
```

Typical way of doing: using a variable (change the state).

Example: functional factorial

In OCaml:

```
let rec factorial n =
    if n <= 1 then
      1
    else
      factorial (n-1) * n;;
```

Typical way of doing: using a recursive function (don't change the state).

In practice, forbidding global variables reduces considerably the number of bugs, especially in a parallel setting (cf. Erlang).

Advantages of functional programs

- Very mathematical: calculus of functions.
- ...and thus very much studied from a mathematical point of view. This notably leads to strong typing, a marvellous feature.
- Much less error-prone: no manipulation of global state.

More and more used, from Haskell and Caml to Scala, Javascript and even Java 8 nowadays.

Also emerging for probabilistic programming.

Price to pay: analysis of higher-order constructs.

Advantages of functional programs

Price to pay: analysis of higher-order constructs.

Example of higher-order function: `map`.

`map` φ $[0,1,2]$ returns $[\varphi(0),\varphi(1),\varphi(2)]$.

Higher-order: `map` is a function taking a function φ as input.

Semantics of linear logic and higher-order model-checking

Linear logic: a logical system with an emphasis on the notion of *resource*.

Model-checking: a key technique in *verification* — where we want to determine *automatically* whether a program satisfies a specification.

My thesis: linear logic and its semantics can be enriched to obtain new and cleaner proofs of decidability in higher-order model-checking.

What is model-checking?

The halting problem

A natural question: does a program always terminate?

Undecidable problem (Turing 1936): a machine can not always determine the answer.

What if we use approximations?

Model-checking

Approximate the program $\longrightarrow$ build a model $\mathcal{M}$.

Then, formulate a logical specification φ over the model.

Aim: design a program which checks whether

$$\mathcal{M} \models \varphi.$$

That is, whether the model $\mathcal{M}$ meets the specification φ.

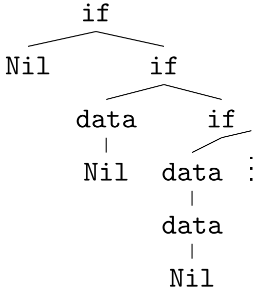
An example

$$\begin{array}{rcl} \texttt{Main} & = & \texttt{Listen Nil} \\ \texttt{Listen}\ x & = & \texttt{if end\_signal() then}\ x \\ & & \texttt{else Listen received\_data()} :: x \end{array}$$

An example

$$
\begin{array}{rcl}
\texttt{Main} & = & \texttt{Listen Nil} \\
\texttt{Listen } x & = & \texttt{if end\_signal() then } x \\
 & & \texttt{else Listen received\_data()} :: x
\end{array}
$$

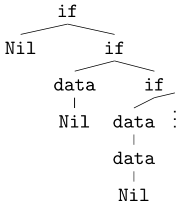
A tree model:



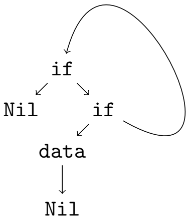
We abstracted conditionals and datatypes.

The approximation contains a non-terminating branch.

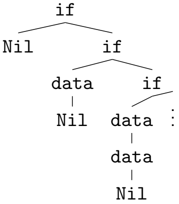
Finite representations of infinite trees



is not regular: it is not the unfolding of a finite graph as



Finite representations of infinite trees



but it is represented by a higher-order recursion scheme (HORS).

Modeling functional programs using higher-order recursion schemes

Higher-order recursion schemes

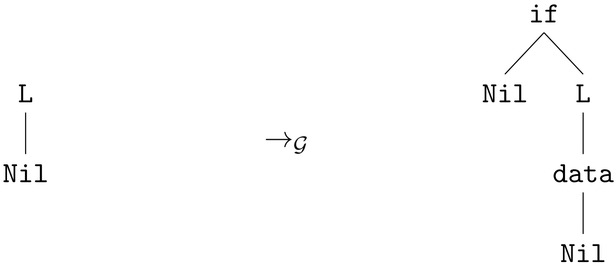
$$\mathcal{G} = \begin{cases} \texttt{S} & = \texttt{L Nil} \\ \texttt{L}\ x & = \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{cases}$$

Rewriting starts from the start symbol S:

$$\texttt{S} \quad \rightarrow_{\mathcal{G}} \quad \begin{array}{c} \texttt{L} \\ | \\ \texttt{Nil} \end{array}$$

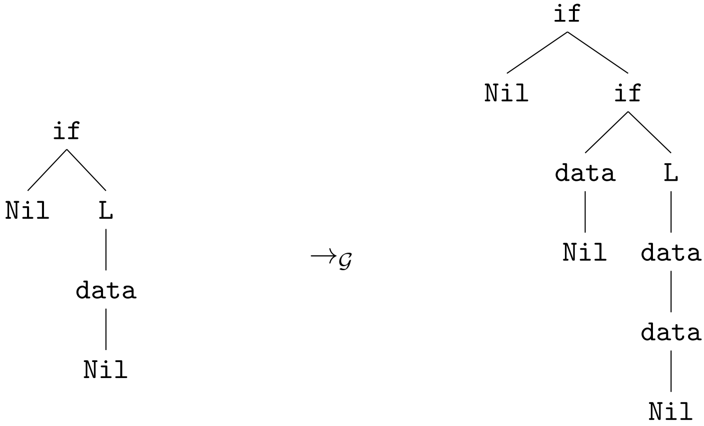
Higher-order recursion schemes

$$\mathcal{G} = \begin{cases} \mathtt{S} & = \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

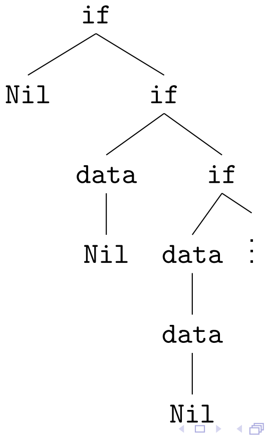
$$\mathcal{G} = \begin{cases} \mathtt{S} & = \mathtt{L}\ \mathtt{Nil} \\ \mathtt{L}\ x & = \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x\ )\ ) \end{cases}$$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \begin{cases} \texttt{S} & = \quad \texttt{L Nil} \\ \texttt{L}\ x & = \quad \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{cases}$$

$\langle \mathcal{G} \rangle \quad =$



Higher-order recursion schemes

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L}\ x & = & \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x\ )\ ) \end{array} \right.$$

HORS can alternatively be seen as simply-typed λ-terms with

simply-typed recursion operators $Y_\sigma \; : \; (\sigma \rightarrow \sigma) \rightarrow \sigma$.

Alternating parity tree automata

Checking specifications over trees

Monadic second order logic

MSO is a common logic in verification, allowing to express properties as:

" all executions halt "

" a given operation is executed infinitely often in some execution "

" every time data is added to a buffer, it is eventually processed "

It is also equivalent to modal μ-calculus over trees.

Alternating parity tree automata

Checking whether a formula holds can be performed using an automaton.

For an MSO formula φ, there exists an equivalent APT $\mathcal{A}_\varphi$ s.t.

$$\langle \mathcal{G} \rangle \models \varphi \quad \text{iff} \quad \mathcal{A}_\varphi \text{ has a run over } \langle \mathcal{G} \rangle.$$

APT $=$ alternating tree automata (ATA) $+$ parity condition.

Alternating tree automata

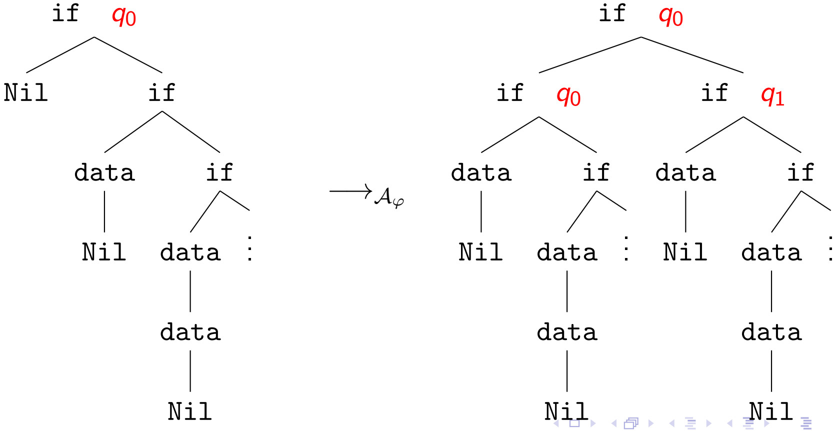
ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$.

Alternating tree automata

ATA: non-deterministic tree automata whose transitions may duplicate or drop a subtree.

Typically: $\delta(q_0, \texttt{if}) \; = \; (2, q_0) \wedge (2, q_1)$.

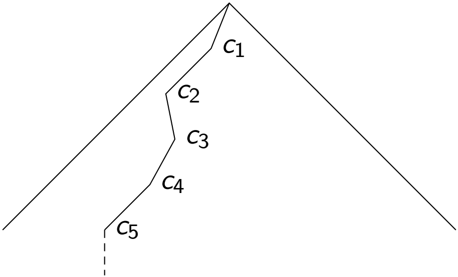


Alternating parity tree automata

Each state of an APT is attributed a color

$$\Omega(q) \in Col \subseteq \mathbb{N}$$

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.



Alternating parity tree automata

Each state of an APT is attributed a color

$$\Omega(q) \in Col \subseteq \mathbb{N}$$

An infinite branch of a run-tree is winning iff the maximal color among the ones occuring infinitely often along it is even.

A run-tree is winning iff all its infinite branches are.

For a MSO formula φ:

$\mathcal{A}_\varphi$ has a winning run-tree over $\langle \mathcal{G} \rangle$ iff $\langle \mathcal{G} \rangle \vDash \varphi$.

The higher-order model-checking problem ($\star$)

($\star$) : there are three but we present just one here

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

Output: `true` if and only if $\langle \mathcal{G} \rangle \vDash \varphi$.

Example: φ = " there is an infinite execution "

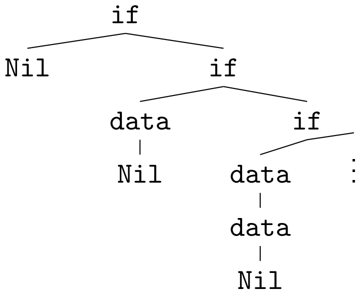
Output: `true`.

The (local) HOMC problem

Input: HORS $\mathcal{G}$, formula φ.

Output: `true` if and only if $\langle \mathcal{G} \rangle \vDash \varphi$.

Example: φ = " there is an infinite execution "



Output: `true`.

Recognition by homomorphism

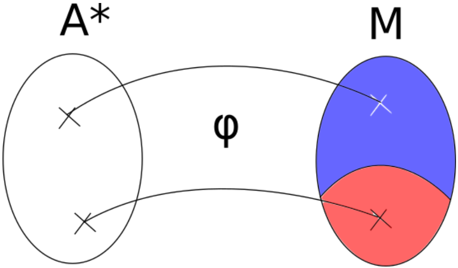
Where semantics comes into play

Automata and recognition

For the usual finite automata on words: given a regular language $L \subseteq A^*$,

there exists a finite automaton $\mathcal{A}$ recognizing L

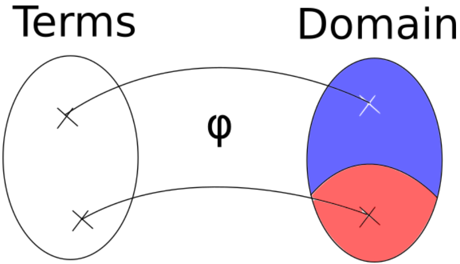
if and only if. . .



there exists a finite monoid M, a subset $K \subseteq M$
and a homomorphism $\varphi : A^* \to M$ such that $L = \varphi^{-1}(K)$.

Automata and recognition

The picture we want:



(after Aehlig 2006, Salvati 2009)

but with recursion and w.r.t. an APT.

Intersection types and alternation

A first connection with linear logic

Alternating tree automata and intersection types

A key remark (Kobayashi 2009):

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

can be seen as the intersection typing

$$\texttt{if} \;:\; \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0$$

refining the simple typing

$$\texttt{if} \;:\; o \rightarrow o \rightarrow o$$

Alternating tree automata and intersection types

In a derivation typing the tree $\texttt{if} \;\; T_1 \;\; T_2$:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{if} : \emptyset \rightarrow (q_0 \wedge q_1) \rightarrow q_0}\,\delta \qquad \emptyset}{\emptyset \vdash \texttt{if}\ T_1 : (q_0 \wedge q_1) \rightarrow q_0}\,\text{App} \qquad \dfrac{\vdots}{\emptyset \vdash T_2 : q_0} \qquad \dfrac{\vdots}{\emptyset \vdash T_2 : q_1}}{\emptyset \vdash \texttt{if}\ T_1\ T_2 : q_0}\,\text{App}$$

Intersection types naturally lift to higher-order – and thus to $\mathcal{G}$, which finitely represents $\langle \mathcal{G} \rangle$.

Theorem (Kobayashi 2009)

$\vdash \mathcal{G} : q_0$ *iff* *the ATA $\mathcal{A}_\varphi$ has a run-tree over $\langle \mathcal{G} \rangle$.*

A closer look at the Application rule

In the intersection type system:

$$\text{App} \quad \frac{\Delta \vdash t : (\ \theta_1 \ \wedge \cdots \wedge \ \theta_n) \rightarrow \theta \qquad \Delta_i \vdash u \ : \ \theta_i}{\Delta\ ,\ \Delta_1\ ,\ \ldots\ ,\ \Delta_n \ \vdash \ \ t\, u \ : \ \theta}$$

This rule could be decomposed as:

$$\frac{\Delta \ \vdash \ t \ : \ (\ \bigwedge_{i=1}^{n} \ \theta_i\ ) \rightarrow \theta' \qquad \dfrac{\Delta_i \ \vdash \ u \ : \ \theta_i \qquad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n \ \vdash \ u \ : \ \bigwedge_{i=1}^{n} \ \theta_i} \ \text{Right} \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n \ \vdash \ t \ u \ : \ \theta'}$$

A closer look at the Application rule

In the intersection type system:

$$\text{App} \qquad \frac{\Delta \vdash t : (\,\theta_1 \;\wedge \cdots \wedge\; \theta_n) \to \theta \qquad \Delta_i \vdash u \,:\, \theta_i}{\Delta\,,\, \Delta_1\,,\,\ldots\,,\,\Delta_n \;\vdash\; t\,u \,:\, \theta}$$

This rule could be decomposed as:

$$\frac{\Delta \;\vdash\; t \;:\; (\,\bigwedge_{i=1}^{n}\;\theta_i\,) \to \theta' \qquad \dfrac{\Delta_i \;\vdash\; u \;:\; \theta_i \qquad \forall i \in \{1,\ldots,n\}}{\Delta_1,\ldots,\Delta_n \;\vdash\; u \;:\; \bigwedge_{i=1}^{n}\;\theta_i}\;\text{Right}\;\bigwedge}{\Delta,\,\Delta_1,\ldots,\,\Delta_n \;\vdash\; t\;u \;:\; \theta'}$$

A closer look at the Application rule

$$\dfrac{\Delta \;\vdash\; t : (\bigwedge_{i=1}^{n}\ \theta_i) \to \theta' \qquad \dfrac{\Delta_i \;\vdash\; u : \theta_i \qquad \forall i \in \{1, \ldots, n\}}{\Delta_1, \ldots, \Delta_n \;\vdash\; u : \bigwedge_{i=1}^{n}\ \theta_i}\ \text{Right } \bigwedge}{\Delta, \Delta_1, \ldots, \Delta_n \;\vdash\; t\ u : \theta'}$$

Linear decomposition of the intuitionistic arrow:

$$A \Rightarrow B \;=\; !\,A \multimap B$$

Two steps: duplication / erasure, then linear use.

Right $\bigwedge$ corresponds to the Promotion rule of indexed linear logic.
(see G.-Melliès, ITRS 2014)

Intersection types and semantics of linear logic

$$A \Rightarrow B \;=\; !A \multimap B$$

Two interpretations of the exponential modality:

Qualitative models
(Scott semantics)

$$!A \;=\; \mathcal{P}_{fin}(A)$$

$$[\![o \Rightarrow o]\!] \;=\; \mathcal{P}_{fin}(Q) \times Q$$

$$\{q_0,\, q_0,\, q_1\} \;=\; \{q_0,\, q_1\}$$

Order closure

Quantitative models
(Relational semantics)

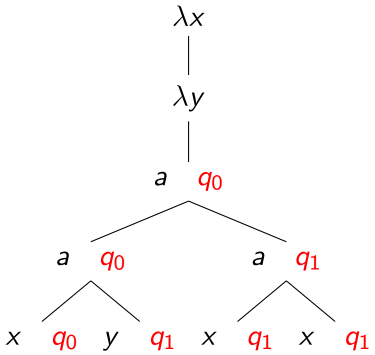
$$!A \;=\; \mathcal{M}_{fin}(A)$$

$$[\![o \Rightarrow o]\!] \;=\; \mathcal{M}_{fin}(Q) \times Q$$

$$[q_0,\, q_0,\, q_1] \;\neq\; [q_0,\, q_1]$$

Unbounded multiplicities

An example of interpretation



In *Rel*, one denotation:

$$([q_0, q_1, q_1], [q_1], q_0)$$

In *ScottL*, a set containing the principal type

$$(\{q_0, q_1\}, \{q_1\}, q_0)$$

but also

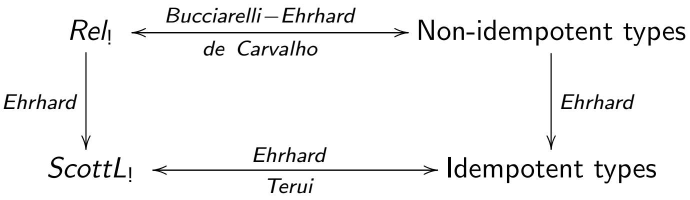
$$(\{q_0, q_1, q_2\}, \{q_1\}, q_0)$$

and

$$(\{q_0, q_1\}, \{q_0, q_1\}, q_0)$$

and ...

Intersection types and semantics of linear logic



Let t be a term normalizing to a tree $\langle t \rangle$ and $\mathcal{A}$ be an alternating automaton.

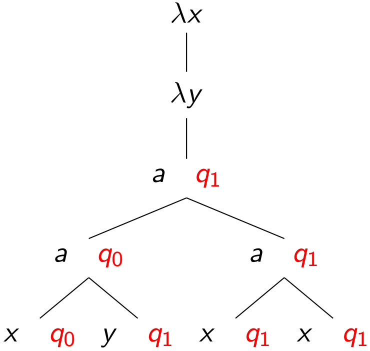
$$\mathcal{A} \text{ accepts } \langle t \rangle \text{ from } q \quad \Leftrightarrow \quad q \in [\![t]\!] \quad \Leftrightarrow \quad \emptyset \vdash t : q :: o$$

Extension with recursion and parity condition?

Adding parity conditions to the type system

An example of colored intersection type

Set $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$.



has now type

$$\square_0\, q_0 \wedge \square_1\, q_1 \rightarrow \square_1\, q_1 \rightarrow q_1$$

Note the color 0 on q_0...

A type-system for verification

We devise a type system capturing all MSO:

Theorem (G.-Melliès 2014, from Kobayashi-Ong 2009)

$S : q_0 \vdash S : q_0$ admits a winning typing derivation iff the alternating parity automaton $\mathcal{A}$ has a winning run-tree over $\langle \mathcal{G} \rangle$.

We obtain decidability by considering idempotent types.

Our reformulation

- shows the modal nature of $\Box$ (in the sense of S4),
- internalizes the parity condition,
- paves the way for semantic constructions.

Colored semantics

We extend:

- *Rel* with countable multiplicities, coloring and an inductive-coinductive fixpoint
- *ScottL* with coloring and an inductive-coinductive fixpoint.

Methodology: think in the relational semantics, and adapt to the Scott semantics using Ehrhard's 2012 result:

the finitary model *ScottL* is the extensional collapse of *Rel*.

Finitary semantics

In ScottL, we define $\Box$, λ and $\mathbf{Y}$ using downward-closures.
ScottL$_{\lightning}$ is a model of the λY-calculus.

Theorem

An APT $\mathcal{A}$ has a winning run from q_0 over $\langle \mathcal{G} \rangle$ if and only if

$$q_0 \in [\![\lambda(\mathcal{G})]\!].$$

Corollary

The local higher-order model-checking problem is decidable (and is n-EXPTIME complete).

Similar model-theoretic results were obtained by Salvati and Walukiewicz the same year.

Conclusion

J'ai également travaillé :

- en combinatoire des mots (stage de L3)
- en algèbre universelle (mémoire de M2 maths)
- sur la terminaison des programmes fonctionnels probabilistes (postdoc à Bologne)

Projet de recherche :

- model-checking d'ordre supérieur : aller vers une compréhension logique plus poussée (lien avec les preuves circulaires, les travaux de Luigi Santocanale, de Baelde-Doumane-Saurin. . .)
- terminaison probabiliste : finir mes travaux avec Ugo Dal Lago et essayer d'aller un peu plus loin
- logiques pour l'IA (logiques modales non-normales, modalités probabilistes, quantificateurs non-standards) avec Nicola Olivetti
- je suis ouvert à d'autres collaborations avec LIRICA !