

An Interactive Foundation for Computation as Proof-Search

Alexis Saurin

Lambda, Università degli Studi di Torino (currently)
Parsifal, INRIA Saclay & École Polytechnique (until recently)

30th january 2009

***Second Workshop on Formal and
Automated Theorem Proving and Applications,
University of Belgrade***

What this talk is about:

- Logical foundations of programming languages;
- Interactive approach to logic: proofs as dialogical argumentations, game-theoretic approach to proofs;
- The central role the *cut rule* plays in the dynamics of proofs;
- Relating the *functional programming* approach and the *logic programming* approach;
- A new setting to specify computation-as-proof-search.

Outline of the presentation

- *Sequent calculus, proof theory and computation;*
- *Background on linear logic;*
- *Interactive proof search in $MALL^\times$;*
- *Abstracting away from sequent proofs: from $MALL^\times$ to Ludics;*
- *A uniform framework for computation-as-proof-search.*

Proof Theory, Sequent Calculus and Computation

Sequent Calculus and the Cut Rule: Lemmas

Most often, to prove theorem \mathcal{T} :

Indirect arguments

- 1 First find an appropriate lemma \mathcal{L} ;
- 2 Establish that \mathcal{L} holds;
- 3 Prove that \mathcal{L} entails \mathcal{T} ;
- 4 Finally deduce that \mathcal{T} holds.

Here lies the **mathematical idea**: to find the appropriate lemma, the one that makes the proof simple...

This is reflected in sequent proofs thanks to the **cut rule**:

$$\frac{\frac{\Pi_1}{\vdash \mathcal{L}} \quad \frac{\Pi_2}{\mathcal{L} \vdash \mathcal{T}}}{\vdash \mathcal{T}} \text{ cut}$$

Natural questions:

Is it possible never to use lemmas?
To consider only direct proofs?

Is it possible never to use lemmas? To consider only direct proofs?
Yes [Gentzen]:

- Cut-admissibility: LK (resp. LJ) proves the same theorems with or without the cut rule;
- Cut-elimination: given an LK (resp. LJ) proof, there is a systematic and mechanical procedure to transform it into a proof which does not use the cut rule (cut-free proof).

\implies ***The key to the connection between proof theory and computation.***

Proof theory and computation are strongly related.

Mainly in two ways, two dynamical approaches to proofs:

- Cut-elimination: the dynamics lies in the process of transforming **a proof with cuts into a cut-free proof**;
- Proof-search: the dynamics lies in the search for a **cut-free proof**.

Both approaches rely on Gentzen's Hauptsatz.

Related with 2 styles of programming languages:

- Functional programming (*see Silvia's talk*);
- Logic programming (*next slide*)

Proof Theory and Logic Programming.

Examples of languages: Prolog, λ -Prolog, GNU-Prolog, Forum, ...

- The program is encoded as a sequent, typically $\mathcal{P} \vdash G$;
- **Dynamics of computation**: search for a cut-free proof;
- The **operational meaning** of this search lies in constraints that are imposed to the search strategy, for instance a **goal-directed search**. Example:

$$\frac{\mathcal{P}, A \vdash G}{\mathcal{P} \vdash A \Rightarrow G} \text{ load/} \Rightarrow$$

Logic Programming	\longleftrightarrow	Proof Search
Program	\longleftrightarrow	Sequent
Program Clause	\longleftrightarrow	Formula
Computation	\longleftrightarrow	Search for a Proof
Results	\longleftrightarrow	Cut-free Proofs

Cut Rule Plays a Crucial Role

- *Proof-Search*: the dynamics of the computation comes from the search for a cut-free proof;
- *Cut-Elimination*: the dynamics of the computation lies in the normalization of a proof into a cut-free proof;
- **Cut-admissibility** vs. **Cut-elimination** (*two aspects of the same result*);
- In both cases, results of computations are cut-free proofs;
- Complexity lies in the choice of the cut-formula;
- Though, it is difficult to relate functional programming and logic programming in a logically satisfying way.

Proof theory provides a satisfying foundation for functional and logic programming languages...

... which is much less satisfying as soon as control is concerned:

- **(FP)** Extending Curry-Howard from intuitionistic to classical logic took a long time (Howard 69 \rightarrow Griffin 90, Parigot 91)
- **(LP)** Backtracking and pruning operators are not satisfyingly treated from a proof-theoretical point of view in the computation-as-proof-search framework.

Algorithm = Logic + Control

Is it possible to capture *Control* with logical methods?

Understand the Logic of Control

Background on Linear Logic

Linear Logic [Girard, 1987]:

- *is the result of a careful analysis of **structural** rules in sequent calculus;*
- *has more connectives than LK (2 conjunctions, 2 disjunctions plus modalities), but the new inference rules are actually derived in a simple way from the usual rules for LK;*
- *is built on strong duality principles \Rightarrow one-sided sequents.*

LL formulas:

$$F ::= \begin{array}{l} a \mid F \otimes F \mid F \oplus F \mid \mathbf{1} \mid \mathbf{0} \mid \exists x.F \mid !F \\ a^\perp \mid F \wp F \mid F \& F \mid \perp \mid \top \mid \forall x.F \mid ?F \end{array} \begin{array}{l} \text{positive} \\ \text{negative} \end{array}$$

positive/negative duality

Two Conjunctions

In LK, the following inference rules are provably equivalent:

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \wedge^m \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B} \wedge^a$$

But not in LL because of the restriction on the structural rules.
This leads to:

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \&$$

$$\frac{}{\vdash a, a^\perp} \text{ini} \quad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta} \text{cut}$$

$$\frac{}{\vdash \mathbf{1}} \mathbf{1} \quad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \quad \frac{\vdash \Gamma, A_1}{\vdash \Gamma, A_1 \oplus A_2} \oplus_1 \quad \frac{\vdash \Gamma, A_2}{\vdash \Gamma, A_1 \oplus A_2} \oplus_2$$

$$\frac{}{\vdash \Gamma, \top} \top \quad \frac{\vdash \Gamma}{\vdash \Gamma, \perp} \perp \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \wp \quad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \&$$

$$\frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x. A} \exists \quad \frac{\vdash \Gamma, A[c/x]}{\vdash \Gamma, \forall x. A} \forall \quad \text{provided } c \text{ is new}$$

$$\frac{\vdash ?\Gamma, B}{\vdash ?\Gamma, !B} ! \quad \frac{\vdash \Gamma, B}{\vdash \Gamma, ?B} ?d \quad \frac{\vdash \Gamma}{\vdash \Gamma, ?B} ?w \quad \frac{\vdash \Gamma, ?B, ?B}{\vdash \Gamma, ?B} ?c$$

positive/negative duality

Proof Search by Cut-Elimination: Interactive Proof Search

Proof search is specified using non-uniform components:

- a language of formulas and a set of inferences (LK, LJ, LL, ...);
- a grammar for program clauses and goals;
- a search strategy (uniform proofs) + pruning heuristics (cut).

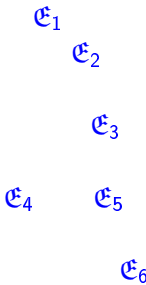
The mismatch may be more fundamental: We actually work with ***unfinished/uncompleted proofs*** which are not objects of the theory of sequent calculus. For instance, pruning operators prune the “***search space***”.

For instance:

- How to use a ***failed search*** for future computations?
- How to use the ***past computations*** in order to improve the next computations?

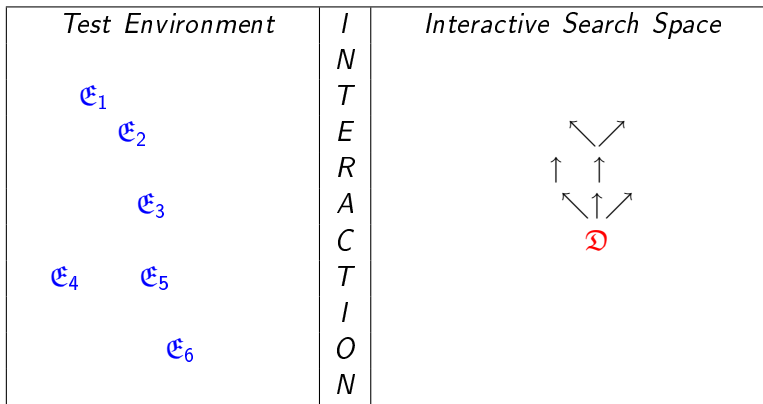
- A very natural approach, back to 1986 (*Van Emden, 1986*);
- Yet, much less investigated than the game-semantical approaches to functional programming;
- Ludics [Girard, 2001] gives a status to partial proofs:
 - There are ***partial proofs***;
 - There are both ***proofs*** and ***counter-proofs: the designs***;
 - Ludics theory is ***interactive***;
 - A good candidate for an interactive approach to logic programming?
 - Some ludics keywords: *monism, focalization, locations, tests through normalization, behaviours, ...*

Interactive Proof-Search, in Principle

<i>Test Environment</i>	<i>I N T E R A C T I O N</i>	<i>Interactive Search Space</i>
		

Duality test environment/interactive search space.

Interactive Proof-Search, in Principle



Duality test environment/interactive search space.

$F ::= F \otimes F \mid F \oplus F \mid \mathbf{1} \mid \mathbf{0}$ (positive fomulas)
 $F \wp F \mid F \& F \mid \perp \mid \top$ (negative fomulas)

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta} \text{ cut}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \quad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_0 \oplus A_1} \oplus_i \quad i \in \{0, 1\} \quad \overline{\vdash \mathbf{1}} \mathbf{1}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \& \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \wp \quad \overline{\vdash \Gamma, \top} \top \quad \frac{\vdash \Gamma}{\vdash \Gamma, \perp} \perp$$

$$F ::= F \otimes F \mid F \oplus F \mid \mathbf{1} \mid \mathbf{0} \quad (\text{positive fomulas})$$

$$F \wp F \mid F \& F \mid \perp \mid \top \quad (\text{negative fomulas})$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta} \text{ cut}$$

$$\boxed{\vdash \Gamma} \boxtimes$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \otimes \quad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_0 \oplus A_1} \oplus_i \quad i \in \{0, 1\} \quad \overline{\vdash \mathbf{1}} \mathbf{1}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \& \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \wp B} \wp \quad \overline{\vdash \Gamma, \top} \top \quad \frac{\vdash \Gamma}{\vdash \Gamma, \perp} \perp$$

In \boxtimes , Γ contains no negative formula.

MALL \boxtimes formulas will be given addresses to distinguish occurrences:

$$\mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})$$

$$\mathfrak{D}_i = \frac{\frac{\overline{\vdash \mathbf{1}_0} \quad \mathbf{1}}{\vdash \mathbf{1}_0} \quad \frac{\frac{\overline{\vdash} \quad \boxtimes}{\vdash \perp_{1i}} \quad \perp}{\vdash \perp_{10} \oplus_1 \perp_{11}} \oplus_i}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \&, i \in \{0, 1\}$$

$$\mathfrak{D}_i = \frac{\frac{\overline{\vdash \mathbf{1}_0} \quad \mathbf{1} \quad \frac{\frac{\overline{\vdash} \quad \boxtimes}{\vdash \perp_{1i}} \quad \perp}{\vdash \perp_{10} \oplus_1 \perp_{11}} \oplus_i}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \&, i \in \{0, 1\}$$

Used to build, by interaction:

$$\frac{\frac{\overline{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \quad \frac{\overline{\vdash \perp_0 \oplus_{\langle \rangle} (\mathbf{1}_{10} \&_1 \mathbf{1}_{11})}}{\vdash} \text{cut}}{\vdash} \text{cut}$$

$$\downarrow \text{cut-elim}$$

$$\overline{\vdash} \quad \boxtimes$$

Interacting with

$$\mathfrak{D}_i = \frac{\overline{\vdash \mathbf{1}_0} \quad \mathbf{1} \quad \boxed{\frac{\overline{\vdash} \quad \boxtimes \quad \perp}{\vdash \perp_{1i}} \quad \perp}{\vdash \perp_{10} \oplus_1 \perp_{11}} \oplus_i}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \&, i \in \{0, 1\}$$

can lead to

$$\mathfrak{D} = \frac{\overline{\vdash \mathbf{1}_{10}} \quad \mathbf{1} \quad \overline{\vdash \mathbf{1}_{11}} \quad \mathbf{1}}{\vdash \mathbf{1}_{10} \&_1 \mathbf{1}_{11}} \&}{\vdash \perp_0 \oplus_{\langle \rangle} (\mathbf{1}_{10} \&_1 \mathbf{1}_{11})} \oplus_1$$

Interacting with

$$\mathfrak{D}_i = \frac{\boxed{\overline{\vdash \mathbf{1}_0} \quad \mathbf{1}} \quad \frac{\overline{\vdash} \quad \text{✕} \quad \perp}{\overline{\vdash \perp_{1i}}} \quad \oplus_i}{\overline{\vdash \mathbf{1}_0} \quad \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \quad \&, i \in \{0, 1\}$$

can lead to

or to

$$\mathfrak{D} = \frac{\overline{\vdash \mathbf{1}_{10}} \quad \mathbf{1} \quad \overline{\vdash \mathbf{1}_{11}} \quad \mathbf{1}}{\overline{\vdash \mathbf{1}_{10} \&_1 \mathbf{1}_{11}}} \quad \& \quad \oplus_1}{\overline{\vdash \perp_0} \quad \oplus_{\langle \rangle} (\mathbf{1}_{10} \&_1 \mathbf{1}_{11})} \quad \oplus_1$$

$$\mathfrak{D}' = \frac{\overline{\vdash} \quad \text{✕} \quad \perp}{\overline{\vdash \perp_0}} \quad \oplus_0}{\overline{\vdash \perp_0} \quad \oplus_{\langle \rangle} (\mathbf{1}_{10} \&_1 \mathbf{1}_{11})} \quad \oplus_0$$

But \mathfrak{D}' uses a ✕: it is a failure.

How to avoid this second interaction for \mathfrak{D}' ?

One could add new tests to the environment:

$$\mathfrak{D}_2 = \frac{\frac{\overline{\vdash \perp_{10} \oplus_1 \perp_{11}} \quad \text{⊗}}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \quad \&|_1}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})}$$

Adding \mathfrak{D}_2 would have **forbidden** the search that leads to a failure by forcing the selection of \oplus_1 .

Slice of the $\&$ rule.

Ludics contains appropriate ingredients to represent MALL \boxtimes proofs.

$$\mathcal{D}_2 = \vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11}) \quad \boxtimes \quad \&|_1$$

The partial inference rule $\&|_1$ is a **negative** rule with **active formula** indexed by $\langle \rangle$ producing one **subformula** located in 1.

This can be summarized in $(\langle \rangle, 1)^-$:

$$\frac{\boxtimes}{(\langle \rangle, 1)^-}$$

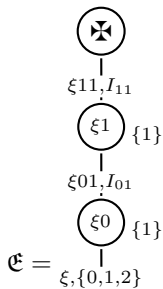
From MALL \boxtimes to Designs.

$$\mathfrak{D}_i = \frac{\frac{\frac{\overline{\vdash \mathbf{1}_0} \quad \mathbf{1}}{\vdash \mathbf{1}_0} \quad \frac{\frac{\overline{\vdash \perp_{1i}} \quad \perp}{\vdash \perp_{1i}} \quad \perp}{\vdash \perp_{10} \oplus_1 \perp_{11}} \oplus_i}{\vdash \mathbf{1}_0 \&_{\langle \rangle} (\perp_{10} \oplus_1 \perp_{11})} \&, i \in \{0, 1\} \longrightarrow$$

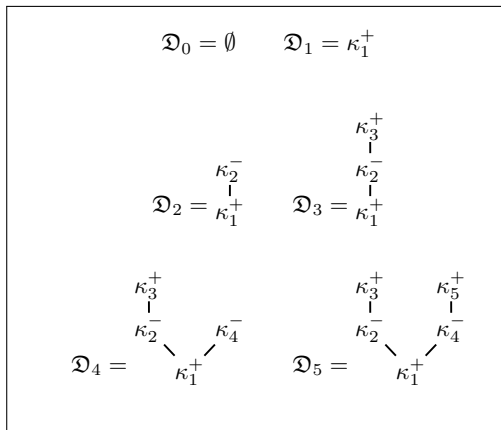
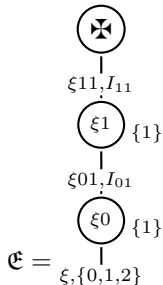
$$\mathfrak{D} = \frac{\frac{\frac{\overline{\vdash \mathbf{1}_{10}} \quad \mathbf{1}}{\vdash \mathbf{1}_{10}} \quad \frac{\overline{\vdash \mathbf{1}_{11}} \quad \mathbf{1}}{\vdash \mathbf{1}_{11}}}{\vdash \mathbf{1}_{10} \&_1 \mathbf{1}_{11}} \&}{\vdash \perp_0 \oplus_{\langle \rangle} (\mathbf{1}_{10} \&_1 \mathbf{1}_{11})} \oplus_1 \longrightarrow$$

Those designs can interact. But why to go through MALL \boxtimes and *not directly interact(ively search) in Ludics?*

Scheme of IPS on an Example.

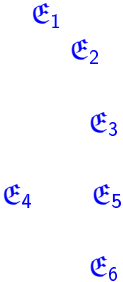
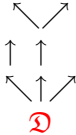


Scheme of IPS on an Example.

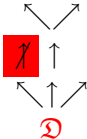


This interactive search mechanism is defined thanks to an abstract machine, the SLAM.

Backtracking, Interactively

<i>Test Environment</i>		<i>Interactive Search Space</i>
	I N T E R A C T I O N	

Backtracking, Interactively

<i>Test Environment</i>		<i>Interactive Search Space</i>
\mathfrak{E}_1 \mathfrak{E}_2 \mathfrak{E}_3 \mathfrak{E}_4 \mathfrak{E}_5 \mathfrak{E}_6 $\text{Backtrack}(\mathfrak{D})$	I N T E R A C T I O N	

Using the interaction paths for \mathfrak{D}

Next step:

Consider other usual pruning mechanisms in Prolog and see how "interactive" they can be made:

- `!/0`
- soft cut
- other backtracking modifiers
- intelligent backtracking
- ...

- Relate cut-elimination and proof search;
- An interactive framework for proof search;
- Enriching the search environment by adding more tests as computation goes on;
- Syntax/Semantics: due to the “monistic” approach of Ludics, we get an abstract evaluation framework (SLAM) and we benefit from ludics semantical tools (behaviours, internal completeness);
- Much to make it more expressive: control/pruning operators, first order, exponentials, ...