Towards Ludics Programming: Interactive Proof Search

Alexis Saurin*

Abstract. Girard [10] introduced Ludics as an interactive theory aiming at overcoming the distinction between syntax and semantics in logic. In this paper, we investigate how ludics could serve as a foundation for logic programming, providing a mechanism for interactive proof search, that is proof search by interaction (or proof search by cut-elimination).

1 Introduction.

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 by providing formal tools to analyze programs properties. 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 [12]. 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. Uniform Proofs and Abstract Logic Programming Languages [19] and Focalization [2] in Linear Logic [9] allowed to consider computation as proof search for much richer fragments of logic than first-order Horn clauses with resolution (Hereditary Harrop formulas, higher order, linear logic) and to benefit from the structure of sequent calculus which enrich the dynamics of proof search. This impacted deeply the design of logic programming languages by allowing to model various programming primitives logically (HO programming, modules, resource management, concurrent primitives, ...). Nevertheless some essential programming constructions could not be dealt with logically, in particular when concerned with the control of computation [21, 22] (`cut` predicate, (intelligent) backtracking, ...). As a consequence, some parts of the languages do not have a very well established nor declarative semantics, and thus it is difficult to analyze programs using those constructions even though they are extremely common in Prolog programming. A long-standing research direction on proof search is to treat those extra-logical primitives in a logical way in order to get closer to the "ideal" correspondence: "Algorithm = Logic" [17].

We can draw a 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 [11] or thanks to extensions of λ-calculus such as $\lambda\mu$-calculus [23].

* INRIA-Futurs & École polytechnique (LIX), *saurin@lix.polytechnique.fr*

However, corresponding extensions in logic programming could not be achieved up to now, this may be understood as the result of a mismatch between sequent calculus proof theory and logic programming: while in sequent calculus, we manipulate proofs, the process of searching for proofs does not deal with proofs until the computation is finished. Instead, the objects of proof search are partial proofs (or open 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.

Ludics and Interaction. Girard introduced Ludics [10] in which unfinished proofs are given a clear status being at the heart of this theory of interaction. Ludics is a logical theory that attempts to overcome the distinction between syntax and semantics 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 [9]) proof theory and of proof search [18] and of game semantics [1, 13] as well since ludics object can be seen as innocent strategies and the interaction process in Ludics as a play [6].

Games and logic programming. Game-theoretic approaches to logic programming are fairly natural and however not much developped. Van Emden was the first to notice the connections between logic programming computations and two-person games with $\alpha\beta$-algorithm [26], which was later studied in more details by Loddo et al. [3, 16, 15]. Pym and Ritter [24, 25] proposed a game semantics for uniform proofs and backtracking by relating intuitionnistic and classical provability while the author, Miller and Delande [20, 4] developed a neutral approach to proofs and refutations based on games which was inspired by Prolog search engine [20]. More recently, Galanaki et al. [8] generalized van Emden's games for logic programs with (well-founded) negation.

Structure of the paper. In this paper we investigate the use of Ludics as a foundation for proof search and logic programming by means of a model of interactive proof search (IPS). We first draw in Section 2 the general picture of this paradigm of "computation as interactive proof search" (or proof search as cut-elimination) and illustrate this approach on a sequent calculus derived from linear logic. In Section 3, we introduce the basic definitions of Ludics. The heart of the paper is Section 4: we present an abstract machine for IPS, the SLAM, establish some of its properties and finally explain how backtracking can be elegantly dealt with.

2 Logic programming, interactivity and Ludics

In the uniform-proof model, computation is modelled as a search for a proof of a sequent $\mathcal{P} \vdash G$ which is directed by the goal G, the logic program $\mathcal{P}$ being used only when the goal is an atomic formula. While the dynamics of proof search is concerned with partial proofs, sequent calculus theory is a theory of complete proofs: it is thus difficult to speak about failures, backtrack or pruning of the search tree [22] (like the `cut` in Prolog) in this setting.

We propose another approach which considers proof search interactively.

2.1 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. These constraints on the dynamics of proof search are of different kinds and are uneasy to relate and compare.

The interactive approach to proof search we are investigating precisely aims at providing a uniform framework for expressing and analyzing 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.

IPS computation. We shall develop a computational setting as follows. We are willing to search for a proof $\mathfrak{D}$ of $\vdash A$. Formula A is described as a set of tests: $(\mathfrak{E}_i)_{i \in I}$. 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 all the tests. At some point, it may become impossible to extend $\mathfrak{D}$ further: *(i)* either $\mathfrak{D}$ cannot pass some of the tests *(ii)* or all tests are satisfied and no more constraint applies to $\mathfrak{D}$ so that there is no need (and no guideline) to extend it further. Case *(i)* is a failure while case *(ii)* is a success. In case of a failure $\mathfrak{D}_{\maltese}$, one may try another search. Apart from backtracking up to the last choice point and restarting the search, there is another option: to use $\mathfrak{D}_{\maltese}$ in order to provide new tests $\mathfrak{E}_j^{\mathfrak{D}_{\maltese}}$ to constrain the search even more.

2.2 Motivations and intuitions for Ludics

We describe ludics intuitions, with connections towards logic programming:
Monism. Ludics has been introduced by Girard [10] 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 afterwards: designs can be viewed both as an abstraction of multiplicative additive linear logic (MALL) sequent proofs (syntactical viewpoint) and as a concrete presentation of game semantics innocent strategies [6] (semantical viewpoint).
Focalization. Andreoli's Focalization [2] is the root of a polarized approach to logic [14] and allows to define synthetic connectives and synthetic rules (which are clusters of connectives or rules of the same polarity) in MALL. MALL connectives can be classified in two sets of connectives: positive connectives ($\otimes, \oplus, \mathbf{1}, \mathbf{0}$) and negative ones ($\parr, \&, \bot, \top$). Provability cannot be lost during the negative phase while it can be during the positive phase by making a wrong choice of rule. Thus there is clearly an active phase (positive) and a passive phase (negative) and when searching for a proof, one alternates between those two phases.

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

$$\frac{}{\vdash \Gamma, \top}\ \top \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ \bot \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\ \& \qquad \frac{}{\vdash \Gamma}\ \maltese$$

Fig. 1. $MALL\maltese$ sequent calculus. (In $\maltese$, Γ contains no negative formula)

Proof Normalization. In the cut elimination process, 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: 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 $A^{\perp}$ [20, 4]. 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 $\vdash$.

The normalization between two paraproofs is a ***process through which they test each other***. The one that is caught using $\maltese$ 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 $\maltese$ is reached, the paraproofs are said to be ***orthogonal***. A paraproof that wins an interaction may still contain itself a daimon: it is simply not part of this precise interaction, but would be detected by some other interactions.

Locations. Whereas in functional programming it matters to know if the types of functions and arguments match, it is not relevant for proof search to know the complete structure of formulas to be proved: we only need to know enough to choose the next rule. In Ludics, we use addresses (or ***loci***): a formula is dealt with through its address ξ and an inference rule R on ξ creates ***subloci*** ($\xi i, \xi j, \dots$) which refer to *where* the subformulas are (not *what* they are).

Behaviours. A provable formula may be interpreted as the set of its proofs or rather its 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'. Sometimes Π is entirely visited by some Π' but 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 is the class of paraproofs that can be completely visited during normalization with elements of $\mathbf{A}^{\perp}$, they are said to be ***material***.

2.3 Searching for proofs interactively in MALL.

Adding more proofs: *$MALL\maltese$*. If we want to search for proofs by interaction, we need to have enough proof objects to interact with, as noticed in 2.2: we need to extend logic in order to have more proofs and provable formulas. In the following, we consider $MALL\maltese$ proofs which are built from unit-only $MALL$-formulas ($F ::= F \otimes F \mid F \oplus F \mid \mathbf{1} \mid \mathbf{0} \mid F \parr F \mid F \,\&\, F \mid \bot \mid \top$) by adding the $\maltese$ rule to $MALL$ proof system, see Figure 1.

$$\mathfrak{D}_2 = \dfrac{\dfrac{}{\vdash \mathbf{1}_0}\,\maltese}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\,\&|_0 \qquad\qquad \mathfrak{D}_3 = \dfrac{\dfrac{}{\vdash \bot_{10} \oplus \bot_{11}}\,\maltese}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\,\&|_1$$

Fig. 2. MALL$\maltese$ proofs with partial inferences.

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

$$\mathfrak{D}_i = \dfrac{\dfrac{}{\vdash \mathbf{1}_0}\,\mathbf{1} \qquad \dfrac{\dfrac{\dfrac{}{\vdash}\,\maltese}{\vdash \bot_{1i}}\,\bot}{\vdash \bot_{10} \oplus_1 \bot_{11}}\,\oplus_i}{\vdash \mathbf{1}_0 \,\&\, (\bot_{10} \oplus_1 \bot_{11})}\,\& \qquad\qquad \text{with } i \in \{0,1\}$$

$\mathfrak{D}$ shall be a proof of sequent $\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})$ such that paraproofs built by cutting $\mathfrak{D}$ with any of the $\mathfrak{D}_i$s normalize:

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

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:

$$\mathfrak{D} = \dfrac{\dfrac{\dfrac{}{\vdash \mathbf{1}_{10}}\,\mathbf{1}^{\star_0} \quad \dfrac{}{\vdash \mathbf{1}_{11}}\,\mathbf{1}^{\star_1}}{\vdash \mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11}}\,\&}{\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})}\,\oplus_1 \qquad \text{or} \qquad \mathfrak{D}' = \dfrac{\dfrac{\dfrac{}{\vdash}\,\maltese}{\vdash \bot_0}\,\bot}{\vdash \bot_0 \oplus (\mathbf{1}_{10} \,\&_1\, \mathbf{1}_{11})}\,\oplus_0$$

In $\mathfrak{D}$, the branch ending at $\star_i$ has been built thanks to $\mathfrak{D}_i$. $\mathfrak{D}'$ is a failure.

Beyond MALL$\maltese$. Finally, one could even imagine adding paraproofs of figure 2 (which use only partial $\&$ proof rules) to the set of tests. 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 forbid the search of failure $\mathfrak{D}'$ 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 brief study shows the many 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 has a good theory of interaction.

The following section is devoted to the introduction of Ludics.

3 Introduction to Ludics.

Actions and Designs. In Ludics, proofs are replaced by designs and proof rules by actions while formulas are now accessed through their location.

[1] We index formulas to identify them more easily, anticipating on the use of addresses.

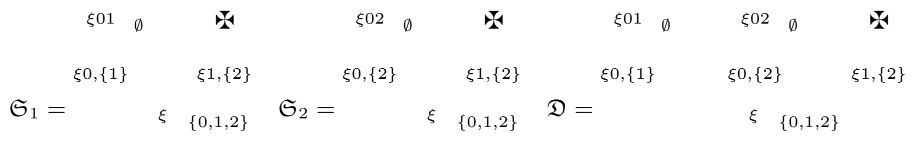


Fig. 3. Two slices and a design.

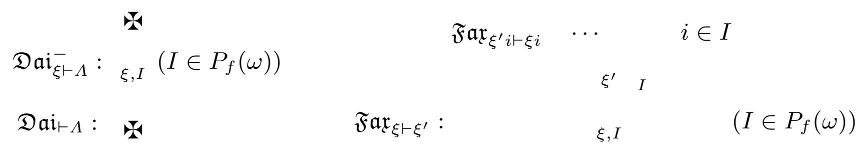

$\maltese$
$\mathfrak{Dai}^-_{\xi \vdash \Lambda}$:
ξ, I
$(I \in P_f(\omega))$
$\mathfrak{Dai}_{\vdash \Lambda}$:
$\maltese$
$\mathfrak{Fax}_{\xi' i \vdash \xi i}$
$\cdots$
$i \in I$
ξ'
I
$\mathfrak{Fax}_{\xi \vdash \xi'}$:
ξ, I
$(I \in P_f(\omega))$

Fig. 4. Important designs: $\mathfrak{Dai}, \mathfrak{Dai}^-$ & $\mathfrak{Fax}$.

An ***address*** (or ***locus***) is a finite sequence of integers (written ξ). An ***action*** is either a pair of an address (the ***focus***) and a finite set of integers together with a polarity (we write $(\xi, I)^+$ or $(\xi, I)^-$ and speak of ***proper actions***) or the ***daimon*** ($\maltese$) which is positive. When forgetting the polarity of a proper action κ, we speak of a ***neutral action*** and write κ_ν. We say that $(\xi, I)^\epsilon$ ***creates*** addresses $\xi \star i$ $(i \in I)$ and that action κ ***justifies*** κ' when they have opposite polarity and κ creates the focus of κ'. A ***base*** is a finite set of polarized addresses (ξ^+ or ξ^-) with at most one negative address and such that no address is prefix of another address. We write $\xi \vdash \Lambda$ (resp. $\vdash \Lambda$) for ***negative*** (resp. ***positive***) ***bases***. A singleton base ($\xi \vdash, \vdash \xi$) is ***atomic*** and $\ \vdash\ $ is the ***empty base***.

A ***design*** $\mathfrak{D}$ on a base β is a (possibly infinite) prefix-closed set of finite sequences of actions (*ie.* a forest of actions) such that:
Chronicles. Let $\chi = (\kappa_0, \dots, \kappa_n) \in \mathfrak{D}$. In χ, actions have alternating polarities and addresses occur at most once. If $\kappa_i \in \chi$, either $\kappa_i = (\xi, I)^\epsilon$ and $\xi^\epsilon \in \beta$ or κ_i is justified by κ_j with $j < i$ ($j = i - 1$ for κ_i negative) or $\kappa_i = \maltese$ and $i = n$;
Positivity. The leaves of the forest are positive;
Positive branching. The tree only branches on positive actions: if $\chi_1, \chi_2 \in \mathfrak{D}$ are not prefix of each other, they first differ on negative actions;
Additive sharing. If κ_0, κ_1 are distinct actions with the same focus then the sequences leading to κ_0 and κ_1 first differ on negative actions with same focus;
Totality. If the base is positive, $\mathfrak{D} \neq \emptyset$.

A design is ***positive or negative*** according to its base. A ***slice*** is a design where no address occurs twice. A slice of a design $\mathfrak{D}$ is any slice included in $\mathfrak{D}$. In particular, a negative slice is a tree. When drawing designs and slices, we adopt ***Faggian's convention***: positive actions are circled while negative actions (which are not branching) are not circled. We give in figure 3 and 4 examples of designs. Notice that $\mathfrak{D}$ (on base $\vdash \xi$) is the superimposition of $\mathfrak{S}_1$ and $\mathfrak{S}_2$.

Another approach to designs is as ***co-inductively generated*** by a grammar:
$\mathfrak{P} ::= \maltese^{\vdash \Gamma} \mid (\xi, I)^+ \cdot \{\mathfrak{N}_i^{\xi i \vdash \Gamma_i}, i \in I, i \neq j \Rightarrow \Gamma_i \cap \Gamma_j = \emptyset, \forall i \in I, \Gamma_i \subset \Gamma\}^{\vdash \xi, \Gamma}$
$\mathfrak{N} ::= \{(\xi, I)^- \cdot \mathfrak{P}_I^{\vdash \xi I, \Gamma_I}, I \in \mathcal{N} \subset \mathcal{P}_f(\omega), \forall I \in \mathcal{N}, \Gamma_I \subset \Gamma\}^{\xi \vdash \Gamma}$
For instance, $\mathfrak{Fax}_{\xi \vdash \xi'}$ is $\mathfrak{Fax}_{\xi \vdash \xi'} = \{(\xi, I)^- \cdot (\xi', I)^+ \cdot \{\mathfrak{Fax}_{\xi' i \vdash \xi i}, i \in I\}, I \in \mathcal{P}_f(\omega)\}$.

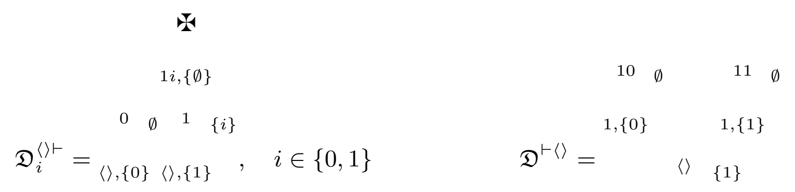

✠
$1i,\{\emptyset\}$
0 $\emptyset$ 1 $\{i\}$
$\mathfrak{D}_i^{\langle\rangle\vdash} =$
$\langle\rangle,\{0\}$ $\langle\rangle,\{1\}$
$, \quad i \in \{0,1\}$
10 $\emptyset$ 11 $\emptyset$
$1,\{0\}$ $1,\{1\}$
$\mathfrak{D}^{\vdash\langle\rangle} =$
$\langle\rangle$ $\{1\}$

Fig. 5. Designs corresponding to MALL✠ proofs from section 2.3

Normalization and Interaction. Interaction is built with cut-nets normalization which reflects linear logic cut-normalization. Designs are cut-free: a cut is the coincidence of a locus with opposite polarity in the base of two designs.

A ***cut-net*** $\mathfrak{R} = (\mathfrak{D}_i)_{i\in I}$ is a non-empty finite set of designs on bases $(\beta_i)_{i\in I}$ such that *(i)* the loci in $(\beta_i)_{i\in I}$ are either equal or disjoint; *(ii)* a locus ξ appears in at most two bases (then it occurs with different polarities and is called a ***cut*** in $\mathfrak{R}$) and *(iii)* the cuts define a binary relation over the designs which shall be connected and acyclic. The ***base*** of $\mathfrak{R}$ is the set of polarized loci of the $(\beta_i)_{i\in I}$ which **are not** cuts. A net with empty base is ***closed***. An action in $\mathfrak{R}$ is ***visible*** if it is ✠ or if its focus is not sublocus of a cut, otherwise it is ***hidden***. In any cut-net, there is a ***main design***: the only positive design of the net if such a design exists or the only negative design of base $\xi \vdash \Lambda$ such that ξ is not a cut.

Whereas in slices all actions are distinct, a design $\mathfrak{D}$ may contain several copies of the same action. To describe an action occurrence, we need additional information on the *position* of the action in the design: the branch leading to the action, called the ***chronicle*** for κ and written $Ch_{\mathfrak{D}}(\kappa)$. Views will allow to find the chronicle for an action provided we know a certain path in the design. The ***positive and negative views*** for a sequence of neutral actions are[2]: (i) $\ulcorner\epsilon\urcorner^{+} = \ulcorner\epsilon\urcorner^{-} = \epsilon$; (ii) $\ulcorner s\cdot(\xi,I)\urcorner^{+} = \ulcorner s\urcorner^{-}\cdot(\xi,I)^{+}$; (iii) $\ulcorner s\cdot(\xi,I)\urcorner^{-} = \ulcorner t\urcorner^{+}\cdot(\xi,I)^{-}$ if $s = tu$ and u is the longest suffix of s such that no action in u creates ξ.

A path p in a slice $\mathfrak{S}$ (*ie.* a sequence of actions in $\mathfrak{S}$) is a ***visit path*** if it is: (i) of alternating polarities; (ii) made only of proper actions; (iii) downward closed (if $p'\cdot\kappa$ is a prefix of p, all actions below κ in $\mathfrak{S}$ are in p'). The polarity of p is its last action polarity. Given a path p, we write p_ν for the sequence of neutral actions canonically associated with p. Notice that a visit path cannot necessarily be realized by interaction. The following is an essential property of views: If p is a visit path in a slice $\mathfrak{S}$ with last action κ of polarity ϵ then $\ulcorner p_\nu\urcorner^{\epsilon}$ is the chronicle for κ in $\mathfrak{S}$, $Ch_{\mathfrak{S}}(\kappa)$.

The ***Loci Abstract Machine*** (LAM [5]), is an abstract machine that computes the interaction of a cut-net $\mathfrak{R}$, described as tokens[3] travelling on the cut-net. Let $\mathfrak{R}$ be a cut-net on a base β. Let $T_{\mathfrak{R}}$ be the set of all positions reached by the tokens during normalization.

[2] Notice that in case (iii), either t is empty or its last neutral action is (σ, J) with $\xi = \sigma j$ for some $j \in J$. Moreover, one can trivially extend positive views to sequences ending with the ✠: $\ulcorner s\cdot ✠\urcorner^{+} = \ulcorner s\urcorner^{-}\cdot ✠$.

[3] A ***token*** is a pair (s,κ) of a neutral sequence of actions s and an action κ, where s records the path followed by the token from the initial state up to κ.

- **Initialization.** If κ is at the root of the main design in $\mathfrak{R}$ the $(\epsilon, \kappa) \in T_{\mathfrak{R}}$;
- **Transitions.** Let $(s, \kappa) \in T_{\mathfrak{R}}$. There are 3 cases:
  - **Visible.** If κ is a visible action of polarity ϵ, then for each κ' such that $\ulcorner s\kappa \urcorner^{\epsilon} \kappa' \in \mathfrak{R}$, $(s\kappa, \kappa') \in T_{\mathfrak{R}}$ (notice $\ulcorner s\kappa \urcorner^{\epsilon}$ is the chronicle leading to κ);
  - **Up.** If κ is a hidden negative action, then let κ' be the successor of the extremal action of $\ulcorner s\kappa \urcorner^{-}$, we have $(s\kappa, \kappa') \in T_{\mathfrak{R}}$;
  - **Jump.** If κ is a hidden positive action, then let $\kappa' = \kappa^{-}$. If $\ulcorner s\kappa \urcorner^{-} \in \mathfrak{R}$ then $(s, \kappa') \in T_{\mathfrak{R}}$. Otherwise normalization fails.

Let $\mathfrak{R}$ be a cut-net and let $T_{\mathfrak{R}}$ be the positions reached by the tokens during normalization. A ***normalization path*** is the sequence of actions which are visited during the normalization of $\mathfrak{R}$: $Path(\mathfrak{R})$ is defined to be the set $\{s \cdot \kappa_{\nu} / (s, \kappa) \in T_{\mathfrak{R}}$ such that s is maximal$\}$. We also define $hide(p)$ to be the sequence obtained by removing all hidden actions in p and ***Hide($\mathfrak{R}$)*** to be the set $\{hide(p), p \in Path(\mathfrak{R})\}$. The ***normal form*** of a cut-net $\mathfrak{R}$ is the design defined to be[4]: $[\![\mathfrak{R}]\!] = \{\chi / \chi$ is a prefix of p^{+} with $p \in \text{Hide}(\mathfrak{R})\}$.

If $\mathfrak{R}$ is a closed cut-net, we call ***dispute*** the normalization path of $\mathfrak{R}$. If the net is $\{\mathfrak{D}, \mathfrak{E}\}$, we write $[\mathfrak{D} \rightleftharpoons \mathfrak{E}]$ for the dispute.

Orthogonality and Behaviours. Orthogonality describes those normalizations that were successful: designs $\mathfrak{D}, \mathfrak{E}$ are ***orthogonal*** if they form a cut-net and $[\![\mathfrak{D}, \mathfrak{E}]\!] = \maltese$, written $\mathfrak{D} \perp \mathfrak{E}$. If $\mathfrak{D}$ has base $\xi_1^{\epsilon_1}, \ldots, \xi_n^{\epsilon_n}$ and $(\mathfrak{E}_{\xi_i})_{1 \leq i \leq n}$ are designs on atomic base $\xi_i^{-\epsilon_i}$, $(\mathfrak{D}, \mathfrak{E}_{\xi_1}, \ldots, \mathfrak{E}_{\xi_n})$ forms a closed cut-net; if $[\![\mathfrak{D}, \mathfrak{E}_{\xi_1}, \ldots, \mathfrak{E}_{\xi_n}]\!] = \maltese$ we write $\mathfrak{D} \perp (\mathfrak{E}_{\xi_i})_{1 \leq i \leq n}$. The ***orthogonal*** of an atomic design $\mathfrak{D}$ is: $\mathfrak{D}^{\perp} = \{\mathfrak{E} / \mathfrak{D} \perp \mathfrak{E}\}$. A set of designs on the same atomic base, written $\mathbf{E}$, is called an ***ethic*** and its orthogonal is $\mathbf{E}^{\perp} = \{\mathfrak{D} / \forall \mathfrak{E} \in \mathbf{E}, \mathfrak{D} \perp \mathfrak{E}\}$. $\prec$ is a relation on designs defined by: $\mathfrak{D} \prec \mathfrak{D}'$ if, and only if, $\mathfrak{D}^{\perp} \subseteq \mathfrak{D}'^{\perp}$. $\prec$ is actually a ***partial order*** (Separation theorem, [10]).

A ***behaviour*** $\mathbf{G}$ is an ethic which is equal to its bi-orthogonal: $\mathbf{G} = \mathbf{G}^{\perp\perp}$. It is immediate that ***the orthogonal of an ethic*** is a behaviour. Let $\mathfrak{D}$ be a design, the ***principal behaviour*** of $\mathfrak{D}$ is $\{\mathfrak{D}\}^{\perp\perp}$: it is the smallest behaviour containing $\mathfrak{D}$. If $\mathfrak{E} \in \mathbf{G}$, there exists a smallest design $\mathfrak{D} \subset \mathfrak{E}$ such that $\mathfrak{D} \in \mathbf{G}$. It is the ***incarnation*** of $\mathfrak{E}$ in $\mathbf{G}$ written $|\mathfrak{E}|_{\mathbf{G}}$. A design is said to be ***material*** in a behaviour when it is equal to its own incarnation.

4 Interactive proof search algorithm.

In this section, we present an algorithm for Interactive Proof Search. We give a machine inspired by Faggian's LAM, the Searching LAM (SLAM) which allows us to build interactively designs by orthogonality to a set of tests.

4.1 The Idea of the algorithm

Before going to the formal definitions of IPS procedure, we sketch how IPS works on a simple example in order to show the main structure of the search: consider

[4] We use notation $s^{+/-}$ to mean: $\epsilon^{+} = \epsilon^{-} = \epsilon$; $(s \cdot \kappa)^{+} = s^{-} \cdot \kappa^{+}$; $(s \cdot \kappa)^{-} = s^{+} \cdot \kappa^{-}$.

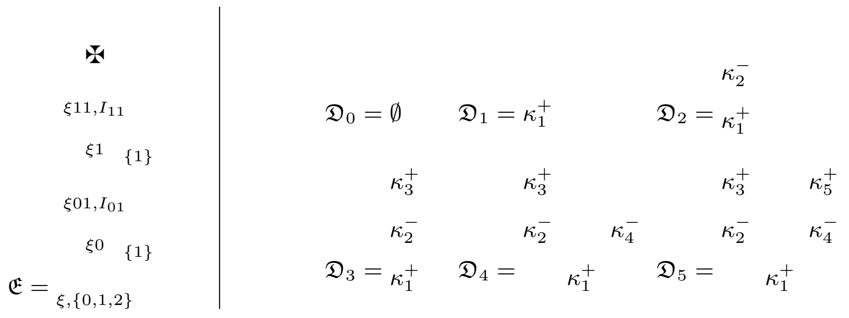

$\maltese$
$\xi 11, I_{11}$
$\xi 1 \quad \{1\}$
$\xi 01, I_{01}$
$\xi 0 \quad \{1\}$
$\mathfrak{E} = \; \xi, \{0,1,2\}$
$\mathfrak{D}_0 = \emptyset$
$\mathfrak{D}_1 = \kappa_1^+$
$\mathfrak{D}_2 = \kappa_1^+ \; \kappa_2^-$
$\mathfrak{D}_3 = \kappa_1^+ \; \kappa_2^- \; \kappa_3^+$
$\mathfrak{D}_4 = \kappa_1^+ \; \kappa_2^- \; \kappa_3^+ \; \kappa_4^-$
$\mathfrak{D}_5 = \kappa_1^+ \; \kappa_2^- \; \kappa_3^+ \; \kappa_4^- \; \kappa_5^+$

Fig. 6. Interactive search for $\mathfrak{D}$.

the Interactive Proof Search driven by one very simple design. Let us proceed with an IPS with environment $\{\mathfrak{E}\}$ (see figure 6) 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, in $\text{Init}_{\mathfrak{E}} = \{(\xi, \{0,1,2\})^-\}$, one of which shall be followed during a normalization process. 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 $\mathfrak{D}$: $Path_1 = \langle(\xi, \{0,1,2\})\rangle$;
2. Design $\mathfrak{D}$ 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 $\mathfrak{E}$: $\kappa_2^+ = (\xi 0, \{1\})^+$ and $Path_2 = \langle \kappa_{1\nu}, \kappa_{2\nu} \rangle$;
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 $\kappa_4^+ = (\xi 1, \{1\})^+$ and thus, $Path_3$ is extended with $\kappa_{4\nu}$ and $\mathfrak{D}_4$ with κ_4^- which is put right above its justifyer. $Path_4 = \langle \kappa_{1\nu}, \kappa_{2\nu}, \kappa_{3\nu}, \kappa_{4\nu} \rangle$ and the branch leading to κ_4^- in $\mathfrak{D}$ is given by: $\ulcorner Path_4 \urcorner^- = \kappa_1^+, \kappa_4^-$;
5. In $\mathfrak{E}$, $Succ(\kappa_4^+) = \{(\xi 11, I_{11})^-\}$. $Path_5 = \langle \kappa_{1\nu}, \ldots, \kappa_{5\nu} \rangle$ 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 $[\mathfrak{D} \rightleftharpoons \mathfrak{E}] = \langle \kappa_{1\nu}, \ldots, \kappa_{5\nu}, \maltese \rangle$.
$\rightarrow$ After IPS, we built a design $\mathfrak{D}$ on $\vdash \xi$ such that $[\![\mathfrak{D}, \mathfrak{E}]\!] = \maltese$ with $\maltese$ used by $\mathfrak{E}$.

This example was intended to illustrate the basic mechanisms that we shall encounter while doing IPS. We now introduce formally the IPS process.

4.2 SLAM-1

We first introduce an abstract machine for interactive search of designs in the restricted case of 4.1, when the test-environment is made of only one design, $\mathfrak{E}$.

Definition 1 (States of SLAM-1). States *of SLAM-1 are triples* $\langle p \bullet \mathbf{E} \mid \mathfrak{D} \rangle$ *of a sequence of neutral actions* p, *a set of designs* $\mathbf{E}$ *(the current test-environment) containing at most one positive design and a set of chronicles* $\mathfrak{D}$ *(the design under construction, for which* p *is a visit path). An* **initial state** *is of the form* $\langle \epsilon \bullet \{\mathfrak{E}\} \mid \emptyset \rangle$. *A* **final state** *of the form* $\langle p \bullet \emptyset \mid \mathfrak{D} \rangle$.

We saw in 4.1 that there may be choices to make during IPS when several negative actions are available. In order to define a deterministic search machine (for instance a depth-first search strategy with left most choice), we introduce ***selection functions*** which shall parametrize the abstract machine. Those selection functions take as input a state $\mathcal{S}$ of the machine together with a set of negative actions *Init* and return a subset of *Init* (which is not empy unless *Init* is itself empty). A selection function *Select* is said ***deterministic*** when $Select(\mathcal{S}, Init)$ is a singleton except when $Init = \emptyset$. When taking as selection function the second projection one has the fully non-deterministic machine. Moreover, the set *Init* of initial negative actions is obtained as follows: given a family of negative designs $\mathbf{E} = (\mathfrak{D}_i = \{{\kappa_j^i}^- \cdot \mathfrak{D}_j^i, j \in J_i\})_{i \in I}$, one sets $Init(\mathbf{E})$ to $\{{\kappa_j^i}^-, j \in J_i, i \in I\}$. Finally, given a sequence of neutral actions p, a set of negative actions I and a set of chronicles $\mathfrak{D}$, one sets $\Im(p, I, \mathfrak{D})$ to be $\{\kappa \in Init \,/\, \kappa$ is justified by an action in $\ulcorner p \urcorner^-$.$\}$

Definition 2 (SLAM-1). *Let Select be a selection function and $\mathfrak{C}$ an atomic design. SLAM-1 is defined as follows:*
Initial State: $\langle \epsilon \;\bullet\; \{\mathfrak{C}\} \;\mid\; \emptyset \rangle$
Transitions: $\langle p \;\bullet\; \mathbf{E} \;\mid\; \mathfrak{D} \rangle \longrightarrow \langle p' \;\bullet\; \mathbf{E}' \;\mid\; \mathfrak{D}' \rangle$
• *If* $\mathbf{E}$ *contains a positive design* $\mathfrak{D}^+ = \kappa^+ \cdot \{\mathfrak{D}'_j, j \in J\}$. *If* $\kappa^+ = \maltese$, *then final state* $\langle p \cdot \maltese \;\bullet\; \emptyset \;\mid\; \mathfrak{D} \rangle$ *is reached. Otherwise,* κ^+ *is proper and we set* **(i)** p' *to* $p \cdot \kappa$, **(ii)** $\mathbf{E}'$ *to* $\mathbf{E} \setminus \{\mathfrak{D}^+\} \cup \{\mathfrak{D}'_j, j \in J\}$ *and* **(iii)** $\mathfrak{D}'$ *to* $\mathfrak{D} \cup \{\ulcorner p \cdot \kappa \urcorner^-\}$.
• *Otherwise* $\mathbf{E} = (\mathfrak{D}_i = \{{\kappa_j^i}^- \cdot \mathfrak{D}_j^i, j \in J_i\})_{i \in I}$. *Let* $Ini = \Im(p, Init(\mathbf{E}), \mathfrak{D})$. *If* $Select(\langle p \;\bullet\; \mathbf{E} \;\mid\; \mathfrak{D} \rangle, Ini) \neq \emptyset$, *one chooses some* $\kappa \in Select(\langle p \;\bullet\; \mathbf{E} \;\mid\; \mathfrak{D} \rangle, Ini)$, *and considers* $\mathfrak{D}_i$ $(i \in I)$ *the negative design of which* κ *is an initial action, and* $\mathfrak{D}_j^i$ $(j \in J_i)$ *the positive design immediately above* κ *in* $\mathfrak{D}_i$ *(ie.* $\kappa = \kappa_j^i$*). We set* **(i)** p' *to* $p \cdot \kappa$, **(ii)** $\mathbf{E}'$ *to* $\mathbf{E} \setminus \{\mathfrak{D}_i\} \cup \{\mathfrak{D}_j^i\}$ *and* **(iii)** $\mathfrak{D}'$ *to* $\mathfrak{D} \cup \{\ulcorner p \cdot \kappa \urcorner^+\}$.
If $Ini = \emptyset$, *final state* $\langle p \;\bullet\; \emptyset \;\mid\; \mathfrak{D} \cup \{\ulcorner p \cdot \maltese \urcorner^+\} \rangle$ *is reached.*

A **result** *of the machine consists in the third component of a final state.*

4.3 Properties of SLAM-1.

We consider here an IPS with test environment $\mathbf{E}$. The sets of chronicles built by interaction during an evaluation of the machine satisfy the coherence conditions for designs in section 3:

Proposition 1. *The results of SLAM-1 executions are slices.*

Proposition 2. *If* $\mathfrak{D}$ *is a result of SLAM-1,then* $\mathfrak{D} \in \mathbf{E}^\perp$. $\mathfrak{D}$ *is material in* $\mathbf{E}^\perp$.

Definition 3 ($\mathfrak{D}_i^\maltese$). *If* $(\langle p_i \;\bullet\; \mathbf{E}_i \;\mid\; \mathfrak{D}_i \rangle)_{0 \leq i \leq n}$ *is a run of SLAM-1, then for* $0 < i < n$ *one may build a design* $\mathfrak{D}_i^\maltese$ *by adding a daimon if the last action visited is negative or replacing the last visited rule with a daimon if it is positive.*

$\mathfrak{D}_i^\maltese$ are more and more precise:

Proposition 3. *For* $0 < i < n$, $\mathfrak{D}_i^\maltese$ *is a slice, it is material in* $\mathbf{E}^\perp$ *and for* $0 < i \leq j < n$, *one has:* $\{\mathfrak{D}_i^\maltese\}^{\perp\perp} \subseteq \{\mathfrak{D}_j^\maltese\}^{\perp\perp} \subseteq \mathbf{E}^\perp$.

The IPS procedure described by SLAM-1 only produces slices as asserted by proposition 2. As a result, this setting is fairly restricted and moreover the test-environments considered are very constrained and as a conclusion do not allow much flexibility. For instance it does not allow to build proofs with additive branching and it does not allow to treat backtracking. For instance one would like to work with more general test environments such as the ones considered in section 2.3 when using $\mathfrak{D}_0$ and $\mathfrak{D}_1$ to build the two premisses of a with rule or with $\mathfrak{D}_2$ or $\mathfrak{D}_3$ to avoid visiting some branch. We shall now remove this restriction resulting in a more complex machine that we define in what follows.

4.4 SLAM-n

SLAM-n will consider states storing several tests and the interactive construction will depend on several designs and not only one: as a consequence there shall be a mechanism to synchronize the tests that contribute to the same branch. Moreover distinct parts of the test environment may contribute to different additive branches of the design; it is thus necessary to locate the interactions.

Definition 4 (SLAM-n States). *States have the form $\langle (p_i \bullet (\mathbf{E}^{ij}_E)_{j\in J_i})_{i\in I} \mid \mathfrak{D}\rangle$ where $(p_i)_{i\in I}$ are pairwise incomparable sequences of neutral actions, $(\mathbf{E}^{ij})_{i\in I, j\in J_i}$ are sets of designs such that for $i \in I$ either all $\mathbf{E}^{ij}$ $(j \in J_i)$ contain one positive design or they contain only negative designs, and $\mathfrak{D}$ is a set of chronicles.*

Definition 5 (SLAM-n). *Let Select be a selection function and $(\mathfrak{E}_j)_{j\in J}$ be designs on some atomic base $\xi \vdash$, SLAM-n is defined as follows:*
Initial State: $\langle (\epsilon \ \bullet \ (\{\mathfrak{E}_j\})_{j\in J}) \ \mid \ \emptyset\rangle$
Transitions: $\langle (p_i \ \bullet \ (\mathbf{E}^{ij}_E)_{j\in J_i})_{i\in I} \ \mid \ \mathfrak{D}\rangle \longrightarrow \langle (p'_i \ \bullet \ (\mathbf{E}'^{ij}_E)_{j\in J'_i})_{i\in I'} \ \mid \ \mathfrak{D}'\rangle$

One chooses some $i_0 \in I$ such that the last action of p_{i_0} is not ✠.

• *If each $\mathbf{E}^{i_0 j}_E$ contains a positive design $\mathfrak{D}^+_{i_0 j} = \kappa^+_{i_0 j} \cdot \{\mathfrak{D}'_k, k \in K_{i_0 j}\}$ then let $J'_{i_0} = \{j \in J_{i_0}, \kappa^+_{i_0 j}$ is a proper action$\}$. One partitions J'_{i_0} in maximal non-empty subsets $(J^l_{i_0})_{l\in L}$ such that if $\forall l \in L, \forall m, n \in J^l_{i_0}, \kappa^+_{i_0 m} = \kappa^+_{i_0 n}$ (and thus if $k \neq l, m \in J^k_{i_0}, n \in J^l_{i_0}$ then $\kappa^+_{i_0 m} \neq \kappa^+_{i_0 n}$). Let $\kappa'_{i_0 l}$ be the action canonically associated with $J^l_{i_0}$.*
(i) If $\exists l \in L, \ulcorner p_{i_0} \cdot \kappa'_{i_0 l} \urcorner^- \in \mathfrak{D}$, SLAM-n is **stuck**, *(ii) otherwise:*

- $I' = I \setminus \{i_0\} \cup L$, $J'_i = J_i$ *if* $i \in I \setminus \{i_0\}$ *and* $J'_l = J_{i_0 l}, l \in L$
- $p'_i = p_i$ *if* $i \in I \setminus \{i_0\}$, $p'_l = p_{i_0} \cdot \kappa'_{i_0 l}, l \in L$
- $\mathbf{E}'^{ij} = \mathbf{E}^{ij}$ *if* $i \in I \setminus \{i_0\}, j \in J_i$
- $\mathbf{E}'^{lj} = \mathbf{E}^{i_0 j} \setminus \{\mathfrak{D}^+_{i_0 j}\} \cup \{\mathfrak{D}'_k, k \in K_{i_0 j}\}$ *for* $l \in L, j \in J_{i_0 l}$
- $\mathfrak{D}' = \mathfrak{D} \cup \{\ulcorner p_{i_0} \cdot \kappa'_{i_0 l} \urcorner^-, l \in L\}$

• *If for any $j \in J_{i_0}$, $\mathbf{E}^{i_0 j}$ contains only negative designs: $(\mathfrak{D}_{jl} = \{\kappa^{jl^-}_k \cdot \mathfrak{D}^{jl}_k, k \in K_{jl}\})_{l\in L_j} = \mathbf{E}^{i_0 j}$. Let $Init_j = Init(\mathbf{E}^{i_0 j})$ and $Init = \Im(p_{i_0}, \cap_{j\in J_{i_0}} Init_j, \mathfrak{D})$.*

(i) If $Init \neq \emptyset$, let κ be some action in $Select(\langle (p_i \bullet (\mathbf{E}^{ij})_{j\in J_i})_{i\in I} \mid \mathfrak{D}\rangle, Init)$, and for every $j \in J_{i_0}$, one considers the negative design $\mathfrak{D}_{jl}$ of which κ is an initial action in $\mathbf{E}^{i_0 j}$ and $k_0 \in K_{jl}$ such that $\mathfrak{D}^{jl}_{k_0}$ is the positive design immediately above κ in $\mathfrak{D}_{jl}$. Then:

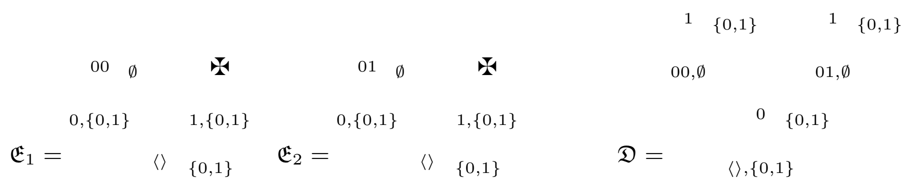


Fig. 7. An execution of SLAM-n from $\mathfrak{E}_1$ and $\mathfrak{E}_2$ not resulting in a design.

- $I' = I$ *and* $J'_i = J_i, \forall i \in I'$
- $p'_i = p_i$ *for* $i \neq i_0$ *and* $p'_{i_0} = p_{i_0} \cdot \kappa$
- $\mathbf{E'}^{ij} = \mathbf{E}^{ij}$ *for* $i \in I', j \in J'_i, i \neq i_0$
- $\mathbf{E'}^{i_0 j} = \mathbf{E}^{i_0 j} \setminus \{\mathfrak{D}_{jl}\} \cup \{\mathfrak{D}^{jl}_{k_0}\}$, $\forall j \in J_{i_0}$
- $\mathfrak{D}' = \mathfrak{D} \cup \{\ulcorner p_{i_0} \cdot \kappa \urcorner^+\}$

(ii) If $Init = \emptyset$*, then we add chronicle* $\ulcorner p_{i_0} \cdot \maltese \urcorner^+$ *to the design under construction moving to the state:* $\langle (p_i \bullet (\mathbf{E}^{ij}_E)_{j \in J_i})_{i \in I \setminus \{i_0\}} \mid \mathfrak{D} \cup \{\ulcorner p_{i_0} \cdot \maltese \urcorner^+\} \rangle$.

SLAM-n contains a case where the machine is stuck. Moreover, when the tests are chosen totally arbitrarily, the set of chronicles which is produced by SLAM-n may not be a design as examplified in figure 7: $\mathfrak{D}$ which results from IPS with $\mathfrak{E}_1$ and $\mathfrak{E}_2$, violates the ***additive sharing*** condition. In order to fix this problem, we slightly modify the definition of $\Im(p, I, \mathfrak{D})$ as follows: $\Im(p, I, \mathfrak{D}) = \{(\sigma, I)^- / (\sigma, I)^-$ is justified in $\ulcorner p \urcorner^-$ and if $\exists \chi \cdot (\sigma, L)^+ \in \mathfrak{D}$ then the first difference between χ and $\ulcorner p \urcorner^-$ involves negative actions on the same focus$\}$.

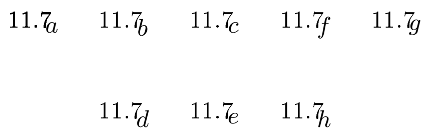
Proposition 4. *If* $\langle (\epsilon \bullet \{(\mathfrak{E}_i)_{i \in I}\}) \mid \emptyset \rangle$ *is an initial state, then an execution of SLAM-n that is never stuck (case 1.(i) is never encountered) results in a set of chronicles interactively built which is a design.*

Proposition 5. *A final state for an execution which is never* **stuck** *(case 1.(i) of SLAM-n) is of the form* $\langle (p_i \cdot \maltese \bullet \emptyset)_{i \in I} \mid \mathfrak{D} \rangle$.

4.5 Backtracking

In the present section we briefly explain how backtracking can be dealt with using generalized environments. We shall consider a final state $\mathcal{S}' = \langle (p_i \cdot \maltese \bullet \emptyset)_{i \in I} \mid \mathfrak{D} \rangle$ reached from an initial state $\mathcal{S} = \langle (\epsilon \bullet (\{\mathfrak{E}_j\})_{j \in J}) \mid \emptyset \rangle$. If $I \neq \emptyset$, then $\mathfrak{D}$ is a failure (it contains $\maltese$ at $\ulcorner p_i \cdot \maltese \urcorner^+$). One shall use those paths $p_i, i \in I$ to enrich the test environment with new designs.

Definition 6 ($\mathfrak{Test}(p)$). (i) $\mathfrak{Test}(\epsilon) = \emptyset$
(ii) $\mathfrak{Test}(\kappa) = \{\kappa^+\}$*;*
(iii) $\mathfrak{Test}(s \cdot \kappa \cdot \kappa') = \{\ulcorner s \cdot \kappa \cdot \kappa' \urcorner^+, \ulcorner s \cdot \kappa \urcorner^-\} \cup \mathfrak{Test}(s)$.



```
adj(a, b).     adj(b, c).
adj(b, d).     adj(c, e).
adj(c, f).     adj(f, g).
adj(h, g).
 p(X,Y) :- adj(X,Z), adj(Z,Y).
```

Fig. 8. Graph

Proposition 6. *$\mathfrak{Test}(p_i), i \in I$ is a slice. Moreover, $\mathfrak{Test}(p_i)$ is the smallest design (as sets of chronicles) realizing interaction $p_i \cdot \maltese$ with the final design $\mathfrak{D}$.*

In order to model the backtrack instruction, one shall use a variant of $\mathfrak{Test}(\_)$. Indeed, $\mathfrak{Test}(\_)$ contains both too many and too few chronicles to be used to backtrack: a backtrack design should not allow to interact along p_i up to reaching the daimon and it should be able to interact with any other design.

Definition 7 ($\mathfrak{Backtrack}(p)$). *$\mathfrak{Backtrack}(p)$ is the smallest design such that:*

- *$\mathfrak{Backtrack}(p)$ contains all positive chronicles of $\mathfrak{Test}(p)$ except $\ulcorner p \urcorner^{+}$;*
- *if $\chi \in \mathfrak{Backtrack}(p)$ has last action $(\xi, I)^{+}$, then for any $i \in I$ and $J \in \mathcal{P}_f(\omega)$ such that $\chi \cdot (\xi i, J)^{-} \notin \mathfrak{Test}(p)$, one has $\chi \cdot (\xi i, J)^{-} \cdot \maltese \in \mathfrak{Backtrack}(p)$.*

Remark 1. In definition 7, $\mathfrak{Backtrack}(p)$ is drastically infinite because of all the chronicles $\chi \cdot (\xi i, J)^{-} \cdot \maltese$ that are added. However, by collecting information in the course of a search (at step 1. of SLAM-n, when considering the set of initial positive actions in the environment), one may retain only the actions that may be encountered and build a finite $\mathfrak{Backtrack}(p)$ if original test-environments are made of finitely branching designs.

Proposition 7. *$\langle (\epsilon \bullet (\{\mathfrak{C}_j\})_{j \in J} \cup (\{\mathfrak{Backtrack}(p_i)\})_{i \in I}) \mid \emptyset \rangle$ is an initial state that will not compute disputes $(p_i)_{i \in I}$ anymore.*

4.6 A concrete example.

Let $\mathcal{G}$ be the graph represented in figure 8. We want to implement the search for paths of length 2 in this graph using interactive proof search. This corresponds to the predicates shown in figure 8. For instance, $p(c, g)$ could be represented as the MALL formula:

$$\bigoplus_{x \in \{a, \dots, h\}} (adj(c, x) \ \& \ adj(x, g))$$

The graph and the path relation p shall be represented as counter-designs[5], as tests that will guide an interactive search for a design $\mathfrak{D}$. The counter-design

[5] Here is how $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are built: Let us choose a location p in which one shall locate the designs of the environment (on base $p \vdash$) and the design to construct by ISP (on base $\vdash p$). Let us suppose $a, b, \dots, g, h$ are integer codes representing nodes of the graph in the obvious way (one can choose arbitrary distinct integers). pe will thus represent the formula $(adj(c, e) \ \& \ adj(e, g))$ and $pe1$ and $pe2$ will respectively represent $(adj(c, e)$ and $(adj(e, g)$. $\mathfrak{C}_1$ represents the arcs having their origin in c and $\mathfrak{C}_2$ represents the arcs having g as goal.

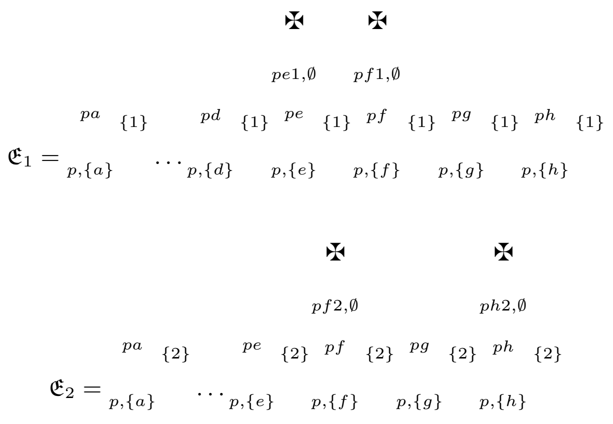

$\mathfrak{E}_1 =$
$pe1,\emptyset$ $pf1,\emptyset$
pa $\{1\}$ pd $\{1\}$ pe $\{1\}$ pf $\{1\}$ pg $\{1\}$ ph $\{1\}$
$p,\{a\}$ $\cdots$ $p,\{d\}$ $p,\{e\}$ $p,\{f\}$ $p,\{g\}$ $p,\{h\}$
$\mathfrak{E}_2 =$
$pf2,\emptyset$ $ph2,\emptyset$
pa $\{2\}$ pe $\{2\}$ pf $\{2\}$ pg $\{2\}$ ph $\{2\}$
$p,\{a\}$ $\cdots$ $p,\{e\}$ $p,\{f\}$ $p,\{g\}$ $p,\{h\}$

Fig. 9. Designs $\mathfrak{E}_1$ and $\mathfrak{E}_2$.

environnement could be made of two designs $\mathfrak{E}_1$ and $\mathfrak{E}_2$ of figure 9: We are in a case of a 2-environment.

There are 8 choices for the first action in constructing design $\mathfrak{D}$, but this leads then to the designs of figure 10 (8 possible computations) depending on the choice of the first action (only one being a success).

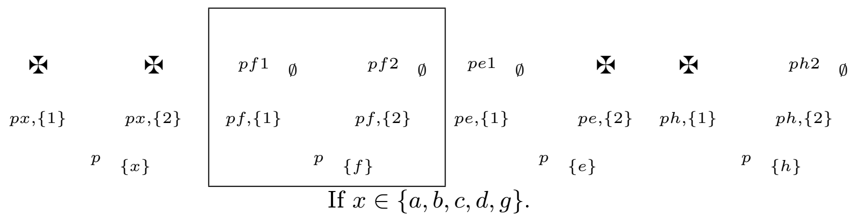


Fig. 10. Results of an IPS with $\mathfrak{E}_1$ and $\mathfrak{E}_2$.

5 Conclusion.

The aim of this paper was to introduce a novel approach to proof search as computation where the search is not guided by a sequent as in standard proof search but is contrained by an environment of tests.

Contributions. The contributions of the paper are as follows:

- we provided a motivation for an interactive approach for proof-search;
- we examplified a "concrete" approach to interactive proof-search on a sequent calculus derived from MALL sequent calculus;

– we introduced ludics concepts by emphasizing concepts that are the most relevant for the logic programming community;
– we introduced IPS thanks to the SLAM, an abstract machine inspired by Faggian's LAM [5] and provided some analysis of its search behaviour;
– we explained how to treat backtracking in interactive proof search Backtracking becomes a logical search instruction.

Related works. In [5], Faggian introduced the LAM and studied some properties of its execution. Those results will be helpful to develop IPS. Pym and Ritter [24] give a semantics for proof search which is related with game semantics. They have a treatment of backtracking using relations between intuitionistic and classical proofs. We shall investigate the connections with our work.

Future Works. Lots of things are still to be done in order to have a computation model based on interactive proof search:

– we shall develop the treatment of the `cut` and other pruning operations in the same way we did for backtracking;
– this work is very foundational at the moment. In particular we shall work towards extending the expressiveness of interactive proof-search, mostly in two directions: first-order and recursive definitions. Fleury and Quatrini [7] proposed a theory of first-order in Ludics. On the other hand, Ludics is based on MALL and except for very recent proposals by Faggian and Basaldella there are no exponentials in Ludics that would allow using a formula (here, an action) several times. To enlarge the setting one may be interesting in considering at least recursive definitions or fixpoints.

Aknowledgments. The author thanks Dale Miller for his advice and directions, Jean-Yves Girard for his comments on this project as well as Claudia Faggian for helpful discussions regarding the material in this paper.

References

1. Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *J. of Symbolic Logic*, 59(2):543–574, 1994.
2. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
3. Roberto Di Cosmo, Jean-Vincent Loddo, and Stephane Nicolet. A game semantics foundation for logic programming. In *PLILP/ALP*, pages 355–373, 1998.
4. Olivier Delande and Dale Miller. A neutral approach to proof and refutation in MALL. *23th Symp. on Logic in Computer Science.* IEEE Comp. Soc. Press, 2008.
5. Claudia Faggian. Travelling on designs. In *CSL'02*, pages 427–441, 2002.
6. C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL'02*, 2002.
7. M.-R. Fleury and M. Quatrini. First order in ludics. *MSCS*, 14(2):189–213, 2004.
8. C. Galanaki, P. Rondogiannis, and W. W. Wadge. An infinite-game semantics for well-founded negation in logic programming. *APAL*, 151(2):70–88, 2008.
9. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

10. Jean-Yves Girard. Locus solum. *MSCS*, 11(3):301–506, June 2001.
11. Timothy Griffin. A formulae-as-types notion of control. In *POPL'90*, 1990.
12. William A. Howard. The formulae-as-type notion of construction, 1969. In J. P. Seldin and R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, New York, 1980.
13. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–408, 2000.
14. Olivier Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, 2002.
15. Jean-Vincent Loddo. *Généralisation des Jeux Combinatoires et Applications aux Langages Logiques.* Thèse de doctorat, Université Paris VII, 2002.
16. Jean-Vincent Loddo and Roberto Di Cosmo. Playing logic programs with the alpha-beta algorithm. In *LPAR*, pages 207–224, 2000.
17. Dale Miller. Sequent calculus and the specification of computation. In *Computational Logic*, volume 165 of *Nato ASI Series*, pages 399–444. Springer, 1999.
18. Dale Miller. Overview of linear logic programming. In T. Ehrhard, J.-Y. Girard, P. Ruet, and P. Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note*, pages 119–150. CUP, 2004.
19. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *APAL*, 51:125–157, 1991.
20. Dale Miller and Alexis Saurin. A game semantics for proof search: Preliminary results. In *MFPS05*, number 155 in ENTCS, pages 543–563, 2006.
21. Lee Naish. *Negation and Control in Prolog*, volume 238 of *LNCS*. Springer 1986.
22. Lee Naish. Pruning in logic programming. Technical Report 95/16, Department of Computer Science, University of Melbourne, Melbourne, Australia, june 1995.
23. Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of International Conference on Logic Programming and Automated Deduction*, volume 624 of *LNCS*, pages 190–201. Springer, 1992.
24. David Pym and Eike Ritter. *Reductive Logic and Proof-search: proof theory, semantics, and control*, volume 45. Oxford Logic Guides, Oxford, 2004.
25. David Pym and Eike Ritter. A games semantics for reductive logic and proof-search. In D. Ghica and G. McCusker, editors, *GaLoP 2005*, pages 107–123, 2005.
26. Maarten H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.