Mini-defense manuscript

Interactive Deep Reasoning

Pablo Donato

May 7, 2022

1 General context

Proof assistants are software systems allowing for the precise synthesis and verification of mathematical proofs. They are based on the idea that mathematical knowledge can be represented syntactically and unambiguously inside *proof formalisms*. These have been developed since the beginning of the 20th century, in the well-established field of mathematical logic known as *proof theory*.

State-of-the-art proof assistants such as Coq or Isabelle are usually based on very expressive logics, so that virtually any mathematical development can be expressed in them. While this generality hinders the potential for fully-automated theorem-proving, formal proofs still tend to be too detailed and verbose to be written by hand, much like assembly code. This demands for a higher-level, more interactive approach to computer-assisted proof authoring.

The problem is usually tackled through *tactic languages*, which offer to users a set of primitive text commands to manipulate the proof state, as well as basic combinators to build more complex tactics. For example, given (hypothetical) proofs `H1` of A and `H2` of $A \Rightarrow B$, one can type the command `apply H2 in H1` to get a proof of B.

2 Initial research problem

While basic logical reasoning is considered universal, if not intuitive, each proof assistant currently has its own syntax to perform it. This induces an unnecessary cognitive burden for newcomers, especially those unfamiliar with textual interfaces; but also for seasoned practicioners, who spend a non-negligible amount of time performing mundane tasks such as naming, destructuring and application of hypotheses, or selection of subterms. This calls for friendlier and more ergonomic interfaces, which fully exploit the capabilities of modern devices.

A first attempt in this direction was made in the 90's by the team of G. Kahn at Inria, where they coined the "Proof-by-Pointing" (hereafter PbP) paradigm [1].

The idea was to synthesize complex tactics from the simple act of *pointing* at parts of expressions, typically with a mouse cursor. More recently [2], K. Chaudhuri proposed a variation on this idea termed “subformula linking”, where instead of selecting expressions in isolation, one can *link* two of them together to make them interact. In both cases, the expressions considered were logical formulas, and the associated actions chains of inferences in first-order logic (hereafter FOL).

The purpose of this thesis is to explore to how extent these two proof paradigms, and potentially others, can be integrated together in a single framework, in order to provide a unified, extensible, and complete graphical user interface to general-purpose proof assistants. By complete, we mean that interaction with the software should be mainly driven by graphical actions — in this case direct manipulation of logical propositions with a pointing device — rather than textual commands.

To that effect, we have two main axis of investigation: one, more practical, consists in developing a prototype of graphical interface called Actema, which allows us to test out the feasability and usefulness of various graphical proof actions, as well as their integration together. The other, more foundational, tries to understand the proof-theoretical structure of the transformations performed by these actions, but also explores how old and new ideas in proof theory, programming language theory and automated theorem proving can inspire new interaction principles, or unify existing ones.

3 Logical reasoning through Drag-and-Drop

3.1 PbP-based DnD

At the beginning of my PhD, the first thing I did was to extend ideas explored during my master’s internship within the Typical team, where I was already under the supervision of Benjamin Werner and Pierre-Yves Strub. At the time, I had performed the following tasks:

1. Familiarizing myself with the usage and codebase of Actema, the prototype of GUI for interactive theorem proving developed by the team.

2. Designing and implementing a so-called *drag-and-drop* (herafter DnD) proof tactic in Actema.

An example of such a DnD action corresponds to the familiar `apply` tactic mentioned earlier: given two hypotheses $P(t)$ and $\forall x.P(x) \Rightarrow Q(x)$, the user can grab one of them and bring it to the other, in order to produce a new hypothesis $Q(t)$. This is an instance of *forward* reasoning. Another example involves the conclusion of the goal to be proved: given a hypothesis A and a conclusion $B \wedge (A \vee C) \wedge D$, bringing the two items together will have the effect of simplifying the conclusion into $B \wedge D$. This is an instance of *backward* reasoning.

While the first example could be handled by the tactic I developed during my internship, the second could not. This is because it was based on a variant of the PbP algorithm [1], whose principle is to systematically decompose formulas starting from their topmost connective by applying rules of sequent calculus. In this case it would have generated two subgoals $A \vdash B$ and $A \vdash D$, which correspond to the premisses of the following partial derivation:

$$\dfrac{A \vdash B \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A}}{A \vdash A \lor C}\,\lor R_1 \qquad A \vdash D}{A \vdash (A \lor C) \land D}\,\land R}{A \vdash B \land (A \lor C) \land D}\,\land R$$

3.2 SFL-based DnD

The solution was to use a different algorithm, based on the subformula linking (hereafter SFL) paradigm of [2]. At the end of my internship, I sketched a variant of SFL for intuitionistic propositional logic (hereafter IPL), since at the time it was only formulated for linear FOL, and Actema is based on intuitionistic FOL. In the first months of my PhD, I improved intuitionistic SFL and completed the extension to the first-order case. It consisted in first designing proper inference rules for the deep inference system which would underlie the DnD algorithm. In particular, the rules handling first-order quantifiers differ from the original SFL of [2], in that they are designed to work with a substitution which instantiates positive $\exists$ and negative $\forall$. This substitution is pre-computed by *unification* during the DnD, which is a novelty of our approach.

Then I implemented the SFL-based DnD tactic in Actema, which allowed me to reveal some bugs and make adjustments. I completed the design process with a proof of correctness of the algorithm on paper, which comprises both *logical soundness* and a property called *productivity*, ensuring termination with a result of the expected form. I did not however prove completeness as in [2]: since the tactic was integrated in an already complete system[1], this was not necessary. I still intend to prove it in the future, for foundational rather than practical purposes.

All this lead to the redaction of an article presenting the Actema system and the DnD tactic, which was not accepted for publication. We did have the opportunity though to present our work at two conferences ([3], [4]).

3.3 Rewriting equalities

In the following months, I extended the DnD paradigm by adding support for *rewriting* of equalities. The basic idea is that linking an equality hypothesis $t = u$ with

[1]Indeed, one could already prove goals in Actema through *click actions*, which implement rules from natural deduction and sequent calculus.

occurrences of t (resp. u) in the goal should rewrite them into u (resp. t). In our graphical setting, it is easy for the user to specify a selection of subterm occurrences to be rewritten, especially since selection of multiple entities through pointing is a pervasive interaction mechanism in modern interfaces.

I first implemented these rewrite DnD actions in Actema as a frontend to a standard rewrite tactic, completely separate from the SFL tactic underlying all other DnD actions. This required a clear conceptual separation in the interaction model between tactics, and so-called *linkages*, which correspond to the data of a selection of subterms in a given subgoal being linked by DnD. While a tactic specifies a function turning a goal into multiple subgoals, a linkage specifies arguments to this tactic that can be retrieved from the goal. This emphasizes another distinction between the "internal" arguments of a tactic, and the "external" ones which cannot be provided by linkages, and in our graphical setting would need an additional input mechanism.

Then at some point I realized that the semantics of the rewrite tactic could be accounted for in the SFL tactic, by adding two rules for equality in the underlying deep inference system. We noticed that this extension is not only simple and elegant, but that it generates non-trivial and interesting rewriting behaviors, because of the interaction with the semantics of purely logical SFL. In particular, two features that are found in current proof assistants can be implemented by SFL with equality: *rewriting modulo unification* and *conditional rewriting*, which correspond respectively to the combination of *quantifiers* and *implication* with equality. The benefit of the deep inference approach is two-fold: on a technical level, it is in the first-order case a wide generalization of state-of-the-art rewrite tactics, since it supports arbitrary combinations of logical connectives, and not only hypotheses of the form $\forall x_1, \ldots, \forall x_n. A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow A$. On a philosophical level, it gives a proof-theoretical explanation to the behavior of rewrite tactics, and suggests that other tactics might be expressed and related in this framework.

To experiment with this "deep rewriting" tactic, I added ad hoc support in Actema for Peano arithmetic, and in particular *induction* on natural numbers with a dedicated click action. In equational theories like PA, rewriting makes for an important part of proofs, and doing so with DnD actions seems to be both more intuitive and ergonomic than with textual commands. But we still need extensive testing, and possibly user studies to assess this on firm ground.

On the basis of these positive experiments, we rewrote the DnD tactic article in order to include the deep rewriting extension, but also to improve the overall presentation. The article was accepted for publication at the CPP conference [5].

4 Reasoning without formulas

4.1 The chemical metaphor

The Actema prototype offers multiple ways to the user to attack the proof of a theorem: DnD actions for SFL and rewriting are the main mechanism, but they only work in a goal comprising multiple items. Since it is customary in proof assistants to specify the goal to be proved as a single logical formula, one needs a way to decompose it into many items for further processing through DnD. This is precisely what the introduction rules for logical connectives in sequent calculus do, and following the PbP paradigm we map them to click actions[2].

So visually, a proof in Actema consists in breaking logical items into subitems positioned freely in space, and then bringing those items together to make them interact and produce a new item. This is quite evocative of a *chemical reaction* controlled by the user, where logical formulas are akin to molecules made of propositional atoms linked together by logical connectives[3]. Click actions are then a mean to "heat" molecules to the point of breaking these chemical bonds. The most canonical examples are the right-introduction rule for implication $\Rightarrow$ and the left-introduction rule for conjunction $\wedge$, which break respectively a conclusion/red item/positive ion into a hypothesis/blue item/negative ion and a new conclusion, and a hypothesis into two hypotheses. In fact, it is strongly conjectured that these are the only click actions needed to obtain a complete deductive system for IPL: breaking red implications allows for backward DnDs, and blue conjunctions for forward DnDs[4].

Rather than completeness, the issue here is *consistency* of the user interface: if the user is allowed to decompose red $\Rightarrow$ and blue $\wedge$, she will assume naturally that she can also decompose blue $\Rightarrow$ and red $\wedge$, as well as $\vee$ of any color. While red $\vee$ can be handled by increasing the depth of PbP by 1 in order to select the disjunct to be proved, other configurations correspond to rules of sequent calculus with multiple premisses. In Actema, this corresponds to creating a new subgoal for each premise, where subgoals are displayed one at a time in different *tabs*: this new interface mechanism breaks the chemical metaphor. The root cause lies in the way sequent calculus implements *context-scoping*: each subgoal will share the same initial context of hypotheses, but future hypotheses "buried" in the conclusions must be available only in their respective subgoals. The tabs mechanism implements this by forcing the user to focus on exactly one tab/subgoal, thus making it impossible to display items from different subgoals on the same screen, which makes interaction between them physically impossible.

[2]although we restrict ourselves to the topmost connectives of propositions, since deep reasoning is already (and better) handled by SFL.

[3]Chemical metaphors were already drawn by logicians as early as Peirce [6] and Wittgenstein [7].

[4]Interestingly, those rules are the basis for the adjunction between $\wedge$ and $\Rightarrow$ in the interpretation of IPL into cartesian closed categories.

4.2 Adding bubbles

In order to accomodate context-scoping within the chemical metaphor, we were led to explore a notion of *bubble* inspired by the *membranes* of the Chemical Abstract Machine [8]. The latter are used to delineate zones of *local* interaction, which are still porous to external data. This is precisely what we want to do here: let us consider that the user tries to prove the sequent $\Gamma \vdash A \wedge B$. By clicking on the red item $A \wedge B$, she will break it into two bubbles $(\!|\vdash A|\!)$ and $(\!|\vdash B|\!)$. Then she might decompose A and B further into sequents $\sigma_A = \Gamma_A \vdash C_A$ and $\sigma_B = \Gamma_B \vdash C_B$, and use hypotheses from Γ by dragging them inside either $(\!|\sigma_A|\!)$ or $(\!|\sigma_B|\!)$. However, hypotheses from Γ_A and Γ_B cannot be dragged out from their respective bubble, since then they could be used in the other bubble and violate context-scoping.

Bubbles can be seen as a way to internalize in the syntax of sequents the notion of subgoal, which requires in turn to allow nesting of sequents inside each other. The proof state is not a set of subgoals anymore, but a single nested sequent of this sort, that we call a *solution.* In textual syntax, solutions σ are generated by the following grammar:

$$\sigma ::= \Gamma \, \langle \sigma_1 \, ; \, \dots \, ; \, \sigma_n \rangle \, \Delta \qquad\qquad \Gamma ::= A_1, \dots, A_n \qquad\qquad \Delta ::= \varnothing \mid A$$

Inference rules are just rewriting rules on solutions, and a proof of a solution is a sequence of rewrites starting from (or in our proof-search setting, ending with) the empty solution $\langle\rangle$. Thus we arrived at a formalism which is a blend of deep inference and sequent calculus, and therefore that can express the rules associated with both click and DnD actions. We call this system the *single-succedant intuitionistic bubble calculus*, or $\mathsf{BJ_s}$ for short, and a more visual presentation in terms of multiset rewriting as in [8] is available in draft [9].

Beyond the recovered uniformity of the user interface in terms of the chemical metaphor, $\mathsf{BJ_s}$ exhibits many features that are interesting both at the proof-theoretical and user-experience levels:

- It implements a form of *context-sharing* between subgoals: that is, one can perform transformations on shared hypotheses (forward reasoning) without going back to a proof state anterior to the splitting of said subgoals.

- The tree structure of subgoals is immediately apparent in the proof state through nested bubbles. Thus part of the information on the proof construction process, which was made implicit and temporal in the proof state history, is now made explicit and spatial in the proof state itself[5]. There are multiple ways to visualize trees on a planar surface, but if we are to maintain the bubble metaphor, *zoomable user interfaces* seem to be a right fit: they allow for efficient space management and navigation, and zooming in intuitively conveys

[5]This concern of finding an explicit graphical representation of the "motions of reasoning *in actu*", and not only the states of mind, can be found already in the works of Peirce on his existential graphs [10]. We will come back to this soon.

the idea of focusing on a specific subgoal. One could also zoom out to have an overview of the different subgoals and their shared context, something which is hard to do in current proof assistants.

- Most inference rules are *local*, in the sense that applying some action to one or two items will not involve other items (the only exceptions are clicks on blue $\bot$ and $\vee$, but the only extra item they involve is the conclusion). Non-local rules are less natural for a beginner, because they modify a global state (here other items) which is not clearly correlated to the transformed data. This is of limited importance however in our case, because sequent calculus rules always perform the same trivial operation on the global state: duplicating the whole context of hypotheses.

4.3 Reducing non-determinism

In all known sequent calculus formulations of IPL, there are at least two rules which are invariably *irreversible*:

1. a left introduction of $\Rightarrow$ (there might be many ones, as in the calculus LJT of [11]);
2. the right introduction of either:
   - $\vee$ when sequents have at most or exactly one conclusion;
   - $\Rightarrow$ when sequents have multiple conclusions, e.g. in the multi-succedant variant of LJT in [11].

In $\mathsf{BJ_s}$, this means that click actions on blue $\Rightarrow$ and red $\vee$ need to be performed in a specific order to complete proofs.

In his thesis [12], N. Guenot introduced a specific kind of nested sequent system, where like in $\mathsf{BJ_s}$ inference rules are expressed as rewriting rules. An interesting feature of these systems is that all introduction rules for connectives are *reversible*, which means that in proof-search, formulas can be completely decomposed until atoms are reached before applying other rules. Non-determinism then arises in the choice of atoms that are to be connected in axioms, as well as the choice of sub-sequents to be duplicated for reuse.

In our setting, this would translate to an interface where all click actions are redundant, and in fact even *logical connectives* could be entirely dispensed with. As in the original SFL of [2], only DnD actions would remain. But the important point is that all logical connectives would be replaced by metaphorical constructs such as bubbles, which suggest *physically* the possible transformations/inferences, and could make for a very intuitive and discoverable proving interface. Unfortunately, the systems in [12] only handle classical logic and the implicative fragment of IPL.

Thus began our quest for a nested sequent system in the style of Guenot capturing full IPL[6].

4.4 Coloring bubbles

The first direction I followed was to go from the single-succedant $\mathsf{BJ_s}$ to a multi-succedant version BJ. Thus the grammar of succedants is changed to:

$$\Delta ::= A_1, \dots, A_n$$

The main difficulty lies in the way one should interpret a multi-succedant solution σ as a formula $\lfloor \sigma \rfloor$. In the single-succedant case it was straightforward:

$$\lfloor \Gamma \langle \sigma_1 ; \dots ; \sigma_n \rangle \, C \rfloor = \left(\bigwedge_{A \in \Gamma} A \right) \Rightarrow \left(C \wedge \bigwedge_{1 \leq i \leq n} \lfloor \sigma_i \rfloor \right)$$

But in multi-succedant sequent calculi, one distributes the context Δ of conclusions in all premisses, just the same way as for the context of hypotheses Γ. A first approximation would be:

$$\lfloor \Gamma \langle \Gamma_1 \langle \mathcal{S}_1 \rangle \, \Delta_1 ; \dots ; \Gamma_n \langle \mathcal{S}_n \rangle \, \Delta_n \rangle \, \Delta \rfloor = \bigwedge_{1 \leq i \leq n} \lfloor \Gamma, \Gamma_i \langle \mathcal{S}_i \rangle \, \Delta_i, \Delta \rfloor$$

where the $\mathcal{S}_i$ denote multisets of solutions. But this only handles the distribution part, and we clearly miss a base case. Thus we add a new judgment corresponding to standard sequents $\Gamma \vdash \Delta$ that we call *subgoals*, which represent leaves of the subgoal tree:

$$\sigma ::= \Gamma \vdash \Delta \mid \Gamma \langle \mathcal{S} \rangle \, \Delta$$

The usual solutions that contain bubbles are now called *branchings*, following the terminology of [12] for a similar device in classical systems. The interpretation of subgoals is the standard one for sequents:

$$\lfloor \Gamma \vdash \Delta \rfloor = \left(\bigwedge_{A \in \Gamma} A \right) \Rightarrow \left(\bigvee_{B \in \Delta} B \right)$$

In terms of graphical representation, this induces a small overhead because we need to make a clear visual distinction between the two judgments. One possibility would be to fill the interior of branchings $\Gamma \langle \mathcal{S} \rangle \, \Delta$ (that is, the space where the Γ, Δ reside) with a different color.

The second difficulty that arises has to do with the right introduction rule of $\Rightarrow$: in shallow multi-succedant systems, the idea is to erase every succedant but the

[6]nested sequent systems for IPL based on tree-shaped proofs already exist, see e.g. [13]

conclusion of the implication. In addition to being non-local, this makes the rule irreversible, and thus unfit for our purpose.

The solution comes from the system of Fitting [13]: nesting of sequents should not only be allowed in bubbles, but also in succedants. Thus the grammar of succedants is changed to:

$$\Delta ::= \iota_1, \ldots, \iota_n \qquad\qquad \iota ::= A \mid \sigma$$

Graphically, the corresponding metaphor would be to allow *red bubbles* joined disjunctively, in addition to the gray bubbles joined conjunctively. Indeed, the same idea of localized interaction is at work: the canonical example is the formula $a \vee (a \Rightarrow b)$, which is not provable intuitionistically but has a classical proof based on the justification of the left a by the right a. Like gray bubbles, red bubbles trap hypotheses within, thus in the corresponding solution $\vdash a, (a \vdash b)$ the right a cannot be moved outside: this would give the solution $a \vdash a, b$, where the left a is now in scope of the right a. In formulas, this translates to the rejection of the classical principle $(A \Rightarrow (B \vee C)) \Rightarrow (B \vee (A \Rightarrow C))$, sometimes called Grishin (a) [14]. But in this case, the same result would be obtained by moving the left a inside the bubble, thus red bubbles must be made hermetic to red items (and stay porous to blue items).

Interestingly, one notices that these *positively* asserted physical restrictions are in exact correspondance with the propagation rules of the system BiILL_{dn} [14], more precisely with the *negatively* asserted absence of converses to the rules pl_1 and pr_2. One may be tempted to add the symmetric rules pr_1 and pl_2, which in terms of solutions means allowing nesting of solutions in antecedants:

$$\Gamma ::= \iota_1, \ldots, \iota_n$$

As in [14], this is the basis for an extension to *bi-intuitionistic logic*, where blue bubbles are interpreted as nested *subtractions* $-$. A basic presentation of the full bi-intuitionistic system BiBJ is available in draft [15]. The proofs of soundness and completeness are not written down properly in LaTeX yet, but they are fully sketched on paper.

To summarize: we have designed a sound and complete system BiBJ of nested sequents for bi-intuitionistic propositional logic, based on a novel distinction between *subgoals* $\Gamma \vdash \Delta$ and *branchings* $\Gamma \langle \mathcal{S} \rangle \Delta$. Restricting nesting to only succedants or antecedants gives respectively a system BJ for IPL or $\overline{\mathsf{BJ}}$ for dual IPL. With nesting only in branchings, one can still have a complete system by restricting the cardinal of succedants (resp. antecedants) to at most 1, and changing the right introduction rules of $\vee$ and $\Rightarrow$ (resp. left introduction rule of $-$) to their usual version. In this way we have almost gone back full circle to BJ_{s}, modulo the splitting of judgments in two.

4.5 From bubbles to existential graphs

There are two problems remaining with BJ:

- The left introduction rule of $\Rightarrow$ is still irreversible, because based on the standard sequent calculus formulation instead of the calculus of structure-inspired version of [12]. This allows a neat symmetry with the dual-intuitionistic version, but gets in the way of the full formula decomposition property.

- The right introduction rule of $\wedge$ is subject to restrictions on the form of judgments in order to stay reversible, which also gets in the way of full decomposition.

To overcome these limitations, a first intuition I had was to drop the "gray bubbles" or branchings encoding the subgoal tree, and replace them by "colored zones" corresponding to a polarization of said branchings. A very experimental system called the *pasting calculus* implementing this idea is available in draft [16]. It turned out to be very complicated to formulate, making it hard to even prove soundness, and unfit as an intuitive graphical proof system.

Very soon after, I stumbled by chance upon a graphical proof formalism developed by the logician C. S. Peirce at the end of his life called *existential graphs* (hereafter EGs). I was surprised to see that it is based on a metaphor very similar visually to bubbles, that Peirce calls *cuts*. The difference is that cuts are a way to represent *negation*, and the *conjunctive* interpretation of bubbles is handled in Peirce's formalism by simple juxtaposition of graphs.

This can be understood as the consequence of a shift in viewpoint on *polarities*: whereas the difference between negative and positive knowledge was a property of *statements* in our paradigm (being a blue or a red item), it becomes a property of the *space* where statements are uttered in EGs (being in an odd or even number of cuts). I had a glimpse of this intuition with the colored zones, but it was still polluted by the red/blue paradigm attributing different polarities to objects residing in the same space. The main insight of Peirce is that *change of polarity* should be tightly coupled with *context scoping*[7].

4.6 From existential graphs to flowers

EGs, in their *alpha* variant, form a system capturing classical propositional logic. By the functional completeness of the $\{\neg, \wedge\}$ fragment, they satisfy the full formula decomposition property quite trivially: just replace $\neg A$ by a so-called cut around A, and $A \wedge B$ by the juxtaposition of A and B. Peirce devised a sound and complete set of deep inference rules on graphs, thus achieving a fully graphical, formula-free proof system.

From there, I tried to design an intuitionistic variant of EGs, which would exhibit the same properties. It turns out that this has already been done in [18], with a calculus called GrIn. However, GrIn adds on top of EGs a number of rules and axioms

[7]This intuition seems to be also present in the tensorial logic of Melliès [17], where juxtaposition and cuts can represent respectively the tensor and tensorial negation.

that are not very natural in our opinion. It is also based on a graphical syntax which has the merit of being very close to EGs, but is cumbersome to manipulate in practice because of the high level of nesting of cuts.

My idea was to turn the so-called *inloops* (in EGs, the inner cut in a double-cut) inside-out: when there is a sufficient number of them, they start looking like *petals* of a flower disposed around the *outloop*, which thus becomes a kind of *pistil*. This botanical metaphor inspired in turn a simplification of the rules, giving a new system dubbed the *flower calculus*, or FJ. I also devised an inductive, nested-sequent style syntax for flowers, which could prove useful in the comparison with previous systems like BJ. I have sketched proofs of soundness and completeness for FJ by bisimulation with sequent calculus. In particular, soundness is much simpler than in BJ thanks to a straightforward interpretation of flowers into formulas. A succint presentation of the system is available in draft [19]. I also have an experimental extension to FOL in draft [20].

5 Future works

Following is a non-exhaustive list of possible continuations for my research.

5.1 Actema

- Interfacing with Coq

5.2 Subformula linking

- Proof of productivity property in Coq
- Extensions and variants: classical (bilateral), HOL, type theory
- Computational content:
  - tangible functional programming [21] ($\wedge$, $\Rightarrow$)
  - axiom elimination in stellar resolution [22] ($\otimes$, $\parr$)

5.3 Bubbles

- Complete the first-order extension
- Investigate the relations with classical and linear logic through restrictions on resources and flow rules
- Internal cut-elimination
- Graphical ZUI with physical boundary detection for BJ_s?

5.4 Flowers

- Complete the first-order extension
- Prove completeness of the natural fragment for IPL
- Fix and formalize the proof-search algorithm based on the natural fragment, and prove its completeness
- Links between proof search, connection method [23], and combinatorial proofs [24]
- Encoding of SFL by double-pollination (as in proof search)
- Notion of cut and cut-elimination
- Computational content/Curry-Howard (possibly through SFL)
- Dual/bi-intuitionistic logic?
- Extensions and variants: higher-order logic, type theory, linear
- Graphical ZUI with physical boundary detection

References

[1] Yves Bertot, Gilles Kahn, and Laurent Théry. Proof by pointing. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789, pages 141–160. Springer Berlin Heidelberg, 1994. Series Title: Lecture Notes in Computer Science. URL: `http://link.springer.com/10.1007/3-540-57887-0_94`, `doi:10.1007/3-540-57887-0_94`.

[2] Kaustuv Chaudhuri. Subformula linking as an interaction method. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, volume 7998, pages 386–401. Springer Berlin Heidelberg, 2013. Series Title: Lecture Notes in Computer Science. URL: `http://link.springer.com/10.1007/978-3-642-39634-2_28`, `doi:10.1007/978-3-642-39634-2_28`.

[3] Pablo Donato, Benjamin Werner, and Pierre-Yves Strub. A drag-and-drop proof tactic. TYPES, 27th International Conference on Types for Proofs and Programs, 2021. URL: `https://types21.liacs.nl/download/a-drag-and-drop-proof-tactic/`.

[4] Pablo Donato, Benjamin Werner, and Pierre-Yves Strub. A drag-and-drop proof tactic. ThEdu, 10th International Workshop on Theorem proving components for Educational software, 2021. URL: `https://www.uc.pt/en/congressos/thedu/ThEdu21/programme`.

[5] Pablo Donato, Pierre-Yves Strub, and Benjamin Werner. A drag-and-drop proof tactic. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, page 197–209, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3497775.3503692`.

[6] Don D. Roberts. *The Existential Graphs of Charles S. Peirce*, chapter 2, pages 17–18. The Hague: Mouton, 1973. URL: `https://www.felsemiotica.com/descargas/Roberts-Don-D.-The-Existential-Graphs-of-Charles-S.-Peirce.pdf`.

[7] Ludwig Wittgenstein. *Tractatus logico-philosophicus.* 1922.

[8] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 81–94, New York, NY, USA, December 1989. Association for Computing Machinery. `doi:10.1145/96709.96717`.

[9] Pablo Donato. Intuitionistic single-succedant bubble calculus. 2021. URL: `http://www.lix.polytechnique.fr/Labo/Pablo.DONATO/drafts/IPM-monoconcl.png`.

[10] Don D. Roberts. *The Existential Graphs of Charles S. Peirce*, chapter 7, pages 112–113. The Hague: Mouton, 1973. URL: `https://www.felsemiotica.com/descargas/Roberts-Don-D.-The-Existential-Graphs-of-Charles-S.-Peirce.pdf`.

[11] Roy Dyckhoff. Contraction-Free Sequent Calculi for Intuitionistic Logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992. Publisher: Association for Symbolic Logic. `doi:10.2307/2275431`.

[12] Nicolas Guenot. *Nested Deduction in Logical Foundations for Computation.* phdthesis, Ecole Polytechnique X, April 2013. URL: `https://pastel.archives-ouvertes.fr/pastel-00929908`.

[13] Melvin Fitting. Nested Sequents for Intuitionistic Logics. *Notre Dame Journal of Formal Logic*, 55(1):41–61, 2014. URL: `http://projecteuclid.org/euclid.ndjfl/1390246437`, `doi:10.1215/00294527-2377869`.

[14] Ranald Clouston, Jeremy Dawson, Rajeev Goré, and Alwen Tiu. Annotation-Free Sequent Calculi for Full Intuitionistic Linear Logic. page 18 pages, 2013. Artwork Size: 18 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. URL: `http://drops.dagstuhl.de/opus/volltexte/2013/4198/`, `doi:10.4230/LIPICS.CSL.2013.197`.

[15] Pablo Donato. Bi-intuitionistic bubble calculus. 2021. URL: `http://www.lix.polytechnique.fr/Labo/Pablo.DONATO/drafts/BiBJ.pdf`.

[16] Pablo Donato. Pasting calculus. 2021. URL: `http://www.lix.polytechnique.fr/Labo/Pablo.DONATO/drafts/pasting.pdf`.

[17] Paul-André Melliès. *Une étude micrologique de la négation*, chapter 4.

[18] Ma Minghui and ahti Veikko Pietarinen. A graphical deep inference system for intuitionistic logic. *Logique et Analyse*, 245:73–114, January 2019. URL: `http://www.scopus.com/inward/record.url?scp=85066258215&partnerID=8YFLogxK`, `doi:10.2143/LEA.245.0.3285706`.

[19] Pablo Donato. Flower calculus. 2021. URL: `http://www.lix.polytechnique.fr/Labo/Pablo.DONATO/drafts/FJ.pdf`.

[20] Pablo Donato. 1st-order flower calculus. 2021. URL: `http://www.lix.polytechnique.fr/Labo/Pablo.DONATO/drafts/FJ1.pdf`.

[21] Conal M. Elliott. Tangible functional programming. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, page 59–70, New York, NY, USA, 2007. Association for Computing Machinery. `doi:10.1145/1291151.1291163`.

[22] Boris Eng. A gentle introduction to Girard's Transcendental Syntax for the linear logician. April 2022. URL: `https://hal.archives-ouvertes.fr/hal-02977750`.

[23] Jens Otten and Christoph Kreitz. A connection based proof method for intuitionistic logic. volume 918, pages 122–137. January 2006. `doi:10.1007/3-540-59338-1_32`.

[24] Willem B. Heijltjes, Dominic J. D. Hughes, and Lutz StraBburger. Intuitionistic proofs without syntax. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, Vancouver, BC, Canada, June 2019. IEEE. URL: `https://ieeexplore.ieee.org/document/8785827/`, `doi:10.1109/LICS.2019.8785827`.