Interactive Proof Search with Ludics: Short Abstract

Alexis Saurin⋆

1 Introduction: Searching for proofs interactively

Proof Theory and Computation. Recent developments in proof theory have led to major advances in the theory of programming languages. The modelling of computation using proofs impacted deeply the foundational studies of programming languages as well as many of their practical issues. Declarative programming languages have been related mainly in two ways to the mathematical theory of proofs: on the one hand, the "computation as proof normalization" paradigm provided a foundation for functional programming languages through the well-known Curry-Howard correspondence [9]. On the other hand the "computation as proof search" paradigm stands as a foundation for logic programming: the computation of a program is the search for a proof in some deductive system.

Computation as proof search. The proof search paradigm was at first founded on the resolution method: computation corresponded to the search for a resolution for first-order Horn clauses. But this approach was uneasy to extend to larger fragments or richer logics. Later, the use of sequent calculus allowed to overcome this limit: the introduction of Uniform Proofs and Abstract Logic Programming Languages [13] and the discovery of the Focalization Property [2] in Linear Logic [6] allowed to extend proof search to larger fragments (Hereditary Harrop formulas for instance), to richer logics (linear logic, ...) and to benefit from the geometrical properties of sequent calculus[1] which enriches the dynamics of the search and thus of the computation. This approach allowed to enrich logic programming with numerous additional programming primitives by capturing them logically (higher-order programming, modules, ...), nevertheless some essentials programming constructions could not be dealt with logically, especially regarding the control of computation (backtracking, `cut` predicate...). One of the long-standing research directions on proof search is to treat the extra-logical primitives in a logical way in order to get closer to the "ideal" correspondence: "Algorithm = Logic" [11]. We can draw a useful and enlightening comparison with functional programming: the extension of the Curry-Howard correspondence to classical logic allowed to capture logically several control operators that were used in practice (like `call/cc`) thanks to typing rules for those operators [8] or thanks to extensions of λ-calculus such as $\lambda\mu$-calculus [14].

Why is it so difficult to have a logical treatment of control? While in sequent calculus, we manipulate proofs, the process of searching for and constructing proofs does not deal with proofs until the computation is finished. Instead, the objects of proof search are partial proofs which may end up not leading to a proof at all but to a failure. Such failed proofs are not part of the proof theory of sequent calculus. Thus, the very dynamics of computation does not fit well with the theory we work with.

Ludics and Interaction. We noticed earlier the strong relationship between logic and computation. We can thus expect that new developments in proof theory will provide new methods and maybe new paradigms on the side of computation. From this perspective we think that the recent work by Girard on Ludics [7] should be considered as a major step. Ludics is a logical theory that attempts to overcome the distinction between syntax and semantics

⋆ INRIA & École Polytechnique, *saurin@lix.polytechnique.fr*

[1] In the uniform proofs model for instance, computation will be modelled as a search for a proof of a sequent $\mathcal{P} \vdash G$ where $\mathcal{P}$ represents the logic program and G the computation goal. The computation then proceeds as a search for a proof directed by the goal G, the logic program $\mathcal{P}$ being used through backchaining when the goal is an atomic formula.

by considering that interaction comes first and by building syntax and semantics afterwards, thanks to interaction. Ludics objects, designs, can be seen as intermediate objects "between" syntax and semantics. Ludics is founded on many concepts of (linear logic [6]) proof theory and of proof search [12] especially the fundamental result of Focalization [2] that allows for synthetic connectives to be defined. Ludics has lots of connections with game semantics [1, 10] as well since the interaction process in Ludics can be seen as a play [5]. In recent years, interaction has become a crucial concept of the theory of computation. We think that Ludics provides promising tools in order to develop a study of logic programming that would benefit from those concepts and tools from interactive theories, such as an explicit treatment of the computational environment which is uniform to the computational objects [3].

Searching for proofs interactively The sequent $\mathcal{P} \vdash G$ is the current state of the computation but it is also a way to constrain the future of the computation. In the same way, restrictions on the logical rules that are allowed (like in linear logic) or proof strategies also impose constraints on proof search. But all this is implicit and not done explicitly. In particular, it is fairly difficult to analyze these constraints in proof theory itself, even though that would be useful since important programming primitives precisely rely on these constraints. The interactive approach to proof search we are investigating precisely makes explicit the constraints on proof search: instead of building a proof depending on a given sequent, we shall consider building a proof that shall pass some tests, that shall be opposed to attempts to refute it. The tests will have the form of (para)proofs and thus will be built in the same system as the one in which we are searching for proofs. We propose a computational setting which would correspond to the following:

- We are willing to search for a proof $\mathfrak{D}$ of $\vdash A$;
- Formula A is actually given as a set of tests (which is the set of counter-arguments that one should be able to answer): the tests that shall be successfully passed by the proof we are constructing: $\mathfrak{E}_1, \ldots, \mathfrak{E}_n$;
- Proof construction shall proceed by consensus with the tests: $\mathfrak{D}$ can be extended with rule R only if the extended object interacts well with the tests;
- After some interactions, we may have an object that cannot be extended any more. Either the construction is terminated because $\mathfrak{D}$ cannot pass every test or because all the tests are satisfied and no more constraint applies to $\mathfrak{D}$ so that there is no need to extend it further. In the first case, we have a failure while in the later case we have a success
- One interesting point with this interactive approach lies in the fact that the setting is symmetrical: $\mathfrak{D}$ is tested by $\mathfrak{E}_i$ but it is also a test for the $\mathfrak{E}_i$s. In particular, even failures are interesting and useful objects. If the result of a computation is a failure $\mathfrak{D}^{\maltese}$, we may be willing to try another search. Indeed, maybe at some point we chose a wrong way to extend $\mathfrak{D}$ and that caused the failure whereas another way to build $\mathfrak{D}$ could have led to a win. There is a standard way to backtrack, that is erase some part of $\mathfrak{D}^{\maltese}$ and try some other construction. But since we are in an interactive setting, there is another option: we can try to use $\mathfrak{D}^{\maltese}$ in order to provide more tests: $\mathfrak{E}_j^{\mathfrak{D}^{\maltese}}$ that will constrain the search to look for a proof that shall be different from the failure $\mathfrak{D}^{\maltese}$.

The present abstract will only be concerned with motivating further the interactive search and providing examples. The reader may find more technical discussions, definitions and results in [16].

2 Motivations and intuitions for Ludics

Ludics has recently been introduced by Girard [7] as an interactive theory that aims at overcoming the traditional distinction between syntax and semantics by considering that interaction should come first and logic shall be reconstructed from this interactive approach. This new theory is built on the notion of interaction of designs which are intermediate objects between syntax and semantics: they can be viewed both as an abstraction of MALL sequent proofs and as a concrete presentation of game semantics strategies. Thus, one does not start from syntactic objects to which is assigned a semantic interpretation, nor does one begin with a semantic space for which we need an adequate language. They are objects interacting with each other and their properties are defined interactively. Of course, Ludics is not built by sweeping away previous advances in logic. On the contrary it comes from a careful analysis of logic and a clever synthesis so to obtain the appropriate notion of interaction. In our presentation, we try to emphasize connections towards logic programming and proof search.

$$F ::= a \mid F \otimes F \mid F \oplus F \mid \mathbf{1} \mid \mathbf{0}$$
$$\quad a^{\perp} \mid F \parr F \mid F \,\&\, F \mid \perp \mid \top$$

$$\frac{}{\vdash \Gamma}\ \maltese \quad \Gamma \text{ contains no negative formula.}$$

Fig. 1. $MALL$ **formulas and the ✠ rule.**

Focalization. Andréoli's Focalization [2] is the root of a polarized approach to logic which is the first step towards a game theoretic interpretation of proofs: polarization tells you whose turn it is to play. Focalization allows to define synthetic connectives and synthetic rules in multiplicative additive linear logic (MALL) and a hypersequentialized calculus using the synthetic rules. MALL connectives can be classified in two sets of connectives: positive connectives ($\otimes, \oplus, \mathbf{1}, \mathbf{0}$) and negative ones ($\parr, \&, \perp, \top$). When searching for a proof, one alternates between positive and negative phases. Provability cannot be lost during the negative phase while during the positive phase we can make a wrong choice, ending up not finding a proof even if the sequent we started with was provable. Thus there is clearly an active phase (positive) and a passive phase (negative) and the two phases alternate.

Proof Normalization. The cut elimination process reflects this game interpretation: a conversion step corresponds to the selection, by the positive rule, of a continuation for the normalization (*e.g.* selection of one of the &-premises by a $\oplus$-rule during cut-elimination). But there is still a problem for an interactive interpretation to hold: there are not enough proofs! We cannot find a proof for both A and $A^{\perp}$. Notice that if there cannot be proofs for both a formula and its negation, it is perfectly legal to attempt to prove both A and $A^{\perp}$. If the proof search fails, the partial object that we have obtained can be used in an interaction with proof attempts of the negation, except where the failure was encountered (here normalization is still undefined). A failed attempt to prove A is a tree with some open branches. Let us add a new rule, the ***daimon***, to mark the fact that the search for a proof has been stopped: $\frac{}{\vdash \Gamma}\ \maltese$. We thus have paraproofs for any sequents, even for $\vdash$.

Winning and loosing. The normalization between two paraproofs is a process through which they test each other. The one that is caught using ✠ is considered as the loser of the play and the play ends there. Notice that this normalization process is an exploration of the two paraproofs: the cut visits some parts of the paraproofs. When ✠ is reached, the paraproofs are said to be ***orthogonal***.

Locations. Whereas in functional programming it is important to know if the types of a function and its argument match, it is not relevant for proof search to know the complete structure of the proof from the beginning. We only need to know enough to choose a rule to apply. This idea is reflected in Ludics by the use of addresses or locations (or ***loci***). A formula is now manipulated through its address ξ. When we apply a rule R on ξ we come to know *where* the subformulas (or ***subloci***) made available by R are (not *what* they are): $\xi i, \xi j, \ldots$

Behaviours and Incarnation. A provable formula may be interpreted as the set of its proofs or paraproofs. Actually things are even more drastic in Ludics: formulas are defined interactively by a standard technique of biorthogonality closure which defines the ***behaviours***. Given a paraproof Π in behaviour $\mathbf{A}$ and a paraproof Π' in $\mathbf{A}^{\perp}$, a part of Π can be explored by normalization with Π'. Under special circumstances, it may happen that Π is entirely visited by Π'. Usually, there are parts of Π that cannot be explored, whatever $\Pi' \in \mathbf{A}^{\perp}$ you test it with. However, a class of paraproofs which is highly interesting from our interactive perspective is the class of paraproofs that can be completely visited during normalization with elements of $\mathbf{A}^{\perp}$. They are said to be ***material***.

3 Searching for proofs interactively in MALL.

MALL✠. If we want to search for proofs by interaction, we need to have objects to interact with, but we never have a proof for A and $A^{\perp}$ for any formula A. This means that we need to extend the logic in order to have more proofs and more provable formulas. In the remaining of the section, we consider proofs in $MALL$✠. They are built from $MALL$-formulas by adding the ✠ rule to the usual $MALL$ proof system, see Figure 1.

An example of IPS in MALL✠. We can look for a paraproof $\mathfrak{D}$ that would pass the tests $\mathfrak{D}_0$ and $\mathfrak{D}_1$ which are paraproofs of sequent[2] $\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})$:

$$\mathfrak{D}_i = \dfrac{\dfrac{}{\vdash \mathbf{1}_0}\ \mathbf{1} \qquad \dfrac{\dfrac{\dfrac{}{\vdash}\ \maltese}{\vdash \bot_{1i}}\ \bot}{\vdash \bot_{10} \oplus_1 \bot_{11}}\ \oplus_i}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\ \&$$

$\mathfrak{D}$ shall be a proof of the sequent $\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})$ such that the paraproof Π_i $(i \in \{0,1\})$ built by cutting $\mathfrak{D}$ with one of the $\mathfrak{D}_i$s:

$$\dfrac{\dfrac{\mathfrak{D}_i}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})} \qquad \dfrac{\mathfrak{D}}{\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})}}{\vdash}\ cut$$

can normalize to $\dfrac{}{\vdash}\ \maltese$. Performing the cut reduction will impose constraints on $\mathfrak{D}$ that can be used as a guide to search for a paraproof on $\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})$. We end up with:

$$\mathfrak{D} = \dfrac{\dfrac{\dfrac{}{\vdash \mathbf{1}_{10}}\ \mathbf{1}^{\star_0} \qquad \dfrac{}{\vdash \mathbf{1}_{11}}\ \mathbf{1}^{\star_1}}{\vdash \mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11}}\ \&}{\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})}\ \oplus_1$$

The branch ending at $\star_i$ has been built by interacting with $\mathfrak{D}_i$. When normalizing $\mathfrak{D}$ with $\mathfrak{D}_i$, the $\maltese$ used by $\mathfrak{D}_i$ informally means that the paraproof $\mathfrak{D}$ was able to find an error in $\mathfrak{D}_i$, without letting $\mathfrak{D}_i$ discover its potential logical leakage. Another normalization could have led to:

$$\mathfrak{D} = \dfrac{\dfrac{\dfrac{}{\vdash}\ \maltese}{\vdash \bot_0}\ \bot}{\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})}\ \oplus_0$$

But here $\mathfrak{D}$ uses a $\maltese$ and thus is a failure.

Beyond MALL✠ Finally, one could even have imagined adding the following (a bit peculiar) paraproofs $\mathfrak{D}_2$ and $\mathfrak{D}_3$ (which use only partial $\&$ proof rules) to the set of tests:

$$\dfrac{\dfrac{}{\vdash \mathbf{1}_0}\ \maltese}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\ \&|_0 \qquad \dfrac{\dfrac{}{\vdash \bot_{10} \oplus \bot_{11}}\ \maltese}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\ \&|_1$$

If $\mathfrak{D}_2$ is in the normalization environment then $\mathfrak{D}$ ***is forced*** to use $\oplus_0$ as a first rule while interactive search with $\mathfrak{D}_3$ would have forbidden the search that leads to a failure by forcing the selection of $\oplus_1$. $\mathfrak{D}_2$ and $\mathfrak{D}_3$ can thus be used to forbid some interactions to occur. This will be clearer when moving to Ludics. This brief study shows that there are lots of possibilities to guide (or constrain) proof search interactively. However, it is needed to relax some of the logical principles. For instance, it is needed to add the daimon $\maltese$ which allows to prove any sequent, but it is also important to admit "partial" logical rules (see $\mathfrak{D}_2$ and $\mathfrak{D}_3$) and other principles of (linear) logic shall be reconsidered (the weakening for instance). This is one of the reasons why we go to Ludics which is a clean theory with a good theory of interaction.

4 Interactive proof search algorithm.

[2] We index the intermediate formulas to distinguish them more easily, anticipating what will be done later in Ludics.

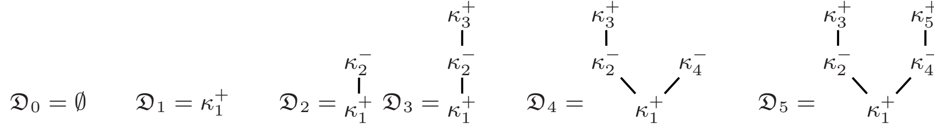
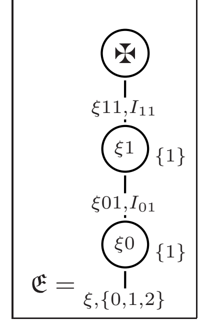


Fig. 2. Interactive search for $\mathfrak{D}$.



In this section, we outline an algorithm for Interactive Proof Search and refer the reader to [16] for more details and in particular for the definition and the study of the abstract machine, the Searching LAM (SLAM) which allows us to interactively build material designs in a behaviour generated by orthogonality to a set of tests. The reader will also find an example of an interactive search.

We will not do more than sketching how IPS works on a simple example in order to show the main structure of the search: consider the Interactive Proof Search driven by one very simple design. Let us proceed with an IPS with environment $\{\mathfrak{E}\}$ in order to build a design $\mathfrak{D}$.

0. To begin with, $\mathfrak{D}_0$ is empty and we have visited an empty path: $Path_0 = \epsilon$;

1. $\mathfrak{E}$ is a negative design so that it is a forest: it may begin with several negative actions on focus ξ, one of which shall be followed during a normalization process. Its initial actions are in $\text{Init}_{\mathfrak{E}} = \{(\xi, \{0,1,2\})^-\}$. We choose some action κ_1^- in $\text{Init}_{\mathfrak{E}}$ and add $\kappa_{1\nu}$ to the normalization path and κ_1^+ as the first action of $\mathfrak{D}$: $Path_1 = \langle(\xi, \{0,1,2\})\rangle = \langle\kappa_{1\nu}\rangle$;

2. Design $\mathfrak{D}$ in construction could have several negative actions above κ_1^+ but at this point, normalization would follow only one action which corresponds to the positive action after κ_1^- in $\mathfrak{E}$. This action is $\kappa_2^+ = (\xi 0, \{1\})^+$ and thus: $Path_2 = \langle\kappa_{1\nu}, \kappa_{2\nu}\rangle$;

3. In $\mathfrak{E}$, κ_2^+ is followed by actions in $\{(\xi 01, I_{01})^-\}$, we choose κ_3^- in this set and we extend $Path_2$ with $\kappa_{3\nu}$ and $\mathfrak{D}$ with κ_3^+: $Path_3 = \langle\kappa_{1\nu}, \kappa_{2\nu}, \kappa_{3\nu}\rangle$;

4. In $\mathfrak{E}$, κ_3^- is followed by the only action $\kappa_4^+ = (\xi 1, \{1\})^+$ and thus, $Path_3$ shall be extended with $\kappa_{4\nu}$. $\mathfrak{D}_4$ must contain κ_4^- in order to interact properly, but this negative action shall be placed right after the positive action justifying it. The branch leading to κ_4^- in $\mathfrak{D}$ is given by: $\ulcorner\kappa_{1\nu}, \kappa_{2\nu}, \kappa_{3\nu}, \kappa_{4\nu}\urcorner^- = \kappa_1^+, \kappa_4^-$ and thus: $Path_4 = \langle\kappa_{1\nu}, \kappa_{2\nu}, \kappa_{3\nu}, \kappa_{4\nu}\rangle$;

5. In $\mathfrak{E}$, $Succ(\kappa_4^+) = \{(\xi 11, I_{11})^-\}$. We extend the dispute: $Path_5 = \langle\kappa_{1\nu}, \dots, \kappa_{5\nu}\rangle$ and we add κ_5^+;

6. In $\mathfrak{E}$, κ_5^- is followed by a unique action, $\kappa_6^+ = \maltese$. The normalization ends with $\mathfrak{E}$ using a $\maltese$ and the final dispute is $[\mathfrak{D} \rightleftharpoons \mathfrak{E}] = \langle\kappa_{1\nu}, \dots, \kappa_{5\nu}, \maltese\rangle$.

$\rightarrow$ After the IPS process, we built a design $\mathfrak{D}$ on $\vdash \xi$ such that $[\![\mathfrak{D}, \mathfrak{E}]\!] = \maltese$ with the daimon caused by $\mathfrak{E}$.

References

1. S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *JSL*, 59(2):543–574, 1994.
2. J-M. Andreoli. Logic programming with focusing proofs in linear logic. *JLC*, 2(3):297–347, 1992.
3. P-L. Curien and H. Herbelin. The duality of computation. In *ICFP*, p 233–243, 2000.
4. C. Faggian. Travelling on designs. In *Proceedings of CSL'02*, pages 427–441, 2002.
5. C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL*, pages 442–457, 2002.
6. J-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
7. J-Y. Girard. Locus solum. *MSCS*, 11(3):301–506, June 2001.
8. T. Griffin. A formulae-as-types notion of control. In *POPL'90*, pages 47–58, 1990.
9. W. A. Howard. The formulae-as-type notion of construction, 1969. *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, p. 479–490. Academic Press, 1980.
10. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–408, 2000.
11. D. Miller. Sequent calculus and the specification of computation. In *Computational Logic*. Springer, 1999.
12. D. Miller. Overview of linear logic programming. In *LL in Computer Science*, p.119–150. Cambridge Univ. Press, 2004.
13. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
14. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. *ICLP'92*, vol. 624 of *LNCS*, 1992.
15. D. J. Pym and E. Ritter. A game semantics for reductive logic and proof-search. *GALOP'05*.
16. A. Saurin. Towards Ludics Programming: Interactive Proof Search. *submitted*, january 2008, `http://www.lix.polytechnique.fr/~saurin/Recherche/Publi/prolud_soumis.pdf`.