Subformula Linking as an Interaction Method

Kaustuv Chaudhuri

Subformula Linking as an Interaction Method

Kaustuv Chaudhuri
INRIA, France
kaustuv.chaudhuri@inria.fr

26 July, 2013

Abstract

Current techniques for building formal proofs interactively involve one or several proof languages for instructing an interpreter of the languages to build or check the proof being described. These linguistic approaches have a drawback: the languages are not generally portable, even though the nature of logical reasoning is universal. We propose a somewhat speculative alternative method that lets the user directly manipulate the text of the theorem, using non-linguistic metaphors. It uses a proof formalism based on linking subformulas, which is a variant of deep inference (inference rules are allowed to apply in any formula context) where the relevant formulas in a rule are allowed to be arbitrarily distant. We substantiate the design with a prototype implementation of a linking-based interactive prover for first-order classical linear logic.

1 Introduction

Formal proofs are more fundamentally computational than mathematical. Say we want to prove $(a \supset b) \supset (b \supset c) \supset (a \supset c)$. Current proof languages are generally always languages of *interaction*, so a proof is an arrangement of instructions to a stateful interpreter that allows it to build (or check) an underlying proof object in a trusted natural deduction or a sequent calculus formulation of the logic. A formal proof of this theorem is, therefore, something like this;

> Suppose (1) $a \supset b$, (2) $b \supset c$, (3) a; to show c, we backchain (2) to change the goal to b, then we backchain (1) to change it to a, which we already have by (3).

(Most proof languages can express the above more succinctly.) This proof naturally resembles a computational trace where the steps of the computation are the (primitive or derived) inference rules of the logic, possibly with the use of additional lemmas. However, this formal proof obscures the simple mathematical intuition behind the proof, function composition; indeed, the composition is fully unfolded and sequentialized. In order to write the more natural mathematical proof, the user would first have to find a lemma in the standard library (or write one) that implements this intuition.

Because formal proof languages are more closely tied to the underlying calculus, and often to formal theories, they are generally not portable as such across different systems. In the rare instances where two different formal proof systems (take, *e.g.*, Coq and Isabelle) do exchange proof objects, they tend to be low-level "logical bytecode" (λ-terms built internally by the proof tactics or rewriting traces) that bear no resemblance to the proof text a user actually writes and are not expected to be read by humans (some examples: [17, 5]). Only the most committed (and masochistic) user learns to be proficient in many systems, so the communities of users of formal systems have become rather balkanized. It is worth asking if there is a sense in which one can build an interactive prover free of the shackles of language.

To answer this question, let us begin by noting that a proof fundamentally consists of two varieties of reasoning: reasoning about the subformula *structure*, and reasoning about *links* between different subformulas. Examples of the first variety are: to prove the goal $A \wedge B$ create two independent subgoals for A and B, or to use an assumption $\forall x.\, A$ add an instantiation $[t/x]A$ as an additional assumption. Examples of the second variety are: to prove the goal A search for an assumption A (use of hypothesis), or to prove the goal C first prove A and then show that C follows from A (cut or substitution). The language for establishing the links is surprisingly similar across proof languages; they generally always use hypothesis labels and terms such as `apply` or `assert`. Structural reasoning differs widely, however; at one end of a spectrum are languages based on tactics that drive a stateful prover through a proof tree, while at the other end are languages that reify the proof tree textually in the proof document but do not prescribe an order of evaluation. Crucially, though, the entire point of the structural reasoning is to enable the links, which are the only ways to finish and compose proofs and are therefore the essence of a proof, so the surprising design focus on structural reasoning seems misplaced.

How can one make linking the primary aspect of formal proof construction? Unsurprisingly, this question has been asked and answered a number of times, but never satisfactorily. This paper combines two existing

answers—*proof by pointing* [4] and the *calculus of structures* [7, 15]—in a way that improves on both and suggests a research direction.

Proof by pointing is the first attempt to systematically construct a non-linguistic interaction method on top of an existing proof interaction language such as the tactics language of Coq [3]. The user proves a theorem by a sequence of mouse clicks on subformulas; each click brings the indicated formula to the foreground of the current goal, and possibly creates additional background subgoals. Depending on the occurrence of the subformula, each click is interpreted as a sequence of tactics that mimics a sequence of sequent rules. Once a subformula is brought into the foreground on both sides of the sequent arrow, the corresponding link is established and the goal is closed. The problem with the pointing approach appears after this closure: the remaining subgoals that were produced to establish the link are left in an "exposed" state and may be difficult to reason about subsequently. To illustrate, in the earlier example of $(a \supset b) \supset (b \supset c) \supset (a \supset c)$ the user can click on (*i.e.*, point to) the two occurrences of b. The residual goals after the main goal is closed are: $a \supset b \vdash a$ and $a \supset b, b \supset c, c \vdash a \supset c$; the former goal is unprovable! This does not correspond to the intuition of linking the two bs, *i.e.*, of composing the assumptions $a \supset b$ and $b \supset c$ to get a new assumption $a \supset c$, which would match the succedent of the original theorem.

The problem with the example above is that interpreting clicks as applications of sequent rules destroys the surrounding formula context by bringing the indicated subformula to the top of the sequent. The interaction method is inherently *deep* (working on arbitrary subformulas), but the underlying shallow calculus prevents deep reasoning. The user is forced to think about the order of clicks to ensure that the sequents have the right shape. For instance, in the above example the user would have to click on the rightmost a first, which would ensure that when the bs are finally clicked on that the first residual subgoal is not $a \supset b \vdash a$ but $a, a \supset b \vdash a$.

To escape the bureaucracy of sequential syntax, we require a calculus of *deep inference* that can preserve formula contexts and thereby perform logically unrelated reasoning steps in a free order. In this paper we use the *calculus of structures* where there is no difference between a formula and a sequent, and proof steps are allowed to apply in any formula context. Unfortunately, the calculus of structures by itself cannot support our desired linking procedure above either, because every rule in the calculus is still limited to immediate operands of the principal connectives. (The two bs in the example were only ancestrally linked.) We need to generalize the calculus of structures so that the interacting subformulas may be themselves separated by other subformulas.

As the main technical contribution, this paper gives a calculus for *linked structures* for first-order classical linear logic (Sec. 3). We pick linear logic because: (1) it seamlessly encodes many intuitionistic and classical logics and has clean proof-theoretic foundations; (2) it directly represents resource-nondeterministic choices (splitting and sharing) that are essential when reasoning about computations; and (3) it is an extension to proof by pointing that was left to future work in [4, 11]. We substantiate the claims in the paper by a prototype implementation of a linking-based proof-assistant called *Profound* (Sec. 4). Our implementation is not currently integrated with any mainstream theorem proving system, so it should be seen as conceptual and somewhat speculative.

2 The Calculus of Structures for Linear Logic

Let us begin with an overview of the calculus of structures for classical first-order linear logic. Formulas (written $A, B, \dots$) have the following grammar,

$$\begin{aligned} A, B, \dots ::= {} & a \mid A \otimes B \mid \mathsf{1} \mid A \oplus B \mid \mathsf{0} \mid !A \mid \exists x.\, A \\ & \mid \overline{a} \mid A \parr B \mid \bot \mid A \mathbin{\&} B \mid \top \mid ?A \mid \forall x.\, A \end{aligned}$$

Atomic formulas (or *atoms*) are written using $a, b, \dots$, and the negation of a is written as $\overline{a}$. We use the term *literal* to refer to either an atom or a negated atom. Each atomic formula is of the form $p(t_1, \dots, t_n)$ where p is a predicate symbol and $t_1, \dots, t_n$ are first-order terms (written $s, t, \dots$) formed from term variables (written $x, y, \dots$) and applications of function symbols (written $f, g, \dots$) to terms. As is standard, we assume an ambient *signature* for the logic that assigns arities to predicate and function symbols. Formulas are in negation-normal form with each vertical column in the above grammar depicting one De Morgan dual pair; we write $\overline{A}$ for the dual of A.

For our definition of *truth* in this logic, we use a standard one-sided cut-free sequent calculus—called system *LLK*—consisting of sequents of the form $\vdash \Gamma$, where Γ is a multiset of formulas. Figure 1 contains the rules of *LLK*. The meta-theory of this calculus can be found in any standard reference [10, 19].

The calculus of structures is a formalism where there is no difference between formula and sequent. It is a rewrite system where some entailment $\vdash \overline{A}, B$ are turned into rewrite rules that replace, reading from conclusion to premises, a subformula B in a formula with A. The system is best described in a *contextual form*, where a *formula context* (written $\chi, \xi, \dots$) is formed by replacing a single subformula of a formula by the *hole* ($\Box$). We write $\chi\{A\}$ for the formula formed by replacing the hole in χ by A (possibly capturing the free variables of A).

Figure 2 contains the rules of the classical linear system *CLS*, which is a minor variant of the system *LS* from [7]. Each rule is to be understood as closed over a formula context; to illustrate, the rule id represents this

$$\frac{}{\vdash \overline{a}, a}\,\mathsf{init} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\,\mathsf{weak} \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\,\mathsf{contr}$$

$$\frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\otimes \qquad \frac{}{\vdash 1}\,1 \qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus A_2}\oplus \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,! \qquad \frac{\vdash \Gamma, [t/x]A}{\vdash \Gamma, \exists x.\, A}\,\exists$$

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,\& \qquad \frac{}{\vdash \Gamma, \top}\,\top \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\,\bot \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\,? \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x.\, A}\,\forall$$

Figure 1: Rules of *LLK*. In the $\oplus$ rule, $i \in \{1, 2\}$. In the ! rule, $?\Gamma$ stands for a multiset of formulas each of which is prefixed by ?. In the $\forall$ rule, x is not free in the conclusion.

$$\frac{1}{a \parr \overline{a}}\,\mathsf{id} \qquad \frac{B \parr A}{A \parr B}\,\mathsf{com} \qquad \frac{(A \parr B) \parr C}{A \parr (B \parr C)}\,\mathsf{asc} \qquad \frac{A}{A \parr \bot}\,\mathsf{bot} \qquad \frac{\bot}{?A}\,\mathsf{wk} \qquad \frac{?A \parr ?A}{?A}\,\mathsf{con}$$

$$\frac{(A \parr C) \otimes B}{(A \otimes B) \parr C}\,\mathsf{lsp} \qquad \frac{B \otimes (A \parr C)}{A \parr (B \otimes C)}\,\mathsf{rsp} \qquad \frac{A_i}{A_1 \oplus A_2}\,\mathsf{chc} \qquad \frac{!(?A \parr B)}{?A \parr !B}\,\mathsf{bng} \qquad \frac{[t/x]A}{\exists x.\, A}\,\mathsf{wit}$$

$$\frac{(A \parr B) \,\&\, (A \parr C)}{A \parr (B \,\&\, C)}\,\mathsf{add} \qquad \frac{\top}{A \parr \top}\,\mathsf{top} \qquad \frac{A}{?A}\,\mathsf{drl} \qquad \frac{\forall x.\, (A \parr B)}{(\forall x.\, A) \parr B}\,\mathsf{all}$$

$$\frac{1}{1 \otimes 1} \qquad \frac{1}{1 \,\&\, 1} \qquad \frac{1}{\top} \qquad \frac{1}{!1} \qquad \frac{1}{\forall x.\, 1}$$

Figure 2: Rules of *CLS*. In the chc rule, $i \in \{1, 2\}$. In the all rule, x is not free in B.

general schema.

$$\frac{\chi\{1\}}{\chi\{a \parr \overline{a}\}}$$

The first line of rules in Fig. 2 are the structural rules of the system. They are, in order, the atomic identity rule; rules for commutativity, associativity, and unit for $\parr$ (due to which $\langle \parr, \bot \rangle$ behaves as a commutative monoid structure); and weakening and contraction for ?-formulas. The second line contains logical rules corresponding to the positive connectives: splits (lsp and rsp) for $\otimes$, choices (chc) for $\oplus$, promotion (bng), and instantiation (wit). In the wit rule, the witness term t is only allowed to mention the bound variables in scope at the hole; in other words, the signature over which conclusion and premise are well-formed does not change on the application of any rule. The third line of rules contain the logical rules for the negative connectives: distributivity (add) for &, absorption (top) for $\top$, dereliction (drl) for ?, and scope-enlargement of universal quantification (all). The final line of rules define the mechanisms of combining different successful "branches" of a *CLS* proof.

A *derivation* ϕ of B from A in any unary proof-system $\mathcal{S}$, written $\phi : A \xrightarrow{\mathcal{S}} B$, is a sequence of rules of $\mathcal{S}$ with the bottom-most rule having conclusion B and the top-most rule having premise A. We write just $A \xrightarrow{\mathcal{S}} B$ to assert that there is a ϕ such that $\phi : A \xrightarrow{\mathcal{S}} B$. A *proof* ϕ of A in *CLS* is a derivation $\phi : 1 \xrightarrow{CLS} A$. Under this notion of proof, *CLS* is sound and complete with respect to the usual linear truth given in terms of *LLK*.

Theorem 1 (Soundness of *CLS* wrt. *LLK*)**.** *If* $A \xrightarrow{CLS} B$*, then* $\vdash \overline{A}, B$ *in LLK.*

Proof. Every inference rule in *CLS* with premise A and conclusion B corresponds to a provable sequent $\vdash \overline{A}, B$ of *LLK*. The result follows by a sequence of applications of cut in *LLK*. □ □

Theorem 2 (Completeness of *CLS* wrt. *LLK*)**.** *If* $\vdash A_1, \ldots, A_n$ *in LLK, then* $1 \xrightarrow{CLS} A_1 \parr \cdots \parr A_n$.

Proof. Under the interpretation of an *LLK* sequent $\vdash A_1, \ldots, A_n$ as the formula $A_1 \parr \cdots \parr A_n$, every rule of *LLK* is derivable in *CLS*. Generally speaking, one *LLK* rule corresponds to a sequence of *CLS* rules; in particular, the $\otimes$ rule of *LLK* corresponds to a sequence of applications of lsp and rsp, while the & rule corresponds to a sequence of applications of add. □ □

Although the system *CLS* is cut-free, the following cut rule is admissible:

$$\frac{\chi\{A \otimes \overline{A}\}}{\chi\{\bot\}}\,\mathsf{cut}$$

As usual, this rule is just the dual of the generalized identity (gid) rule:

$$\frac{\chi\{1\}}{\chi\{A \parr \overline{A}\}}\,\mathsf{gid}$$

Indeed, every rule of *CLS* has a dual form where the premise and conclusion are exchanged and dualized. The dual system of *CLS*, which we write as $\overline{CLS}$, is a system of *refutation*, where a refutation of A is a proof $A \xrightarrow{\overline{CLS}} \bot$, and the system that contains both *CLS* and $\overline{CLS}$ is a system where both cut and identity can be reduced to their atomic forms [7, 15]. Note that the rules lsp and rsp are self-duals, and therefore in the intersection of *CLS* and its dual system.

3 Formula Linking

The system *CLS* forms the foundation of the underlying calculus of the formula linking approach. As mentioned in the introduction, we would like to support a mode of interaction where the user can indicate that any two subformulas are to be linked. The rules of *CLS* are not well suited for such interactions because each inference rules operates on two immediately $⅋$-joined formulas.

The essence of linking is to generalize the interaction from $⅋$-joined formulas to formulas that are (eventually) ancestrally joined by $⅋$. Then, the user can simply mark the formulas they wish to link and the system will apply the right sequence of *CLS* rules to bring the linked formulas into the same context. If the linked formulas are dual literals, then the system can simply replace the $⅋$-formula with 1. The user can therefore repeatedly link dual literals when constructing a proof. (The user is of course free to link other non-atomic subformulas as well.) Importantly, the user is allowed to make incorrect linkages that render the formula unprovable. We require only that any provable formula be also provable by a linkage simplification procedure.

First, let us make the notion of linkage formal. We enlarge the syntax of formulas with a new form, $\mathfrak{u}::A$, called a *linkage*, which indicates that the formula A is *marked* by a *link* $\mathfrak{u}$. (Links are drawn from some infinite set that is disjoint from the set of variables.) We write $\mathfrak{u} \in C$ to assert that C contains a subformula of the form $\mathfrak{u}::A$. We require that every linkage in a formula occurs *exactly twice*, and that the two subformulas marked by a link be ancestrally joined by $⅋$ in the subformula ordering. In other words, for each link $\mathfrak{u} \in C$, there must exist formula contexts ξ, χ_1, χ_2 and subformulas A and B of C such that $C = \xi\{\chi_1\{\mathfrak{u}::A\} ⅋ \chi_2\{\mathfrak{u}::B\}\}$. These restrictions will be preserved by the linkage simplification rules.

The user initiates a linkage by marking two distinct ancestrally $⅋$-joined subformulas of the goal formula by a fresh link. We write this as an inference rule that produces a new kind of connective $*$ (called *interaction*), as follows:

$$\frac{\xi\{\chi_1\{\mathfrak{u}::A\} * \chi_2\{\mathfrak{u}::B\}\}}{\xi\{\chi_1\{A\} ⅋ \chi_2\{B\}\}}\,\mathsf{lnk}$$

We restrict this rule to only be applicable when the conclusion formula is free of all linkages, although this restriction can often be relaxed. Note that unlike the rules of *CLS* (Fig. 2), the rule lnk is *doubly deep*: not only is the principal formula allowed to occur in any formula context, but also the result of applying the rule has an effect on subformulas that are not necessarily the immediate descendants of the principal formula.

Semantically, $*$ has the same truth value as $⅋$, but the rules applicable to $*$ have more restrictions than the analogous rules for $⅋$. If the immediate operands of the interaction are the linkages—*i.e.*, if the context χ_1 and χ_2 in the rule lnk above are just holes—then one of the following two rules will be applied.

$$\frac{\chi\{1\}}{\chi\{\mathfrak{u}::a * \mathfrak{u}::\overline{a}\}}\,\mathsf{lnid} \qquad \frac{\chi\{A ⅋ B\}}{\chi\{\mathfrak{u}::A * \mathfrak{u}::B\}}\,\mathsf{unlnk}$$

where the unlnk rule is only applicable when A and B are not dual literals.

The remaining rules for interaction connectives are used to permute it into smaller contexts. Defining these rules requires a bit of caution, as best illustrated by an example.

Example 3. *Let us temporarily ignore linkages and just consider the interaction connective $*$. Consider the provable formula $(a \otimes b) ⅋ (\overline{a} \,\&\, \overline{b}) ⅋ (\overline{a} \oplus \overline{b})$. Suppose the common $⅋$-ancestor a and the leftmost $\overline{a}$ is turned into $*$ with* lnk*.*

$$\frac{\big((a \otimes b) * (\overline{a} \,\&\, \overline{b})\big) ⅋ (\overline{a} \oplus \overline{b})}{(a \otimes b) ⅋ (\overline{a} \,\&\, \overline{b}) ⅋ (\overline{a} \oplus \overline{b})}\,\mathsf{lnk}$$

Let us attempt to use the same rules as $⅋$ in CLS for the $*$-formulas. If we immediately apply a lsp *to the interaction formula in the premise above, we would obtain an unprovable formula:*

$$\frac{\big((a * (\overline{a} \,\&\, \overline{b})) \otimes b\big) ⅋ (\overline{a} \oplus \overline{b})}{\big((a \otimes b) * (\overline{a} \,\&\, \overline{b})\big) ⅋ (\overline{a} \oplus \overline{b})}\,\mathsf{lsp}$$

The issue here is that lsp *forces both $\overline{a}$ and $\overline{b}$ (that are $\&$-joined) to be "sent" to exactly one of the operands of the $\otimes$, but the theorem is only provable if exactly one of the $\otimes$-operands gets exactly one of the $\&$-operands. This can be achieved by a different order of the inference rules, performing* add *before* lsp *as follows.*

$$\frac{\frac{\big(((a * \overline{a}) \otimes b) \,\&\, (a \otimes (b * \overline{b}))\big) ⅋ (\overline{a} \oplus \overline{b})}{\big(((a \otimes b) * \overline{a}) \,\&\, ((a \otimes b) * \overline{b})\big) ⅋ (\overline{a} \oplus \overline{b})}\,\mathsf{lsp}\ \textit{and}\ \mathsf{rsp}}{\big((a \otimes b) * (\overline{a} \,\&\, \overline{b})\big) ⅋ (\overline{a} \oplus \overline{b})}\,\mathsf{add}$$

$$\frac{(\chi\{u::A\} * \rho\{u::C\}) \otimes B}{(\chi\{u::A\} \otimes B) * \rho\{u::C\}}\,\mathsf{lnspl} \qquad \frac{B \otimes (\rho\{u::A\} * \chi\{u::C\})}{\rho\{u::A\} * (B \otimes \chi\{u::C\})}\,\mathsf{lnspr}$$

$$\frac{!(?\chi\{u::A\} * \xi\{u::B\})}{?\chi\{u::A\} * !\xi\{u::B\}}\,\mathsf{lnbng}$$

$$\frac{(\chi\{u::A\} * \xi\{u::C\}) \,\&\, (B \parr \xi\{C\})}{(\chi\{u::A\} \,\&\, B) * \xi\{u::C\}}\,\mathsf{lnaddl} \qquad \frac{(A \parr \xi\{C\}) \,\&\, (\chi\{u::B\} * \xi\{u::C\})}{(A \,\&\, \chi\{u::B\}) * \xi\{u::C\}}\,\mathsf{lnaddr}$$

$$\frac{\forall x.\, (\chi\{u::A\} * \xi\{u::C\})}{(\forall x.\, \chi\{u::A\}) * \xi\{u::C\}}\,\mathsf{lnall} \qquad \frac{B * A}{A * B}\,\mathsf{lncom}$$

Figure 3: Rules of *CLn*

The formula in the premise is now provable in CLS under the interpretation of $*$ as $\parr$. As a general principle, if there is more than one way to interpret a linkage, we should pick an interpretation that preserves provability.

3.1 Polarities as Organizational Hints

To build our provability-preserving interaction rules, we use the well known concept of *polarity*. Some rules of *CLS* such as the split rules lsp and rsp or the choice rule chc add information to the proof; they correspond to proper implications and are not invertible as rules. Other rules such as add or all are equivalences (the conclusion implies the premise), and therefore the rules are invertible. Following general terminology, we say that the connectives with non-invertible interactions have *positive* polarity, while those with invertible interactions have *negative* polarity. More precisely, the formulas are separated into *positive formulas* (written $P, Q, \dots$) and *negative formulas* (written $N, M, \dots$) according to the following grammar.

$$\begin{array}{lll} A, B, \dots & ::= & P \mid N \\ P, Q, \dots & ::= & a \mid A \otimes B \mid \mathsf{1} \mid A \oplus B \mid \mathsf{0} \mid\, !A \mid \exists x.\, A \\ N, M, \dots & ::= & \overline{a} \mid A \parr B \mid \bot \mid A \,\&\, B \mid \mathsf{T} \mid\, ?A \mid \forall x.\, A \end{array}$$

Note that the dual of a positive formula is a negative formula and *vice versa*.

The key feature of polarity is that the rules for positive connectives commute with those for other positive connectives, and likewise for negative connectives with other negative connectives, but the rules for positive and negative connectives do not necessarily commute with each other. It is common to use this observation to design a *focused* version of a sequent calculus with alternate positive and negative *phases* of rules [1, 8, 12]. Focusing drastically reduces the non-determinism during proof search (see the experimental results in, *e.g.*, [8, 13]), but it also clarifies the structure of sequent proofs by identifying, precisely, the informative portions of full sequent proofs. The calculus of structures can also be focused [7], although the phase structure is not as evident there because of the incremental and interleaved nature of deep inference. We can, of course, adopt the full focusing discipline in this paper, but the additional structure on proofs is sometimes counter-intuitive from a user's perspective. Instead, we will use features from focusing *solely* to handle the interaction connective $*$; in other words, $A \parr B$ is the ordinary unfocused formula, while $A * B$ denotes a *focused interaction* where, if either A or B is positive then it behaves as the analogue of a focused formula (called the *head* in [7]). Note that, unlike the system in [7], there can be zero, one, or two positive formulas in a focused interaction.

Figure 3 defines the system *CLn* of these focused rules for the interaction connective. In these rules, ρ represents a positive formula context, *i.e.*, for any formula A, the substitution $\rho\{A\}$ is either A itself or a positive formula. *CLn* can obviously be seen as a subsystem of *CLS* with the interpretation of $*$ as $\parr$ and ignoring the linkage information. In particular, it corresponds to that fragment of *CLS* that defines the interaction of two $\parr$-joined formulas. The number of occurrences of interaction formulas and marked subformulas is the same in both premise and conclusion in every rule of *CLn*. The add rule of *CLS* splits into two forms in *CLn*, depending on which of the operands of the $\&$ contains the linkage.

The system *CLn* defines all the rules for simplifying interactions. We embed it into a larger system, named *CLnS*, that contains:

- the rules lnk, lnid, and unlnk (from above);
- the rules of *CLn* (Fig. 3); and
- the *CLS* rules com, asc, bot, wk, con, chc, wit, top, drl and all the rules in the final line in Fig. 2.

CLnS is relatively complete with respect to *CLS*. This does not mean, of course, that any application of lnk will be valid; consider, for example, linking the a and the $\overline{a}$ in $!a \parr \overline{a}$. Instead, we have the following correspondence.

Theorem 4 (Completeness of *CLnS* Relative to *CLS*)**.**

1. $\chi\{\xi_3\{1\}\} \xrightarrow{CLS} \chi\{\xi_1\{a\} \parr \xi_2\{\overline{a}\}\}$ *iff* $\chi\{\xi_3\{1\}\} \xrightarrow{CLnS} \chi\{\xi_1\{a\} \parr \xi_2\{\overline{a}\}\}$.

2. *If* A *and* B *are not dual literals, then* $\chi\{\xi_3\{A \parr B\}\} \xrightarrow{CLS} \chi\{\xi_1\{A\} \parr \xi_2\{B\}\}$ *iff* $\chi\{\xi_3\{A \parr B\}\} \xrightarrow{CLnS} \chi\{\xi_1\{A\} \parr \xi_2\{B\}\}$.

Proof. Every *CLnS* proof is manifestly a *CLS* proof under the interpretation of $*$ as $\parr$ and eliding the links. In the other direction, it suffices to observe that the lnk rule can be applied in such a way that the linked subformulas are the immediate operands of the principal $\parr$. Thus, every *CLS* rule that involves an interaction of two $\parr$-joined formulas can be simulated by a lnk, the corresponding *CLn* rule, and then unlnk or lnid. Once the full *CLS* derivation has been simulated in *CLnS* in this way, we remove all but the deepest linkages on A and B, and the corresponding instances of unlnk or lnid. □ □

Corollary 5. *A is provable in CLS if and only if it is provable in CLnS.*

Proof. *CLnS* lacks all the *CLS* rules that permute a $\parr$ into a smaller context: lsp, rsp, bng, add, and all. Thus, the only way for two $\parr$-joined formulas to interact is by means lnk followed by *CLn* rules. We can thus appeal to Thm. 4. □ □

The *CLn* rules preserve the number of occurrences of linkages and interactions. Moreover, they are biased to permute the negative connectives (*i.e.*, & and $\forall$) out of the positive connectives. This guarantees that if the conclusion of any *CLn* rule is provable in *CLnS*, then so is its premise; in other words, every rule of *CLn* is invertible. The order of application of the *CLn* rules is therefore immaterial. While different orders can produce different premises, the premises are equivalent.

Theorem 6 (Order Independence of *CLn*)**.** *If* $A \xrightarrow{CLn} B$ *and* $A' \xrightarrow{CLn} B$*, then* $A \equiv A'$.[1]

Proof. Straightforward inspection of the rules of *CLn*. □ □

As a consequence, the only choices the user needs to explicitly instantiate an $\exists$ using wit, and contract a ?-formula using drl. The other choices in a proof—the multiplicative splits for $\otimes$ and disjunctive choices for $\oplus$—are *deterministically* inferred directly from the links.

Example 7. *The following CLnS proof illustrates the use of linking.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall y, z.\, 1}{\forall y, z.\, \mathrm{u}{::}\overline{p(y)} * \mathrm{u}{::}p(y)}\,\mathsf{lnid}}{\forall y, z.\, (\mathrm{u}{::}\overline{p(y)} \oplus p(z)) * (\overline{p(c)} \oplus \mathrm{u}{::}p(y))}\,\mathsf{chc}\times 2}{\forall y.\, \big(\forall z.\, \mathrm{u}{::}\overline{p(y)} \oplus p(z)\big) * \big(\overline{p(c)} \oplus \mathrm{u}{::}p(y)\big)}\,\mathsf{lnall}}{\forall y.\, \big(\exists x.\, \forall z.\, \mathrm{u}{::}\overline{p(x)} \oplus p(z)\big) * \big(\overline{p(c)} \oplus \mathrm{u}{::}p(y)\big)}\,\mathsf{wit}}{\big(\exists x.\, \forall y.\, \mathrm{u}{::}\overline{p(x)} \oplus p(y)\big) * \big(\forall y.\, \overline{p(c)} \oplus \mathrm{u}{::}p(y)\big)}\,\mathsf{lnall}}{\big(\exists x.\, \forall y.\, \mathrm{u}{::}\overline{p(x)} \oplus p(y)\big) * \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus \mathrm{u}{::}p(y)\big)}\,\mathsf{wit}}{\big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big) \parr \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big)}\,\mathsf{lnk}$$

Observe that the CLn rules are interleaved with the other rules of CLnS.

It is instructive to compare the *CLn* rules with the similar *focused interaction* rules of *LSF* [7], which is a polarized and focused variant of *CLS*. Most of the rules for positive connectives are the same, except for lnbng,[2] which differs only in the fact that the interaction $*$ does not dissipate into a $\parr$ in *CLn*. As a result, *CLn* requires rules for interactions between negative formulas as well, while *LSF* defines focused interactions only for one positive interacting with one negative formula. Conceptually, a *CLnS* proof can be viewed as an *LSF* proof where several neighbouring (focused) interactions are merged into one.

3.2 Positive Equality and First-Order Effects

Let us call a sequence of applications of *CLnS* rules that begin at the bottom with lnk and end at the top with its corresponding unlnk or lnid a *linking phase*. Of the non-*CLn* rules in Ex. 7, the instances of drl and chc are all forced and can be done automatically. The instances of wit, on the other hand, require user guidance. The quantifier structure of this theorem requires the application of at least one lnall before a wit, so in the general case the system will not be able to keep all the rules of a single linking phase together in a block, *i.e.*, the linking

[1] $A \equiv B$ can be defined as usual as $(\overline{A} \parr B)$ & $(\overline{B} \parr A)$.

[2] In the polarized setting, there would be a similar case for the $\downarrow$ connective that can be seen as a purely linear variant of !.

phase are not *atomic*. Notice that this problem is only significant in the first-order case; for the propositional connectives, we can add rules such as the following to perform a non-*CLn* rule in the middle of a *CLn* derivation.

$$\frac{(\chi\{\mathrm{u}::A\}) * \xi\{\mathrm{u}::B\}}{(?\chi\{\mathrm{u}::A\}) * \xi\{\mathrm{u}::B\}}\,\mathsf{lndrl} \qquad \frac{\chi\{\mathrm{u}::A\} * \xi\{\mathrm{u}::C\}}{(\chi\{\mathrm{u}::A\} \oplus B) * \xi\{\mathrm{u}::C\}}\,\mathsf{lnlch} \qquad \frac{\chi\{\mathrm{u}::B\} * \xi\{\mathrm{u}::C\}}{(A \oplus \chi\{\mathrm{u}::B\}) * \xi\{\mathrm{u}::C\}}\,\mathsf{lnrch}$$

To make the linking phase atomic, we add an equality predicate to the language, change lnid to introduce equations between the terms, and add the following lnwit rule to *CLn*.

$$\frac{s_1 = t_1 \otimes \cdots \otimes s_n = t_n}{\mathrm{u}::p(s_1, \ldots, s_n) * \mathrm{u}::\overline{p(t_1, \ldots, t_n)}}\,\mathsf{lnid} \qquad \frac{\exists x.\, \big(\chi\{\mathrm{u}::A\} * \xi\{\mathrm{u}::B\}\big)}{\big(\exists x.\, \chi\{\mathrm{u}::A\}\big) * \xi\{\mathrm{u}::B\}}\,\mathsf{lnwit}$$

Example 8. *Let us revisit Ex. 7 using the above rules. We have this derivation.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\exists x.\, \forall y.\, \exists x_1.\, \forall y_1.\, x_1 = y}{\exists x.\, \forall y.\, \exists x_1.\, \forall y_1.\, \mathrm{u}::\overline{p(x_1)} * \mathrm{u}::p(y)}\,\mathsf{lnid}}{\exists x.\, \forall y.\, \exists x_1.\, \forall y_1.\, (\mathrm{u}::\overline{p(x_1)} \oplus p(y_1)) * (\overline{p(x)} \oplus \mathrm{u}::p(y))}\,\mathsf{lnlch},\mathsf{lnrch}}{\exists x.\, \forall y.\, \exists x_1.\, \big(\forall z.\, \mathrm{u}::\overline{p(y)} \oplus p(y_1)\big) * \big(\overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnall}}{\exists x.\, \forall y.\, \big(\exists x_1.\, \forall y_1.\, \mathrm{u}::\overline{p(x_1)} \oplus p(y_1)\big) * \big(\overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnwit}}{\exists x.\, \big(\exists x_1.\, \forall y.\, \mathrm{u}::\overline{p(x_1)} \oplus p(y)\big) * \big(\forall y.\, \overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnall}}{\big(\exists x.\, \forall y.\, \mathrm{u}::\overline{p(x)} \oplus p(y)\big) * \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnwit}}{\big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big) \parr \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big)}\,\mathsf{lnk}$$

The derivation can now keep all the CLn rules together.

A linking phase now ends (reading from conclusion to premise) with either lnid or unlnk. In Ex. 8 above, the residual premise after the lnid is essentially a first-order unification problem. The equations formed by the instances of lnid are all positively signed atomic formulas, with the following two rules.

$$\frac{1}{x = x}\,\mathsf{refl} \qquad \frac{s_1 = s_2 \otimes \cdots \otimes s_n = t_n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)}\,\mathsf{cong}$$

Fact 9. *The rules* refl, cong, *and* $1 \longrightarrow 1 \otimes 1$ *suffice to show that* $t = t$ *for any term* t. □

Since every derivation can be embedded in any formula context, we thus obtain a smooth continuum between deduction and unification in terms of the equality predicate. This freedom comes at a price: the order of the applications of lnwit and lnall is now important.

Example 10. *This is what happens if the* lnwit *and* lnall *rules of Ex. 8 are done in the wrong order.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\exists x, x_1.\, \forall y, y_1.\, x_1 = y_1}{\exists x, x_1.\, \forall y, y_1.\, \mathrm{u}::\overline{p(x_1)} * \mathrm{u}::p(y_1)}\,\mathsf{lnid}}{\exists x, x_1.\, \forall y, y_1.\, \big(\mathrm{u}::\overline{p(x_1)} \oplus p(y)\big) * \big(\overline{p(x)} \oplus \mathrm{u}::p(y_1)\big)}\,\mathsf{lnlch},\mathsf{lnrch}}{\exists x, x_1.\, \forall y.\, \big(\mathrm{u}::\overline{p(x_1)} \oplus p(y)\big) * \big(\forall y_1.\, \overline{p(x)} \oplus \mathrm{u}::p(y_1)\big)}\,\mathsf{lnall}}{\exists x, x_1.\, \big(\forall y.\, \mathrm{u}::\overline{p(x_1)} \oplus p(y)\big) * \big(\forall y.\, \overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnall}}{\exists x.\, \big(\exists x_1.\, \forall y.\, \mathrm{u}::\overline{p(x_1)} \oplus p(y)\big) * \big(\forall y.\, \overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnwit}}{\big(\exists x.\, \forall y.\, \mathrm{u}::\overline{p(x)} \oplus p(y)\big) * \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus \mathrm{u}::p(y)\big)}\,\mathsf{lnwit}}{\big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big) \parr \big(\exists x.\, \forall y.\, \overline{p(x)} \oplus p(y)\big)}\,\mathsf{lnk}$$

The residual premise is unprovable.

It is not possible to fix this problem without further user guidance, because determining the correct quantifier nesting is equivalent to the full theorem proving problem for first-order logic, which is unsolvable. Fortunately, there is only a single critical pair that needs to be resolved: an interaction between two ∃-formulas. In every other case, we can appeal to Thm 6, suitably generalized.

Our solution therefore involves a modification to linkages. We now have two kinds of *oriented* linkages, $\mathrm{u} \gg A$ (called a *source*) and $\mathrm{u} \ll A$ (called a *sink*), with $\mathrm{u}::A$ standing for either form when irrelevant. The lnk rule is modified to produce a single source and sink pair:

$$\frac{\xi\{\chi_1\{\mathrm{u} \gg A\} * \chi_2\{\mathrm{u} \ll B\}\}}{\xi\{\chi_1\{A\} \parr \chi_2\{B\}\}}\,\mathsf{lnk}$$

Most of the remaining rules stay the same, except for the following new rule for resolving an interaction between ∃-formulas.

$$\frac{\exists y.\, \big((\exists x.\, \chi\{\mathrm{u} \gg A\}) * \xi\{\mathrm{u} \ll B\}\big)}{\big(\exists x.\, \chi\{\mathrm{u} \gg A\}\big) * \big(\exists y.\, \xi\{\mathrm{u} \ll B\}\big)}\,\mathsf{lnex}$$

Intuitively, the source is "brought to" the sink. The rules are prioritized so that the lnwit rule is only applicable in the case that lnex is not. Note that if the source is not on the left of the interaction $*$, then the lncom rule can be used to put it there. Let *CLonS* stand for:

$$\mathit{CLnS} \cup \{\mathsf{lndrl}, \mathsf{lnlch}, \mathsf{lnrch}, \mathsf{lnwit}, \mathsf{lnex}\} \cup \{\mathsf{refl}, \mathsf{cong}\}$$

(with the understanding that the lnk rule introduces oriented linkages), and let *CLon* stand for $\mathit{CLonS} \setminus \mathit{CLS}$.

Theorem 11 (Order Independence of *CLon*)**.** *If* $A \xrightarrow{\mathit{CLon}} B$ *and* $A' \xrightarrow{\mathit{CLon}} B$*, then* $A \equiv A'$*.*

Proof. Same idea as for Thm. 6. □ □

Theorem 12. *For any formula* A *free of* $=$*-subformulas,* $\mathsf{1} \xrightarrow{\mathit{CLonS}} A$ *if and only if* $\mathsf{1} \xrightarrow{\mathit{CLnS}} A$*.*

Sketch. For the forward direction (only if), the *CLonS* rules lnwit, lnex, lndrl, lnlch and lnrch are admissible in *CLnS*. To recover the *CLnS* proof, it suffices to use the wit, drl and chc rules to rewrite all the instances of lnwit, lnex, lndrl, lnlch and lnrch. Then, these rules can be permuted below any occurrences of (the *CLonS* variant of) lnid on subformulas. This leaves just the instances of refl and cong above the *CLonS* variants of lnid; but, these rules do not interact with any other rules, and can therefore be permuted to stand in a block right above the lnid that gives rise to them; the entire block can then be replaced with the *CLnS* variant of lnid.

The reverse direction (if) is a simple consequence of Fact 9. □ □

4 Implementation Notes

The classical system *CLonS* has been implemented as part of the interactive proving tool *Profound* [6]. The tool is launched by giving it a goal theorem either on the command line or using an input file. It then presents the user with an interface where the text of the theorem may be directly manipulated (using the keyboard or the mouse) in two *modes*: a *logical* mode and a *linking* mode. In the logical mode, the user is allowed to perform any of the rules in $\mathit{CLonS} \setminus \mathit{CLon}$. In other words, the only non-trivial actions available in this mode are: (1) instantiating $\exists$ (wit) and choice (chc) when there are no linkages, and (2) contraction (con) and weakening (wk). The user is also allowed the freedom to freely reorganize the formula using associativity and commutativity of $⅋$. The system lets the user also treat the other binary connectives as associative and commutative (the corresponding rules are admissible in *CLS* and *CLnS*).

In the linking mode, the user indicates an oriented link between a pair of subformulas. Using the keyboard, this amounts to traversing the subformula tree and marking two subformulas that are ancestrally $⅋$-linked as source and sink. This action is more intuitive with a pointing device such as the mouse, where one subformula is simply dragged and dropped on another. The main difficulty with pointing is to determine which subformulas the user wishes to point to, as most cursor positions occur within many subformulas. The system uses the heuristic of selecting the smallest subformula surrounding the cursor position, and letting the user go up in the subformula tree by scrolling the mouse-wheel.[3]

The current implementation of *Profound* is in OCaml and uses the GTK+ windowing library for its graphical interface, and has been tested to run on POSIX-compatible systems such as Linux and Mac OS X. In the future, we will probably re-implement it as a server/client setup using Javascript and using the HTML5 Canvas library for the user interface, which would make it portable across a wide range of platforms that support graphical web-browsers. Systems that can support multiple pointers and different interaction metaphors such as "pinching" might also enable *Profound* to perform marking or linking of formulas more directly.

In the rest of this section, we will describe some additional features of *Profound* that are not strictly part of the *CLonS* proof system, but are nevertheless very important for practical use of linking-based interaction.

4.1 Implicit Contraction

Perhaps the most natural aspect of the linking-based interaction metaphor is the ability to treat a logical problem as a kind of matching puzzle: bits of the current obligation are linked to bits of known facts and lemmas until the problem reduces to a known and manageable form. However, the formal *CLonS* system does require an additional step of explicit contraction that can complicates this simple aspect. For example, suppose we want to prove $?(a \otimes \overline{b}) ⅋ ?(a \otimes \overline{c}) ⅋ ?\overline{a} ⅋ (b \otimes c)$. The ideal linking-based proof would just link the $\overline{a}$ to the two occurrences of a. However, the first link would "consume" the $?\overline{a}$. The user is forced to explicitly contract it before drawing the links.

In *Profound*, this issue is solved by means of special variants of lnk that store a copy of the ?formula for reuse in subsequent links. One such rule is as follows.

$$\frac{\xi\{?\chi_1\{A\} ⅋ (?\chi_1\{\mathtt{u} \gg A\} * \chi_2\{\mathtt{u} \ll B\})\}}{\xi\{?\chi_1\{A\} ⅋ \chi_2\{B\}\}}$$

[3] Unfortunately, this turns out to be fairly frustrating when trying to select, for instance, the subformula $B ⅋ C$ from $A ⅋ B ⅋ C$, because $⅋$ is internally treated as a left-associative binary operator. The user is forced to first bring the entire subformula they wish to select to the left (which can be done freely in the logical mode).

Of course, this incurs the dual overhead of ?-formulas that survive unnecessarily. The user will need to clear them with explicit instances of wk, which can quickly become burdensome. From our experimentation, there does not seem to be a preferable default, so *Profound* allows both contracting and non-contracting uses of lnk. Indeed, *Profound* allows the user to independently decide to contract the source and the sink when marking them.

There is a related issue when marking the source and the sink that are not ancestrally ⅋-related, but that have an ancestral ?. The simplest example is $?(a \oplus \overline{a})$. With the *CLonS* calculus and implicit linking as described, the user will still not be able to link the a and the $\overline{a}$ without first explicitly contracting the formula. This is not an altogether contrived example: consider the actual representation of the classical Drinker's formula, $?\exists x.\, \forall y.\, (p(x) \oplus \overline{p(y)})$, as opposed to its linear version in Ex. 7. To solve this issue, we further extend the lnk rule to allow contraction at an ancestral ? when there is no ancestral ⅋ as follows.

$$\frac{\xi\{?(\chi_1\{\mathsf{u} \gg A\} \circ \chi_2\{B\}) * ?(\chi_1\{A\} \circ \chi_2\{\mathsf{u} \ll B\})\}}{\xi\{?(\chi_1\{A\} \circ \chi_2\{B\})\}}$$

where $\circ \in \{\otimes, \oplus, \&\}$.

4.2 Other Convenience Features

The *CLonS* system is cut-free, but in practice it is invaluable to use cuts in a proof, both for conceptual and textual simplicity. The *Profound* implementation therefore allows the user to introduce a cut at any moment, even in the middle of constructing a link. Instead of the general cut rule, however, we actually implement the following variant, which is a composition of the usual cut rule and rsp.

$$\frac{A \otimes (\overline{A} \parr C)}{C}$$

The rules for conjunctive truth on the last line of Fig. 2, while sufficient for completeness, are generally too cumbersome to use in practice. Instead, *Profound* actually implements the following monoidal versions (which are all derivable);

$$\frac{A}{A \circ \dagger} \qquad \frac{A}{\dagger \circ A} \qquad \frac{1}{1 \circ 1} \qquad \frac{A}{\forall x.\, A} \qquad \frac{A}{\exists x.\, A}$$

where $\langle \circ, \dagger \rangle \in \{\langle \otimes, 1 \rangle, \langle \oplus, 0 \rangle, \langle \&, \top \rangle\}$, and in the final two rules x is not free in A. In fact, these rules are not explicitly presented to the user, but are instead folded into the traversal mechanism. That is, whenever the cursor on $\dagger$ in $A \circ \dagger$ or $\dagger \circ A$, traversing to the parent simply changes the entire formula to A. This allows us to implement chc in a very simple manner: we just add the following general rule:

$$\frac{0}{A}\,\mathsf{del}$$

The user can use del to change any subformula to 0, and if it is the operand of a $\oplus$ then traversing to the parent will remove the 0 as a side effect. We find this to be a solid improvement over a direct use of chc, which would otherwise require a further clarification from the user.

5 Caveats

It is important to point out that *Profound* is only a research prototype at this point. To make it more broadly viable, we need at least two more crucial features that would require basic research on the calculus of structures.

- Support for intensional equality and induction: in order to make *Profound* suitable as a user interface for reasoning about computational specifications, the logic must be extended to support intensional (predicate) equality and induction. Equality can be supported by the rules of unification logic, which can be turned into a contextual and incremental form without too much effort. However, for practical uses it will need to support reasoning about equality transitively (*e.g.*, to show $\big((x \neq z) \otimes (x \neq s(z))\big) \parr 0$ where z and s represent zero and successor, respectively). In the general case this will amount to (incremental) congruence-closure.

  The general induction rules in the style of *LINC* [18] or *μMALL* [2] can be readily added to the deep inference formalism. However, experience suggests that proofs written using such induction rules tend to be verbose and confusing. It is more standard to use more restricted induction schemas based on subterm or lexicographic ordering, such as in the Abella system [9]. Unfortunately, such restricted schemas tend to have a global and shallow flavour that runs counter to the incremental nature of deep inference. However, the final word is far from written on this matter.

- Support for typed and higher-order reasoning: supporting typed first-order terms is completely straightforward and *Profound* already assumes that the predicates and terms are simply typed. We intend to extend it to support polymorphically typed predicates and terms in the near future. Supporting dependently typed terms is not particularly critical as dependent restrictions can be recovered relationally; nevertheless, constructing a variant of the calculus of structure for dependent types is an open problem. Second-order quantification [16] should be straightforward. Extending the calculus of structures to full higher-order logic is also open.

Acknowledgements: We thank Dale Miller and Lutz Straßburger for their help with many aspects of this work.

References

[1] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

[2] D. Baelde. Least and greatest fixed points in linear logic. *ACM Trans. on Computational Logic*, 13(1), Apr. 2012.

[3] Y. Bertot. The CtCoq system: Design and architecture. *Formal Aspects of Computing*, 11(3):225–243, Sept. 1999.

[4] Y. Bertot, G. Kahn, and L. Théry. Proof by pointing. In *Theoretical Aspects of Computer Software*, pages 141–160, 1994.

[5] M. Boespflug, Q. Carbonneaux, and O. Hermant. The λΠ-calculus modulo as a universal proof language. In D. Pichardie and T. Weber, editors, *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*, pages 28–43, 2012.

[6] K. Chaudhuri. Profound. Available from: `http://chaudhuri.info/software/profound/`, 2013.

[7] K. Chaudhuri, N. Guenot, and L. Straßburger. The Focused Calculus of Structures. In *Computer Science Logic: 20th Annual Conference of the EACSL*, Leibniz International Proceedings in Informatics (LIPIcs), pages 159–173. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Sept. 2011.

[8] K. Chaudhuri, F. Pfenning, and G. Price. A logical characterization of forward and backward chaining in the inverse method. *J. of Automated Reasoning*, 40(2-3):133–177, Mar. 2008.

[9] A. Gacek. The Abella system and homepage. `http://abella.cs.umn.edu/`, 2009.

[10] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[11] G. K. L. Thery, Y. Bertot. Real theorem provers deserve real user-interfaces. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*. ACM Press, 1992. Software Engineering Notes, Vol 17, No 5.

[12] C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

[13] S. McLaughlin and F. Pfenning. Imogen: Focusing the polarized focused inverse method for intuitionistic propositional logic. In I. Cervesato, H. Veith, and A. Voronkov, editors, *15th International Conference on Logic, Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 5330 of *LNCS*, pages 174–181, Nov. 2008.

[14] Z. Snow, D. Baelde, and G. Nadathur. A meta-programming approach to realizing dependently typed logic programming. In *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 187–198, 2010.

[15] L. Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, Technische Universität Dresden, 2003.

[16] L. Straßburger. Some observations on the proof theory of second order propositional multiplicative linear logic. In P.-L. Curien, editor, *Typed Lambda Calculi and Applications, TLCA'09*, volume 5608 of *LNCS*, pages 309–324. Springer, 2009.

[17] A. Stump. Proof checking technology for satisfiability modulo theories. In A. Abel and C. Urban, editors, *Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.

[18] A. Tiu. *A Logical Framework for Reasoning about Logical Specifications.* PhD thesis, Pennsylvania State University, May 2004.

[19] A. S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.