Subformula Linking for Intuitionistic Logic with Application to Type Theory

Kaustuv Chaudhuri

Subformula Linking for Intuitionistic Logic with Application to Type Theory

Kaustuv Chaudhuri
Inria and LIX/École polytechnique, Palaiseau, France
kaustuv.chaudhuri@inria.fr

January 17, 2022

Abstract

Subformula linking is an interactive theorem proving technique that was initially proposed for (classical) linear logic. It is based on truth and context preserving rewrites of a conjecture that are triggered by a user indicating *links* between subformulas, which can be done by direct manipulation, without the need of tactics or proof languages. The system guarantees that a true conjecture can always be rewritten to a known, usually trivial, theorem. In this work, we extend subformula linking to intuitionistic first-order logic with simply typed lambda-terms as the term language of this logic. We then use a well known embedding of intuitionistic type theory into this logic to demonstrate one way to extend linking to type theory.

1 Introduction

Suppose you want to prove a conjecture such as:

$$\big(\forall x.\,\exists y.\,a(f(x),y)\big) \wedge \big(\forall z.\,a(f(f(c)),z) \supset b(z)\big) \supset \exists u.\,b(f(u))$$

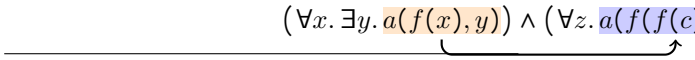
or to find replacements for the ?s that would allow a dependent type such as the following to be inhabited:

$$\Pi u{:}(\Pi x{:}a.\,\Pi y{:}(b\,x).\,c\,x\,y).\,\Pi v{:}(\Pi x{:}a.\,b\,x).\,\Pi w{:}a.\,(c\,?\,?).$$

In a mainstream interactive theorem proving system you would attempt it by giving instructions to a carefully constructed proof verification engine using a *formal proof language*, often with a *read-eval-print* loop for immediate feedback. Your instructions would guide the verifier through the twists and turns of a formal derivation until it is satisfied that all formal obligations have been established. Your language of instructions could be tactics-based (such as in Coq), or it could be a programming language itself (such as in HOL-Light or Agda); it could also have a formal *structure* or be *declarative* (such as Isabelle/Isar).[1] Despite these superficial differences, all such systems can broadly be called *linguistic* because the internal state of the verifier can only be modified by means of the formal proof language (and the whims—or semantics, if you prefer—of the interpreter of the language).

An alternative to such a linguistic system would be a system of *direct manipulation*, wherein there is a tangible representation of the state of the verifier that one can modify directly using such tools as one's fingers, pointing devices, or eye movements. The verifier's job is then to make sure that the direct manipulation attempts are allowed when they are logically permissible and prevented when they are not. A prominent example of such a direct manipulation system is the *proof by pointing* technique [3], where mouse clicks on the representation of a proof state (in a version of Coq) are given a meaning: a click on a connective deep in a formula is interpreted as a sequence of Coq tactics that bring the connective to the top, at which point it could be made to interact with the other hypotheses or the conclusion in the usual manner.

A generalization of this idea, called *proof by linking*, was proposed in [4]. It allows the user not only to point but also to *link* different subformulas, say with a multi-touch input device or with a drag-and-drop metaphor. There are two immediate benefits of linking over pointing: (1) the surrounding context of a formula is not destroyed because the linked subformulas are not brought to the top, and (2) the interaction mode is easier to describe to complete novices. For instance, a novice could be instructed to "match the atoms" for the first example above, in which case they might start by attempting the following link:



[1] These are just illustrative examples of mainstream proof systems and should not be read as assigning them a position of privilege or authority.

The linking procedure would interpret this link as a desire to "bring" the source atom "to" the destination atom. Without touching any other part of the conjecture except the smallest subformula containing both the source and the destination of the link, the conjecture would be *rewritten* to a different one:

$$\exists x.\, \forall y.\, \forall z. \Big(\big(a(f(x), y) \supset a(f(f(c)), z) \big) \supset b(z) \Big) \supset \exists u.\, b(f(u)).$$

The surrounding context of the link is preserved as nothing is brought to the top; instead, the source moves through the formula tree to meet the destination. The rewrites that underlie the transformation are *provability preserving*: if the rewritten conjecture is provable, then so is the original conjecture. Eventually, the conjecture (if true) would be reduced to a trivial theorem such as $\top$. Note that the novice user does not need to know *any* proof language to draw these links, not even a conceptual proof system such as the sequent calculus.

The original *proof by linking* technique was proposed for classical linear logic and freely exploited the *calculus of structures* [17]. In this paper we show how to adapt the technique to intuitionistic logics and intuitionistic type theories, where the calculus of structures is not so well behaved [18, 8] (or, in the case of dependent type theory, entirely missing), and where preserving the context of the rewrites is a more delicate task. We do this by first defining the technique for intuitionistic first-order logic over λ-terms, and then we use an existing complete (shallow) embedding of dependent type theory in this logic [6, 15]. A secondary contribution is to give some insight into what a deep inference formalism might look like for dependent type theory.

2 Subformula Linking for Intuitionistic First-Order Logic

This section will serve both as an introduction to the subformula linking procedure, and as evidence that the technique can be applied to intuitionistic logics. Let us do this in two phases: first for the the propositional fragment, and then extended with first-order quantification.

2.1 The Propositional Fragment

We will use the following grammar of *formulas* (written $A, B, \dots$), where *atomic formulas* are written in lowercase ($a, b, \dots$).

$$A, B, \dots ::= a \mid A \wedge B \mid \top \mid A \vee B \mid \bot \mid A \supset B$$

Following usual conventions, the connectives $\wedge$ and $\vee$ are left-associative, while $\supset$ is right-associative; the binding priority from strongest to weakest is $\wedge, \vee, \supset$.

The true formulas of this calculus can be defined in terms of derivability in a variety of formal systems such as with the sequent calculus LJ or G3ip [11]. In this paper the precise sequent calculus is not of primary concern; however, we will use the notation $\Gamma \vdash C$ where Γ is a multiset of formulas to denote that the formula C is derivable from the assumptions Γ using any such calculus.

A *positively signed formula context* (written $\mathcal{C}\{\}$) is a formula with a single occurrence of a hole $\{\}$ in the place where a positively signed subformula may occur; it is defined mutually recursively with an *negatively signed formula context* (written $\mathcal{A}\{\}$) by the following grammar, where $* \in \{\wedge, \vee\}$.

$$\begin{aligned} \mathcal{C}\{\} &::= \{\} \mid A * \mathcal{C}\{\} \mid \mathcal{C}\{\} * B \mid A \supset \mathcal{C}\{\} \mid \mathcal{A}\{\} \supset B \\ \mathcal{A}\{\} &::= A * \mathcal{A}\{\} \mid \mathcal{A}\{\} * B \mid A \supset \mathcal{A}\{\} \mid \mathcal{C}\{\} \supset B \end{aligned}$$

The *replacement* of the hole in $\mathcal{C}\{\}$ (resp. $\mathcal{A}\{\}$) with a formula A yields a new formula, which we write as $\mathcal{C}\{A\}$ (resp. $\mathcal{A}\{A\}$). For instance, if $\mathcal{C}\{\}$ is $a \wedge ((b \supset \{\}) \vee d)$, then $\mathcal{C}\{c \supset \bot\}$ is $a \wedge ((b \supset (c \supset \bot)) \vee d)$.

Theorem 1. *Suppose that $A \vdash B$. Then:*

- *for any positively signed context $\mathcal{C}\{\}$, it is the case that $\mathcal{C}\{A\} \vdash \mathcal{C}\{B\}$; and*
- *for any negatively signed context $\mathcal{A}\{\}$, it is the case that $\mathcal{A}\{B\} \vdash \mathcal{A}\{A\}$.*

Proof. Induction on the structure of the contexts $\mathcal{C}\{\}$ or $\mathcal{A}\{\}$. □ □

In order to define the subformula linking procedure for this calculus, we work with *interaction formulas*; an interaction formula is a formula where:

- either a *single* occurrence of $\supset$ is replaced with $\triangleright$,
- or a *single* occurrence of $\wedge$ is replaced with $\circ$.

Terminal rules

$$\frac{\mathcal{C}\{\top\}}{\mathcal{C}\{a \triangleright a\}}\ \mathsf{in} \qquad \frac{\mathcal{C}\{A \supset B\}}{\mathcal{C}\{A \triangleright B\}}\ \mathsf{rel}$$

(the conclusion of rel is understood as not overlapping that of in)

Positively signed rules

$$\frac{\mathcal{C}\{(A \triangleright B) \wedge F\}}{\mathcal{C}\{A \triangleright (B \wedge F)\}}\ \triangleright\wedge_1 \qquad \frac{\mathcal{C}\{F \wedge (A \triangleright B)\}}{\mathcal{C}\{A \triangleright (F \wedge B)\}}\ \triangleright\wedge_2$$

$$\frac{\mathcal{C}\{(A \triangleright B) \wedge (F \supset B)\}}{\mathcal{C}\{(A \vee F) \triangleright B\}}\ \vee\triangleright_1 \qquad \frac{\mathcal{C}\{(F \supset B) \wedge (A \triangleright B)\}}{\mathcal{C}\{(F \vee A) \triangleright B\}}\ \vee\triangleright_2$$

$$\frac{\mathcal{C}\{(A \circ B) \supset F\}}{\mathcal{C}\{A \triangleright (B \supset F)\}}\ \triangleright\supset_1 \qquad \frac{\mathcal{C}\{F \supset (A \triangleright B)\}}{\mathcal{C}\{A \triangleright (F \supset B)\}}\ \triangleright\supset_2$$

$$\frac{\mathcal{C}\{A \triangleright B\}}{\mathcal{C}\{A \triangleright (B \vee F)\}}\ \triangleright\vee_1 \qquad \frac{\mathcal{C}\{A \triangleright B\}}{\mathcal{C}\{A \triangleright (F \vee B)\}}\ \triangleright\vee_2$$

$$\frac{\mathcal{C}\{A \triangleright B\}}{\mathcal{C}\{(A \wedge F) \triangleright B\}}\ \wedge\triangleright_1 \qquad \frac{\mathcal{C}\{A \triangleright B\}}{\mathcal{C}\{(F \wedge A) \triangleright B\}}\ \wedge\triangleright_2 \qquad \frac{\mathcal{C}\{F \wedge (A \triangleright B)\}}{\mathcal{C}\{(F \supset A) \triangleright B\}}\ \supset\triangleright$$

Negatively signed rules

$$\frac{\mathcal{A}\{(A \circ B) \vee F\}}{\mathcal{A}\{A \circ (B \vee F)\}}\ \circ\vee_1 \qquad \frac{\mathcal{A}\{F \vee (A \circ B)\}}{\mathcal{A}\{A \circ (F \vee B)\}}\ \circ\vee_2$$

$$\frac{\mathcal{A}\{A \circ B\}}{\mathcal{A}\{A \circ (B \wedge F)\}}\ \circ\wedge_1 \qquad \frac{\mathcal{A}\{A \circ B\}}{\mathcal{A}\{A \circ (F \wedge B)\}}\ \circ\wedge_2$$

$$\frac{\mathcal{A}\{(A \triangleright B) \supset F\}}{\mathcal{A}\{A \circ (B \supset F)\}}\ \circ\supset_1 \qquad \frac{\mathcal{A}\{F \supset (A \circ B)\}}{\mathcal{A}\{A \circ (F \supset B)\}}\ \circ\supset_2$$

(plus all the symmetric variants)

Figure 1. Inference rules for interaction formulas

We will define an inference system for interaction formulas that consist of inference rules with a single conclusion and a single premise, both of which are either formulas or interaction formulas. The inference rule represents an admissible rule of intuitionistic logic: if the premise is a theorem, then so is the conclusion. The full collection of rules is shown in fig. 1. There are three kinds of rules, explained below in an upwards (conclusion to premises) reading.

- *Terminal rules* are used to terminate a $\triangleright$-interaction in a positively signed context. In the case where the $\triangleright$-interaction links two occurrences of the same atom, the result is $\top$; otherwise the $\triangleright$ turns back into $\supset$. These are the only rules that can transition out of interaction formulas.

- *Positively signed rules* operate on a $\triangleright$-interaction in a positively signed context. The rules are written in fig. 1 in such a way that the subformulas A and B are brought together in the premise, and occurrences of F (if they exist) are side formulas.

- *Negatively signed rules* operate on a $\circ$-interaction in an negatively signed context. Fig. 1 only shows one of the two symmetric variants for each case; the other variant is built by permuting A with B and transposing the operands of $\circ$. For instance, $\circ\vee_1$ has the following symmetric variant.

$$\frac{\mathcal{A}\{(A \circ B) \vee F\}}{\mathcal{A}\{(A \vee F) \circ B\}}\ \circ\vee_{1'}$$

We will use primes to systematically name the symmetric variants of rules.

Proposition 2 (Soundness)**.** *Interpreting $\triangleright$ as $\supset$ and $\circ$ as $\wedge$, each rule of fig. 1 with premise P and conclusion Q has the property that $P \vdash Q$.*

Interaction creation rules

$$\frac{\mathcal{C}\{A \triangleright B\}}{\mathcal{C}\{A \supset B\}}\ \triangleright \qquad \frac{\mathcal{A}\{A \circ B\}}{\mathcal{A}\{A \wedge B\}}\ \circ$$

Contraction

$$\frac{\mathcal{C}\{A \supset A \supset F\}}{\mathcal{C}\{A \supset F\}}\ \mathsf{cont}$$

Simplification rules

$$\frac{\mathcal{C}\{\top\}}{\mathcal{C}\{A \supset \top\}} \qquad \frac{\mathcal{C}\{B\}}{\mathcal{C}\{\top \supset B\}} \qquad \frac{\mathcal{C}\{\top\}}{\mathcal{C}\{\bot \supset B\}}$$

$$\frac{\mathcal{C}\{F\}}{\mathcal{C}\{\top \wedge F\}} \quad \frac{\mathcal{C}\{F\}}{\mathcal{C}\{F \wedge \top\}} \qquad \frac{\mathcal{A}\{F\}}{\mathcal{A}\{\bot \vee F\}} \quad \frac{\mathcal{A}\{F\}}{\mathcal{A}\{F \vee \bot\}}$$

$$\frac{\mathcal{A}\{\bot\}}{\mathcal{A}\{\bot \wedge F\}} \quad \frac{\mathcal{A}\{\bot\}}{\mathcal{A}\{F \wedge \bot\}} \qquad \frac{\mathcal{C}\{\top\}}{\mathcal{C}\{\top \vee F\}} \quad \frac{\mathcal{C}\{\top\}}{\mathcal{C}\{F \vee \top\}}$$

Figure 2. Link creation, contraction, and simplification. The conclusion in each case must not be an interaction formula.

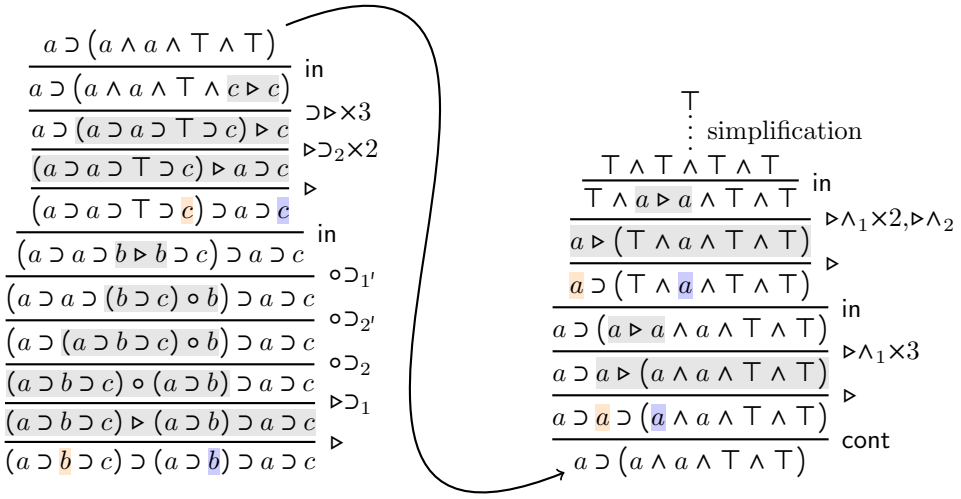


Figure 3. Lnip derivation fragment for the S-combinator

Proof. Straightforward consequence of theorem 1. □ □

Two further administrative steps remain to complete the technique. First, since the rules of fig. 1 always contain an interaction formula in the conclusion, we need to add some rules that can conclude ordinary (non-interaction) formulas. Since we read each inference rule from conclusion to premise, we will call these the *interaction creation* rules, which are shown in the first part of fig. 2. To incorporate non-linearity, we add a separate contraction rule; this keeps the interaction creation rules simple, but it needs to be explicitly invoked. These interaction creation rules are obviously sound under the interpretation of proposition 2.

The final step is to detect when a proof is complete. Since every inference rule presented so far has a single premise, we will say that a proof is complete when the final (again reading bottom to top) premise is, effectively, $\top$. What do we mean by "effectively"? One candidate definition could be that a purely algorithmic procedure can detect when a proof is finished in linear time. For instance, we can say that a proof is complete if its premise can be established using only the *simplification rules* shown in the second part of fig. 2. These rules may be applied in any arbitrary order and at any time. An implementation of the technique may choose to apply these simplification rules on the fly.

Definition 3. *The collection of rules in figures 1 and 2 will be known as the proof system* Lnip*. If A and B are formulas or interaction formulas, we write $A \xrightarrow{\mathsf{Lnip}} B$ to mean that either $A = B$ or there is an* Lnip *derivation where the topmost rule has premise A and the bottom-most rule has conclusion B.* □

Theorem 4 (Completeness of Lnip)**.** *If $\vdash F$, then $\top \xrightarrow{\mathsf{Lnip}} F$.*

Sketch. There are many ways to prove this, both syntactic and semantic. An instructive syntactic proof goes as follows. For a small variant of the G3ip sequent calculus [11], we show that every inference rule is admissible in Lnip under a suitable formula interpretation of sequents. Thus, any sequent proof is recoverable in terms of Lnip inferences. We then just appeal to completeness of the sequent calculus. □ □

Example 5. *A* Lnip *derivation of the S-combinator formula,* $(a \supset b \supset c) \supset (a \supset b) \supset a \supset c$*, is shown in fig. 3. The interaction connectives* $\triangleright$ *and* $\circ$ *take the precedence and associativity of* $\supset$ *and* $\wedge$ *respectively. The locus where a* Lnip *rule is applied is depicted with a highlight. Of course, the S-combinator formula cannot be proved without appealing to contraction at least once, which is seen by the appeal to* cont *in the derivation.*

An extremely interesting aspect of this example Lnip derivation is that it begins by considering the first two assumptions, $(a \supset b \supset c)$ and $(a \supset b)$, of the S-combinator formula. The user might have indicated this consideration by drawing a *link* between the two occurrences of b, highlighted in orange and blue in fig. 3. The effect of this consideration is to perform a "composition" of the two assumptions into the stronger assumption $(a \supset a \supset \top \supset c)$, which could of course have been simplified to $(a \supset a \supset c)$ immediately. In shallow proof systems such as the sequent calculus or natural deduction this kind of compositional step cannot be taken as such, and would require cuts or lemmas.

As explained in the introduction, this kind of composition might have been discovered in the process of exploration by the simple strategy of drawing a *link* between the two occurrences of b. Such a link is legal because in the common context that contains both occurrences of b, their ancestral connective is $\supset$, which can be turned into a $\triangleright$ interaction using the $\triangleright$ rule. Once these two occurrences are linked, we can interpret the interaction rules (fig. 1) as trying to bring the two ends of the link closer. Indeed, in each of the rules of fig. 1, we can say that one of the ends of the link is in the formula A and the other is in the formula B. We are therefore ready to formulate the linking procedure.

Definition 6 (Subformula Linking Procedure)**.** *Repeat the following sequence of steps until the conjecture formula (i.e., end-formula)* F *is transformed to* $\top$ *(success), no fruitful progress can be made (failure), or the proof attempt is aborted by the user.*

1. *(Optional) Ask the user to indicate negatively signed subformulas of* F *that need to be contracted using the* cont *rule.*
2. *Ask the user to indicate two different subformulas of* F*; this is the* link*.*
3. *If the first common ancestor connective of the two linked subformulas is a* $\supset$ *that occurs in a positively signed context, use the* $\triangleright$ *rule to turn it into a* $\triangleright$*; likewise, if the ancestor is a* $\wedge$ *in an negatively signed context, use the* $\circ$ *rule to turn it into a* $\circ$*. If neither case applies, then the user indicated an invalid link, so we return immediately to step 2.*
4. *Use the interaction rules (fig. 1) in such a way that the endpoints of the link stay in the same interaction from conclusion to premise.*
5. *Eventually, one of the terminal rules* in *or* rel *will be applicable to remove the interaction; at this point we say that the link is* resolved*.*
6. *After resolving a link, the simplification rules may be applied eagerly in an arbitrary order.*

The most important step in the inner loop of the procedure is step 4. The rules for interaction are not unambiguous because the conclusions of different rules can overlap. Let us start by examining the positively signed rules; as an example, consider the interaction $\mathcal{C}\{(F \supset A) \triangleright (G \supset B)\}$, with the understanding that the endpoints of the indicated link in step 2 are present in A and B. There are two possible ways to resolve this link:

$$\dfrac{\dfrac{\mathcal{C}\{F \wedge (G \supset (A \triangleright B))\}}{\mathcal{C}\{F \wedge (A \triangleright (G \supset B))\}}\ \triangleright\supset_2}{\mathcal{C}\{(F \supset A) \triangleright (G \supset B)\}}\ \supset\triangleright \qquad \dfrac{\dfrac{\mathcal{C}\{G \supset (F \wedge (A \triangleright B))\}}{\mathcal{C}\{G \supset ((F \supset A) \triangleright B)\}}\ \supset\triangleright}{\mathcal{C}\{(F \supset A) \triangleright (G \supset B)\}}\ \triangleright\supset_2$$

Does the choice matter? Yes, because the formulas $F \wedge (G \supset H)$ and $G \supset (F \wedge H)$ are not intuitionistically equivalent; indeed, the former strictly entails the latter. Hence, one of the two alternatives produces a strictly stronger—and potentially unprovable!—premise. Which one should the procedure pick?

This ambiguity also existed in the original formulation of the formula linking procedure for classical linear logic [4], and we can use the same answer used in that work. The key insight is that many of the ambiguous cases can be resolved by a simple analysis of *polarities*. A detailed discussion of polarity (and the oft-associated *focusing* discipline [1]) is not relevant to this work, however.[2] We will instead just use the observation that some of the interaction rules of fig. 1 are *asynchronous*, meaning that the premise of the rule is equiderivable as the conclusion—assuming we replace $\triangleright$ and $\circ$ with $\supset$ and $\wedge$ respectively—while other rules are *synchronous*, which means that the premise strictly entails the conclusion. For the specific example above, the $\triangleright\supset_2$ rule is asynchronous, because the order of assumptions in an implication is immaterial (at least in intuitionistic logic),

[2] Our choice of connectives here has only negative polarity connectives except $\exists$ and $\vee$. In intuitionistic logic it is also possible to have a positive $\wedge$ and atoms of both polarities [5, 10], but this generality is not necessary for the present work.

Terminal rules

$$\frac{\mathcal{C}\{\vec{s} \doteq \vec{t}\}}{\mathcal{C}\{a{\cdot}\vec{s} \triangleright a{\cdot}\vec{t}\}}\ \mathsf{in}$$

Quantifier rules

$$\frac{\mathcal{C}\{\forall x.\,(A \triangleright B)\}}{\mathcal{C}\{A \triangleright \forall x.\,B\}}\ \triangleright\forall \qquad \frac{\mathcal{C}\{\exists x.\,(A \triangleright B)\}}{\mathcal{C}\{(\forall x.\,A) \triangleright B\}}\ \forall\triangleright \qquad \frac{\mathcal{A}\{\forall y.\,(A \circ B)\}}{\mathcal{A}\{A \circ \forall y.\,B\}}\ \circ\forall$$

$$\frac{\mathcal{C}\{\exists y.\,(A \triangleright B)\}}{\mathcal{C}\{A \triangleright \exists y.\,B\}}\ \triangleright\exists \qquad \frac{\mathcal{C}\{\forall x.\,(A \triangleright B)\}}{\mathcal{C}\{(\exists x.\,A) \triangleright B\}}\ \exists\triangleright \qquad \frac{\mathcal{A}\{\exists y.\,(A \circ B)\}}{\mathcal{A}\{A \circ \exists y.\,B\}}\ \circ\exists$$

(in each rule, $x\#B$ and $y\#A$)

Simplification and instantiation rules

$$\frac{\mathcal{C}\{\top\}}{\mathcal{C}\{\forall x.\,\top\}} \qquad \frac{\mathcal{C}\{\top\}}{\mathcal{C}\{x \doteq x\}}\ \mathsf{refl} \qquad \frac{\mathcal{C}\{\vec{s} \doteq \vec{t}\}}{\mathcal{C}\{\mathsf{f}{\cdot}\vec{s} \doteq \mathsf{f}{\cdot}\vec{t}\}}\ \mathsf{cong} \qquad \frac{\mathcal{C}\{t\ \mathrm{term}\} \quad \mathcal{C}\{[t/x]A\}}{\mathcal{C}\{\exists x.\,A\}}\ \mathsf{inst}$$

Figure 4. System Lni: rules for quantifiers and terms

while the $\supset\triangleright$ rule is synchronous since its conclusion cannot justify the premise. We can draw up this table for all the positively signed rules.

asynchronous rules:	$\triangleright\wedge_1$, $\triangleright\wedge_2$, $\vee\triangleright_1$, $\vee\triangleright_2$, $\triangleright\supset_1$, $\triangleright\supset_2$
synchronous rules:	$\triangleright\vee_1$, $\triangleright\vee_2$, $\wedge\triangleright_1$, $\wedge\triangleright_2$, $\supset\triangleright$

Whenever there is a choice between a synchronous and an asynchronous rule to apply first (reading from bottom to top), we should pick the asynchronous rule, since that does not destroy derivability. If we have a choice of two asynchronous rules, then the choice is immaterial, as derivability is preserved regardless; the procedure can pick arbitrarily. Different choices would just lead to associative-commutative variants of the same ultimate premise. Finally, for a choice between two synchronous rules, we can consider all such pairs from the table above to see that the choice is immaterial: all choices have the same result.

The story is not quite as simple for the negatively signed rules of fig. 1, where every single rule would be synchronous by our definition. Unlike in the positively signed case, here we have a critical pair.

$$\frac{\dfrac{\mathcal{A}\{(F \supset (A \circ B)) \vee G\}}{\mathcal{A}\{((F \supset A) \circ B) \vee G\}}\ \circ\supset_{2'}}{\mathcal{A}\{(F \supset A) \circ (B \vee G)\}}\ \circ\vee_1 \qquad \frac{\dfrac{\mathcal{A}\{F \supset (A \circ B) \vee G)\}}{\mathcal{A}\{F \supset (A \circ (B \vee G))\}}\ \circ\vee_1}{\mathcal{A}\{(F \supset A) \circ (B \vee G)\}}\ \circ\supset_{2'}$$

As before, the premises are not equiderivable. Resolving this ambiguity is going to be as hard as fully automated proof search, which will therefore not be recursively solvable as soon as we introduce quantifiers. The subformula linking procedure needs further guidance from the user to resolve the ambiguity. A variant of this ambiguity can also be found in the original subformula linking work for classical linear logic [4]; there, the solution was to make the links *directed*. Then, whenever there is a choice to be made—which will necessarily have to be a choice between one subformula containing the *source* of the link and the other containing the *destination*—the procedure can choose to perform the rule corresponding to the *destination first*. In the above critical pair, for instance, if A contained the source and B the destination, then we would perform the $\circ\vee_1$ step first (i.e., follow the left derivation). This choice is made to evoke the intuition that *the source is brought to the destination*; the context of the destination swallows the context of the source.

Definition 7 (Directed Subformula Linking Procedure)**.** *We modify the procedure of definition 6 by making the links in step 2 directed, and in the resolution step 4 we break synchronous/synchronous ties for negatively signed rules by performing the rule for the destination first.*

2.2 Quantifiers

Extending Lnip with first-order quantifiers can be done in a number of ways. Here we present a parsimonious extension that avoids any up front commitments with regard to the strength of the term language. Our terms (written $s, t, \dots$) have the following grammar:

$$s, t, \dots ::= x \mid \mathsf{f}{\cdot}\vec{s}$$

where we write $\vec{s}$ to stand for a list of terms $[s_1, s_2, \dots, s_n]$. We use $x, y, \dots$ to range over variables and $\mathsf{f}, \mathsf{g}, \dots$ to range over function symbols, and we abbreviate $\mathsf{f}{\cdot}[\,]$ to f. We also extend atomic formulas: they are now

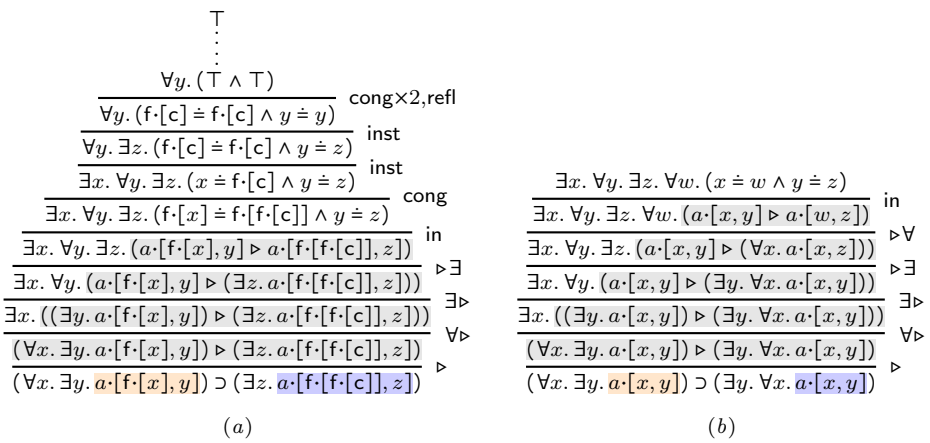


Figure 5. Two example Lni derivations

written $a{\cdot}\vec{s}$ where a is a predicate symbol, and we again abbreviate $a{\cdot}[\,]$ to a. To formulas and contexts we now add the two quantifiers, $\forall$ and $\exists$, to give the following extended grammars, where $* \in \{\wedge, \vee\}$ and $Q \in \{\forall, \exists\}$.

$$\begin{aligned} A, B, \ldots &::= a{\cdot}\vec{s} \mid A \wedge B \mid \top \mid A \vee B \mid \bot \mid A \supset B \mid \forall x.\, A \mid \exists x.\, A \\ \mathcal{C}\{\} &::= \{\} \mid A * \mathcal{C}\{\} \mid \mathcal{C}\{\} * B \mid Qx.\, \mathcal{C}\{\} \mid A \supset \mathcal{C}\{\} \mid \mathcal{A}\{\} \supset B \\ \mathcal{A}\{\} &::= A * \mathcal{A}\{\} \mid \mathcal{A}\{\} * B \mid Qx.\, \mathcal{A}\{\} \mid A \supset \mathcal{A}\{\} \mid \mathcal{C}\{\} \supset B \end{aligned}$$

We write $\mathcal{C}\{t \text{ term}\}$ to assert that the term t is well-formed for the hole in $\mathcal{C}\{\}$, i.e., all the (free) variables of t are bound by some quantifier that the hole in $\mathcal{C}\{\}$ is in the scope of. We also write $x\#t$ or $x\#A$ to indicate that the variable x is not free in t or A respectively. Finally, the capture-avoiding substitution of t for x in a term u or formula A is written $[t/x]u$ or $[t/x]A$ respectively. The replacement of formulas in contexts, on the other hand, is not capture-avoiding $\mathcal{C}\{A\}$; instead, this replacement is considered to be well-formed whenever every free variable x of A has the property that $\mathcal{C}\{x \text{ term}\}$.

In order to give ourselves maximum freedom in the definition of the first-order extension, we will use the additional binary predicate symbol $\doteq$ to denote equality. Given two lists of terms $\vec{s} = [s_1, \ldots, s_n]$ and $\vec{t} = [t_1, \ldots, t_n]$ of equal length, we will write $\vec{s} \doteq \vec{t}$ to stand for $(s_1 \doteq t_1) \wedge \cdots \wedge (s_n \doteq t_n)$ if $n > 0$ and for $\top$ otherwise. Using this additional predicate, the terminal rule in of Lnip is modified to account for the term arguments.

Definition 8 (System Lni)**.** *The system Lni is an extension of Lnip by removing the in rule of Lnip and adding the rules of fig. 4.*

Theorem 9 (Completeness of Lni)**.** *If $\vdash F$ in a complete sequent calculus for first-order intuitionistic logic (e.g., G3i [11]) then $\top \xrightarrow{Lni} F$.*

Sketch. We can follow the same strategy as for theorem 4. Note that for any term t, the rules refl and cong suffice to reduce $\mathcal{C}\{t \doteq t\}$ to $\mathcal{C}\{\top\}$. A transitivity rule for $\doteq$ is not needed: no $\doteq$ is created in an negatively signed context. □ □

Example 10. *Two example Lni derivations are shown in fig. 5.*

(a) *This is a derivation for a provable formula where the user may have linked the two occurrences of a. Observe that the simplification rules {cong, inst, refl} help to implement first-order unification under a mixed quantifier prefix. However, since Lni simplification rules can be applied at any time, we can solve unification problems incrementally, in tandem with logical reasoning.*

(b) *This is a derivation for an unprovable formula containing an illegal quantifier exchange, where once again the indicated link is between the two occurrences of a. This derivation cannot be completed because there is no instantiation for x for which $\forall w.\, x \doteq w$ is true.*

3 Incorporating Arity-Typed λ-Terms

To make the calculus Lni of the previous section suitable to host a type theory as an object language, we will need to generalize from first-order terms to general λ-terms. We will follow a standard technique known variously as *higher-order abstract syntax* (HOAS) [12] or *λ-tree syntax* [7] that treats the *pure* λ-calculus—together with

$\alpha\beta\eta$-equality as its equational theory—to represent object languages. To keep things computable, we will use simply typed λ-terms with only one basic type, which is sometimes known as *arity typing*. Arity types $(\alpha, \beta, \dots)$ and terms $(s, t, \dots)$ have the following grammar.

$$\alpha, \beta, \dots ::= \star \mid \alpha \to \beta \qquad h ::= x \mid \mathsf{k} \qquad s, t, \dots ::= h{\cdot}\vec{s} \mid \lambda x{:}\alpha.\, t$$

where $x, y, \dots$ range over variables, and sans-serif identifiers such as k range over term constants. For formulas, we also change the quantifiers $Qx.\, F$ to their arity typed forms $Qx{:}\alpha.\, F$, where $Q \in \{\forall, \exists\}$.

We keep λ-terms in canonical *spine form*, where the head (h) of an application is identified and separated; in more usual notation, $h{\cdot}[s_1, \dots, s_n]$ would be written as the iterated application $(\cdots(h\ s_1) \cdots\ s_n)$. The definition of substitution, $[t/x]s$, must be modified to retain spine forms, which is usually done by removing redexes on the fly; for example (using @ as an auxiliary operation):

$$[t/x]\mathsf{k} = \mathsf{k}{\cdot}[] \qquad [t/x]x = t \qquad [t/x]y = y{\cdot}[] \qquad \text{(where } x \text{ and } y \text{ are different)}$$

$$\begin{aligned}
[t/x](\lambda y{:}\alpha.\, s) &= \lambda y{:}\alpha.\, [t/x]s \\
[t/x](h{\cdot}[s_1, \dots, s_n]) &= ([t/x]h) \mathbin{@} [[t/x]s_1, \dots, [t/x]s_n] \\
(\lambda x{:}\alpha.\, s) \mathbin{@} [t_1, t_2, \dots, t_n] &= ([t_1/x]s) \mathbin{@} [t_2, \dots, t_n] \\
(h{\cdot}[s_1, \dots, s_m]) \mathbin{@} [t_1, \dots, t_n] &= h{\cdot}[s_1, \dots, s_m, t_1, \dots, t_n]
\end{aligned}$$

Most of the inference rules of system Lni generalize easily to this setting. The immediate differences will be with respect to the simplification rules. For the inst rule, we use a variant judgement $\mathcal{C}\{t : \alpha\}$ to mean that the λ-term t is well-typed at type α based on the type assumptions of its free variables that are bound in the scope of the hole in $\mathcal{C}\{\}$. It is possible to view this judgement as being defined by inference rules; for instance (for $Q \in \{\forall, \exists\}$):

$$\frac{}{\mathcal{C}\{Qx{:}\alpha.\, \mathcal{C}'\{x : \alpha\}\}} \qquad \frac{\mathcal{C}\{\forall x{:}\alpha.\, (t : \beta)\}}{\mathcal{C}\{(\lambda x{:}\alpha.\, t) : \alpha \to \beta\}}$$

$$\frac{\mathcal{C}\{h : \alpha_1 \to \cdots \to \alpha_n \to \beta\} \quad \mathcal{C}\{s_i : \alpha_i\}}{\mathcal{C}\{(h{\cdot}[s_1, \dots, s_n]) : \beta\}}$$

The rules refl and cong of Lni are replaced with:

$$\frac{\mathcal{C}\{\vec{s} \doteq \vec{t}\}}{\mathcal{C}\{h{\cdot}\vec{s} \doteq h{\cdot}\vec{t}\}}\ \mathsf{cong} \qquad \frac{\mathcal{C}\{\forall x{:}\alpha.\, (s \doteq t)\}}{\mathcal{C}\{(\lambda x{:}\alpha.\, s) \doteq (\lambda x{:}\alpha.\, t)\}}\ \mathsf{abs}$$

$$\frac{\mathcal{C}\{(\lambda x{:}\alpha.\, h{\cdot}[s_1, \dots, s_n, x]) \doteq (\lambda x{:}\alpha.\, t)\}}{\mathcal{C}\{h{\cdot}[s_1, \dots, s_n] \doteq (\lambda x{:}\alpha.\, t)\}}\ \eta\text{-}\mathsf{exp} \qquad \text{(and its symm. variant)}$$

Definition 11 (System $\mathsf{Lni}\lambda$)**.** *The system $\mathsf{Lni}\lambda$ is a modification of Lni with the ∇ rules, cong, abs, η-exp, and in above.*

Theorem 12 (Completeness of $\mathsf{Lni}\lambda$)**.** *For any formula F in the language of first-order logic over λ-terms but without any occurrence of $\doteq$, if $\vdash F$ in a complete sequent calculus then $\top \xrightarrow{\mathsf{Lni}\lambda} F$.*

Sketch. Once again, this is a straightforward extension of the proof of theorem 9. Since there are no occurrences of $\doteq$ in F, and in particular no occurrence of it in a negatively signed context, the rules cong, abs and η-exp are sufficient to implement $\alpha\beta\eta$-equivalence. □ □

4 Application: Embedding Intuitionistic Type Theories

The first-order language over arity-typed λ-terms of the previous section has enough expressive power for a complete encoding of any pure type system [6, 15]. To keep things simple in this paper, we will demonstrate the case for LF (aka $\lambda\Pi$) using the *simple* embedding from [15]. Expressions in LF belong to one of the following three syntactic categories: *kinds*, *types*, or *terms*.

$$K ::= \mathsf{type} \mid \Pi x{:}A.\, K \qquad \text{(kinds)}$$

$$A, B, \dots ::= \mathsf{a}\ M_1 \cdots M_n \mid \Pi x{:}A.\, B \qquad \text{(types)}$$

$$M, N, \dots ::= x \mid \mathsf{k} \mid \lambda x{:}A.\, M \mid M\ N \qquad \text{(terms)}$$

The LF type system is formally specified using inference rules in [9] and will not be repeated here. Instead, we will directly present a complete encoding of LF expressions using the language of $\mathsf{Lni}\lambda$.

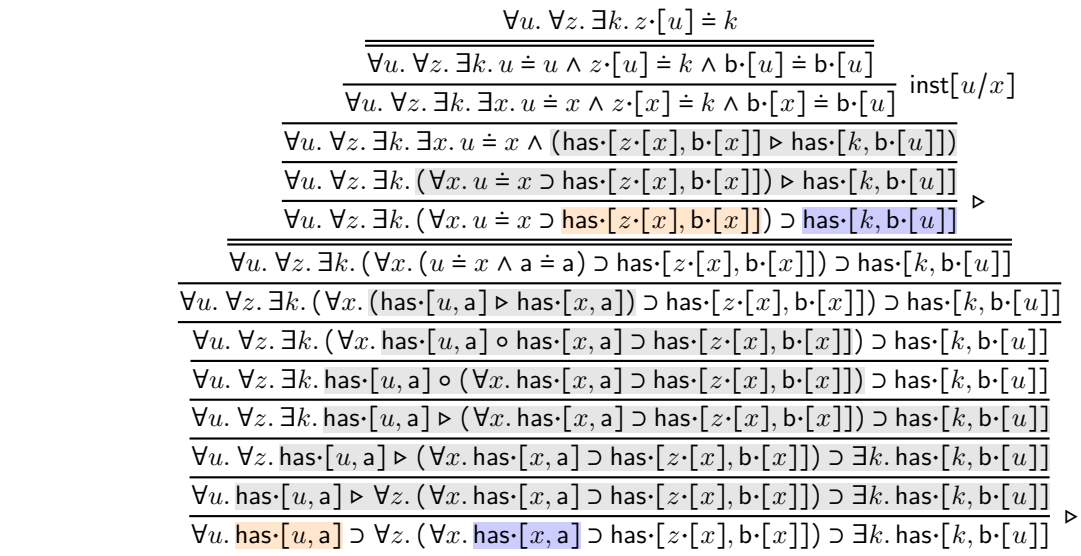


Figure 6. A Lniλ derivation of an embedded LF type (example 16). Some type ascriptions are elided, and doubled lines denote simplifications.

The encoding proceeds in two steps. First, we transform the dependently typed terms of LF into their simply typed forms, normalizing them as necessary. However, since LF terms can mention their types, we simultaneously transform LF types into simple types. This transformation erases not just the type dependencies but also the identities of the types by collapsing all of them to the same base type $\star$.

Definition 13. *The* forgetful map ϕ *specified below transforms LF terms into Lniλ λ-terms and LF types and kinds into Lniλ types.*

$$\begin{aligned} \phi(\mathsf{k}) &= \mathsf{k}{\cdot}[] & \qquad \phi(\mathsf{a}\ M_1 \cdots M_n) &= \star \\ \phi(x) &= x{\cdot}[] & \qquad \phi(\Pi x{:}A.\, B) &= \phi(A) \to \phi(B) \\ \phi(\lambda x{:}A.\, M) &= \lambda x{:}\phi(A).\, \phi(M) & \qquad \phi(\mathsf{type}) &= \star \\ \phi(M\ N) &= \phi(M) \mathbin{@} [\phi(N)] & \qquad \phi(\Pi x{:}A.\, K) &= \phi(A) \to \phi(K) \end{aligned}$$

The second stage of the transformation recovers the information that was lost in the ϕ map by means of one atomic propositions, has. Using this we define a mapping $[\![\,]\!]$ that transforms types and kinds to formulas in such a way that if $M : A$ holds then $[\![A]\!]\phi(M)$ is true.

Definition 14. *The mapping* $[\![\,]\!]$ *transforms an LF type/kind and a Lniλ λ-terms into a Lniλ formula, specified recursively as follows.*

$$\begin{aligned} [\![\mathsf{a}\ M_1 \cdots M_n]\!]m &= \mathsf{has}{\cdot}[m, \mathsf{a}{\cdot}[\phi(M_1), \ldots, \phi(M_n)]] \\ [\![\mathsf{type}]\!]m &= \mathsf{has}{\cdot}[m, \mathsf{type}] \\ [\![\Pi x{:}A.\, J]\!]m &= \forall x{:}\phi(A).\, [\![A]\!]x \supset [\![J]\!](m \mathbin{@} [x]) \end{aligned}$$

(where J can be a LF type or kind).

Proposition 15 (Completeness [15])**.** *If the judgement $x_1{:}J_1, \ldots, x_n{:}J_n \vdash M : A$ is derivable in LF [9], then the following formula is provable in Lniλ: $\forall x_1{:}\phi(J_1).\, [\![J_1]\!](x_1{\cdot}[]) \supset \cdots \supset \forall x_n{:}\phi(J_n).\, [\![J_n]\!](x_n{\cdot}[]) \supset [\![A]\!]\phi(M)$.* □

The converse of proposition 15 does not necessarily hold, since the forgetful map ϕ is injective, not surjective.[3] In particular, since the encoding of atomic types forgets the term arguments, we have that $\phi(\lambda x{:}A_1.\, s) = \phi(\lambda x{:}A_2.\, s)$ if $\phi(A_1) = \phi(A_2)$; however, the latter does not guarantee that $A_1 = A_2$. Thus, $[\![\Pi x{:}A_1.\, B]\!]\phi(\lambda x{:}A_2.\, s)$ may hold even when $A_1 \neq A_2$. To guarantee surjectivity, we must use the *canonical LF* variant of the LF type theory where the type ascription on λ is omitted and the type system is made bidirectional [19]; this will guarantee that only Π-types will ascribe types to bound variables, removing the issue highlighted above.

[3] This issue, pointed out in [16], is a mistake in earlier papers such as [6, 15].

Example 16. *Consider the following LF type* $A \triangleq \Pi u{:}\mathsf{a}.\,\Pi z{:}(\Pi x{:}\mathsf{a}.\,\mathsf{b}\,x).\,\mathsf{b}\,u$. *By definition 14, we have:*

$$\begin{aligned}[\![A]\!]k = {} & \forall u{:}\star.\,\mathsf{has}{\cdot}[u,\mathsf{a}] \supset \\ & \forall z{:}\star \to \star.\,(\forall x{:}\star.\,\mathsf{has}{\cdot}[x,\mathsf{a}] \supset \mathsf{has}{\cdot}[z{\cdot}[x],\mathsf{b}{\cdot}[x]]) \supset \\ & \mathsf{has}{\cdot}[k,\mathsf{b}{\cdot}[u]].\end{aligned}$$

Fig. 6 has an example $\mathsf{Lni}\lambda$ *derivation of this formula where* k *is existentially quantified. As usual, highlights are used to indicate the two links the user indicated in the two* $\triangleright$ *rules. The derivation can be complete with the instantiation* $[z{\cdot}[u]/k]$*; this means that the LF type* A *is inhabited by some LF term* M *for which* $\phi(M) = z{\cdot}[u]$.

Note that the fact that we have not discovered a LF term for k using the $\mathsf{Lni}\lambda$ derivation is not a problem. Given a $\mathsf{Lni}\lambda$ term k for which $[\![A]\!]k$ is derivable, it is possible to find a term M for which $\phi(M) = k$ and $M : A$ holds in LF. One way to do this would be to use *bidirectional type checking* [14, 19] to recreate—deterministically—the missing LF types.

While the encoding of LF in $\mathsf{Lni}\lambda$ suffices to implement the proof by linking technique, it is a leaky encoding. As the derivation in fig. 6 proceeds, the conjecture resembles the image of the $[\![\,]\!]$ map less and less; in particular, the conjecture starts to accumulate things that are not fundamentally present in the LF type system, such as term equations, conjunctions, and existential quantifiers. The purported novice user mentioned in the introduction thus needs to be familiar with at least two languages: LF and (a somewhat esoteric variant of) first-order logic. One way to improve matters would be to try to define the linking procedure directly on the LF type system, but this example seems to indicate that the LF language is not expressive enough to capture all the structures that will occur when resolving a link. At the very least, it seems that some kind of pairing construct—i.e., Σ-types—is essential. Moreover, to capture free floating has assumptions, the language of LF might need to be extended further with judgemental expressions of the form $\langle M{:}A\rangle$.

5 Conclusion and Future Directions

We have presented a formal system of *proof by linking* for intuitionistic logic and a derived system for the dependent type theory LF. We are currently in the process of implementing this system as a variant of the *Profound* tool, which was initially developed for classical linear logic in [4].

In order for this system to be usable in a general purpose interactive theorem prover based on first-order logic (such as Abella [2]) or dependent type theory (such as Twelf [13]), the most important missing ingredient is support for inductive definitions and reasoning by induction. The first step in a proof by structural induction is to indicate which assumption(s) will drive the analysis, which is closer to a *pointing* than a *linking*. Thus, proof by linking and pointing will need to co-exist.

A further improvement that would be made as a matter of course in an implementation would be the use of a unification engine to remove the clutter of $\doteq$ formulas. It is worth investigating (in future work) if the linking metaphor can also be used for algebraic operations on terms based on $\doteq$. In many systems $\doteq$-assumptions can be used to rewrite terms, which is readily incorporated into the linking scheme: just link a term to one side of a $\doteq$. We can in fact see it as variants of the inst rule:

$$\frac{\mathcal{C}\{[t/x]\mathcal{C}'\{\top\}\}}{\mathcal{C}\{\exists x.\,\mathcal{C}'\{x \doteq t\}\}} \qquad \frac{\mathcal{A}\{[t/x]\mathcal{A}'\{\top\}\}}{\mathcal{A}\{\forall x.\,\mathcal{A}'\{x \doteq t\}\}}$$

It is worth investigating if such variants of inst can make the embedding of LF into $\mathsf{Lni}\lambda$ less leaky.

Note that proof by linking, like proof by pointing, can easily be incorporated as a tactic in an existing proof system. After all, each of the inference rules of $\mathsf{Lni}\lambda$ is logically motivated, and can therefore be established as a certifying tactic. The quality of the formal proof terms produced in this way will be poor since most proof term languages are not designed for deep rewriting – indeed, the proof term for each $\mathsf{Lni}\lambda$ inference rule may have a size that is exponential in that of the conjecture. It is perhaps better to see proof by linking as a *proof exploration* tool for quickly testing out logical properties of a conjecture before attempting a traditional structured proof. In the hands of an expert user, this exploration mode can also help to discover useful lemmas to bridge the gap between an existing collection of proved theorems and a desired target theorem.

References

[1] J.-M. Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[2] D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2), 2014.

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

[4] K. Chaudhuri. Subformula linking as an interaction method. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *4th Conference on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 386–401. Springer, July 2013.

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

[6] A. Felty and D. Miller. Encoding a dependent-type λ-calculus in a logic programming language. In *CADE*, volume 449 of *LNAI*, pages 221–235. Springer, 1990.

[7] A. Gacek, D. Miller, and G. Nadathur. A two-level logic approach to reasoning about computations. *Journal of Automated Reasoning*, 49(2):241–273, 2012.

[8] N. Guenot. *Nested Deduction in Logical Foundations for Computation*. Ph.d. thesis, Ecole Polytechnique, 2013.

[9] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

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

[11] S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[12] F. Pfenning and C. Elliott. Higher-order abstract syntax. In *ACM-SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 199–208. ACM Press, June 1988.

[13] F. Pfenning and C. Schürmann. System description: Twelf — A meta-logical framework for deductive systems. In H. Ganzinger, editor, *16th International Conference on Automated Deduction (CADE)*, number 1632 in LNAI, pages 202–206, Trento, 1999. Springer.

[14] B. C. Pierce and D. N. Turner. Local type inference. *ACM Transactions of Programming Language Systems*, 22(1):1–44, 2000.

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

[16] M. Southern. *A Framework for Reasoning about LF Specifications*. PhD thesis, University of Minnesota, Mar. 2021. Defended; final version to appear.

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

[18] A. Tiu. A local system for intuitionistic logic. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4246 of *LNCS*, pages 242–256. Springer, 2006.

[19] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework I: The propositional fragment. In *Post-proceedings of TYPES 2003 Workshop*, number 3085 in LNCS. Springer, 2003.