A Drag-and-Drop Proof Tactic

Pablo Donato
pablo.donato@polytechnique.edu
École polytechnique
LIX
France

Pierre-Yves Strub
pierre-yves.strub@polytechnique.edu
École polytechnique
LIX
France

Benjamin Werner
benjamin.werner@polytechnique.edu
École polytechnique
LIX
France

Abstract

We explore the features of a user interface where formal proofs can be built through gestural actions. In particular, we show how proof construction steps can be associated to drag-and-drop actions. We argue that this can provide quick and intuitive proof construction steps. This work builds on theoretical tools coming from deep inference. It also resumes and integrates some ideas of the former proof-by-pointing project.

CCS Concepts: • **Mathematics of computing** → *Mathematical software*; • **Human-centered computing** → *Graphical user interfaces*; *Gestural input*; • **Theory of computation** → *Proof theory*; *Equational logic and rewriting*; *Constructive mathematics*.

Keywords: logic, formal proofs, user interfaces, deep inference

ACM Reference Format:
Pablo Donato, Pierre-Yves Strub, and Benjamin Werner. 2022. A Drag-and-Drop Proof Tactic. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '22), January 17–18, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3497775.3503692

1 Introduction

Most Interactive Theorem Provers allow the user to incrementally construct formal proofs through an interaction loop. One progresses through a sequence of *states* corresponding to incomplete proofs. Each of these states is itself described by a finite set of *goals* and the proof is completed once there are no goals left. From the user's point of view, a goal appears as a sequent, in the sense coined by Gentzen. In the case of intuitionistic logic that is:

- One particular proposition A which is *to be proved*; we designate it as the goal's *conclusion*,
- a set of propositions Γ corresponding to *hypotheses*.

On paper, this sequent is written $\Gamma \vdash A$. The user performs *actions* on one such goal at a time, and the actions transform the goal, or rather replace the goal by a new set of goals. When this set is empty, the goal is said to be solved.

The actions performed by the user can be more or less sophisticated. But, fundamentally, one finds elementary commands which correspond roughly to the logical rules, generally of natural deduction. For instance, a goal $\Gamma \vdash A \vee B$ (resp. $\Gamma \vdash A \wedge B$) can be turned into either a goal $\Gamma \vdash A$ or a goal $\Gamma \vdash B$ (resp. into two goals $\Gamma \vdash A$ and $\Gamma \vdash B$).

To sum up, during the proof construction process, a state is a set of sequents. These goals/sequents are modified by *commands*, which allow the user to navigate from the original statement of the theorem to the state where there are no goals left to be proved.

In the dominant paradigm, these commands are provided by the user in text form; since Robin Milner and LCF [23], they are called *tactics*. Proof files are literally *proof-scripts*; that is the sequence of tactics typed-in by the user.

The present work is a form of continuation of the *Proof-by-Pointing* (PbP) effort, initiated in the 1990's by Gilles Kahn, Yves Bertot, Laurent Théry and their group [4]. Both works share a main idea which is to replace the textual tactic commands by *gestural actions* performed by the user on a graphical user interface. In both cases, the *items* the user performs actions on are the current goal's conclusion and hypotheses. What is new in our work is that we allow not only to *click* on (subterms of) these items, but also to *move them* in order to drag-and-drop (DnD) one item onto another. This enriches the language of actions in, we argue, an intuitive way. We should point out that what is proposed here is not meant to replace but to complement the proof-by-pointing features. We thus envision a general *proof-by-action* paradigm, which includes both PbP and DnD features.

In this article, we focus on how drag-and-drop actions implement proof construction operations corresponding to the core logic; that is how they deal with logical connectives, quantifiers and equality. We have started to implement this in a prototype named *Actema* (for Active Mathematics) running through a web HTML5/JavaScript interface. This possibility to experiment in practice, even though yet on a small scale, gave valuable feedback for crafting the way DnD actions are

CPP '22, January 17–18, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9182-5/22/01…$15.00
https://doi.org/10.1145/3497775.3503692

to be translated into proof construction steps in an intuitive and practical way.

The rest of this article is organized as follows. Section 2 explains the motivations behind this work, and section 3 briefly outlines its logical setting. Section 4 describes the basic features of a graphical proof interface based on our principles, and illustrates them with a famous syllogism from Aristotle. Section 5 shows how it can integrate basic proof-by-pointing capabilities. The next two sections explain, through further examples, how the drag-and-drop paradigm works; first for so-called *rewrite* actions involving equalities, then for actions involving logical connectives and quantifiers. Section 8 introduces the notions of context and polarity, in order to prove the correctness of our system. Section 9 explains how DnD actions are specified by the user interactively, through schemas called *linkages*. Section 10 describes how linkages translate into logical steps, as well as some properties of this translation. Section 11 studies a proof of a small logical riddle in Actema, highlighting some benefits of our approach compared to textual systems. We end with a discussion on some related works in section 12, and then conclude.

2 Motivations

Since this work is about changing the very way the user interacts with an interactive theorem prover, we feel it is important to make some disclaimers about the aims and the scope of what is presented here.

From a development point of view, we are still at a very preliminary stage. Building a real-size proof system integrating the ideas we present would require an important effort and is still a long term goal. Some concepts however have emerged, which, we hope allow to sketch some aspects of the look-and-feel of such a system, and what some of its advantages could be.

Also, at this stage, we focus on basic proof constructions and on how the gestural approach can help make them more efficient and more intuitive. Some of the illustrative examples we give below could probably be dealt with using advanced proof search tactics, but we believe this does not make them irrelevant. Rather than (sub)goals to be proved, these examples should be seen as generic situations often encountered in the course of a proof, which require small and local transformations to the statements involved.

The idea of interactive theorem provers is that automation and user actions complement each other, and we here focus on the latter for the time being. The question of integrating drag-and-drop actions and powerful proof automation techniques is left for future work.

Finally, precisely because our approach is about giving the user a smoother control of the proof construction process, we see a possibility for our work to help making future proof systems more suited for education.

3 Logical Setting

Any proof system must implement a given logical formalism. What we describe here ought to be applied to a wide range of formalisms, but in this article we focus on the core of intuitionistic first-order logic with equality (FOL). This allows us to consider sequents where hypotheses are unordered which, in turn, simplifies the technical presentations. We will thus write $\Gamma, A \vdash B$ for a sequent where A is among the hypotheses.

We use and do not recall the usual definitions of terms and propositions in first order logic. We assume a first order language (function and predicate symbols) is given. Provability is defined over sequents $\Gamma \vdash A$ by the usual logical rules of natural deduction (NJ) and/or sequent calculus (LJ).

Equality is treated in a common way: $=$ is a binary predicate symbol written in the usual infix notation, together with the reflexivity axiom $\forall x.x = x$ and the Leibniz scheme, stating that for any proposition A one has

$$\forall x.\forall y.x = y \land A \Rightarrow A\,[x \setminus y]\,.$$

We will not consider, on paper, the details of variable renaming in substitutions, implicitly applying the so-called Barendregt convention, that bound and free variables are distinct and that a variable is bound at most once.

Extending this work to simple extensions of FOL, like multi-sorted predicate calculus is straightforward (and actually done in the prototype). Some more interesting points may show up when considering how to apply this work to more complex formalisms like type theories. We will not explore these questions here.

Another interesting and promising question is how our approach extends to classical logic(s), that is multi-conclusion classical sequents. In this text we only give a few hints on this topic.

4 A First Example

4.1 Layout

One advantage of the proof-by-actions paradigm, is that it allows a very lean visual layout of the proof state. There is no need to name hypotheses. In the prototype we also dispense with a text buffer, since proofs are solely built through graphical actions.

Figure 1 shows the layout of the system using the ancient example of Aristotle. A goal appears as a set of *items* whose nature is defined by their respective colors[1]:

- A *red item* which is the proposition to be proved, that is the *conclusion*,
- *blue items*, which are the local *hypotheses*.

[1]We are well aware that, in later implementations, this color-based distinction ought to be complemented by some other visual distinction, at least for users with impaired color vision. But in the present description we stick to the red/blue denomination, as it is conveniently concise.

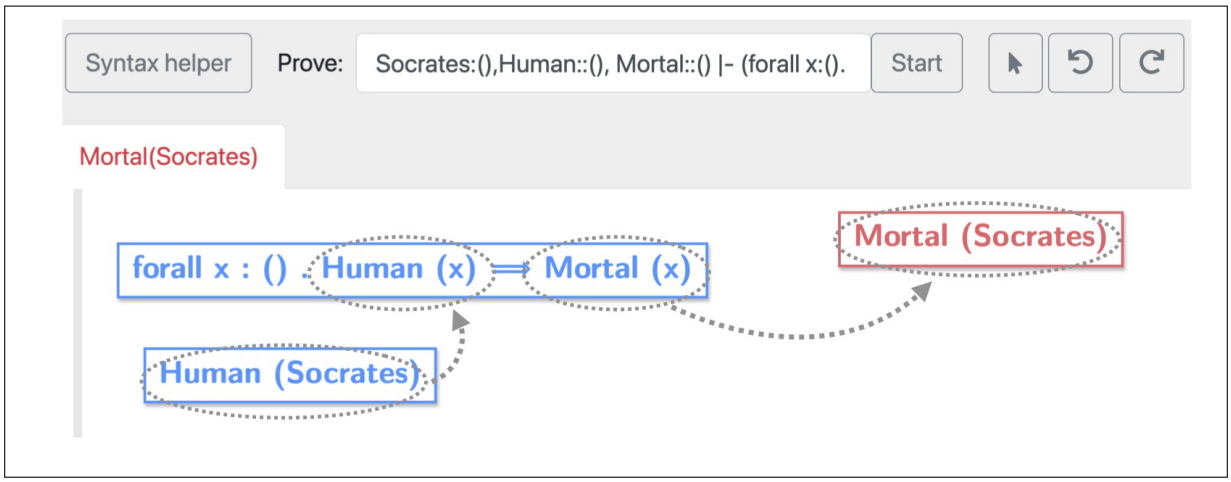

Figure 1. A partial screenshot showing a goal in the Actema prototype. The conclusion is red on the right, the two hypotheses blue on the left. The grey dotted arrows have been added to show the two possible actions.

The items are what the user can act upon: either by clicking *on them, or by* moving *them.*

Finally, note that each goal is displayed on a tab.

4.2 Two Kinds of Actions

In this example, there are two possible actions.

- A first one is to bring together, by drag-and-drop, the red conclusion Mort(Socr) with the succedant of the first hypothesis Mort(x). This will transform the goal by changing the conclusion to Hum(Socr).
- A second possibility is to combine the two hypotheses; more precisely to bring together the item Hum(Socr) with the premise Hum(x) of the first hypothesis. This will yield a new hypothesis Mort(Socr).

The first case is what we call a *backward step* where the conclusion is modified by using a hypothesis. The second case is a *forward step* where two known facts are combined to deduce a new fact, that is an additional blue item.

In both cases, the proof can then be finished invoking the logical axiom rule. In practice this means bringing together the blue hypothesis Hum(Socr) (resp. the new blue fact Mort(Socr)) with the red goal.

4.3 Modeling the Mechanism

A backward step involves a hypothesis, here $\forall x.\mathrm{Hum}(x) \Rightarrow \mathrm{Mort}(x)$ and the conclusion, here Mort(Socr). Furthermore, the action actually links together two *subterms* of each of these items; this is written by squaring these subterms. The symbol $\vdash$, used as an operator, is meant to describe the result of the interaction. Internally, the behavior of this operator is defined by a set of rewrite rules given in figures 3 and 4. Here is the sequence of rewrites corresponding to the example[2]:

$$
\begin{array}{lll}
 & \forall x.\mathrm{Hum}(x) \Rightarrow \boxed{\mathrm{Mort}(x)} \vdash \boxed{\mathrm{Mort}(\mathrm{Socr})} & \\
\triangleright & \mathrm{Hum}(\mathrm{Socr}) \Rightarrow \boxed{\mathrm{Mort}(\mathrm{Socr})} \vdash \boxed{\mathrm{Mort}(\mathrm{Socr})} & \mathsf{L}\forall\mathsf{i} \\
\triangleright & \mathrm{Hum}(\mathrm{Socr}) \wedge (\boxed{\mathrm{Mort}(\mathrm{Socr})} \vdash \boxed{\mathrm{Mort}(\mathrm{Socr})}) & \mathsf{L}\Rightarrow_2 \\
\triangleright & \mathrm{Hum}(\mathrm{Socr}) \wedge \top & \mathsf{id} \\
\triangleright & \mathrm{Hum}(\mathrm{Socr}) & \mathsf{neur}
\end{array}
$$

Notice that:

- These elementary rewrites are not visible for the user. What she/he sees is the final result of the action, that is the last expression of the rewrite sequence.
- The definitions of the rewrite rules in figures 3 and 4 do not involve squared subterms. The information of which subterms are squared is only used by the system to decide which rules to apply in which order.

In general, the action solves the goal when the interaction ends with the trivially true proposition $\top$. The base case being the action corresponding to the axiom/identity rule id: $A \vdash A \rhd \top$.

A forward step, on the other hand, involves two (subterms of two) hypotheses. The interaction operator between two hypotheses is written $*$. In the example above, the detail of the interaction is:

$$
\begin{array}{lll}
 & \forall x.\boxed{\mathrm{Hum}(x)} \Rightarrow \mathrm{Mort}(x) * \boxed{\mathrm{Hum}(\mathrm{Socr})} & \\
\triangleright & \boxed{\mathrm{Hum}(\mathrm{Socr})} \Rightarrow \mathrm{Mort}(\mathrm{Socr}) * \boxed{\mathrm{Hum}(\mathrm{Socr})} & \mathsf{F}\forall\mathsf{i} \\
\triangleright & (\boxed{\mathrm{Hum}(\mathrm{Socr})} \vdash \boxed{\mathrm{Hum}(\mathrm{Socr})}) \Rightarrow \mathrm{Mort}(\mathrm{Socr}) & \mathsf{F}\Rightarrow_1 \\
\triangleright & \top \Rightarrow \mathrm{Mort}(\mathrm{Socr}) & \mathsf{id} \\
\triangleright & \mathrm{Mort}(\mathrm{Socr}) & \mathsf{neul}
\end{array}
$$

The final result is the new hypothesis.

We come back to the study of the rewrite rules of $\vdash$ and $*$ further down.

[2]Note that $\vdash$ has lower precedence than all logical connectives.

5 Proof Steps through Clicks

Drag-and-drop actions involve two items. Some proof steps involve only one item; they can be associated to the action of clicking on this item. The general scheme is that clicking on a connective or quantifier allows to "break" or destruct this connective. The results of clicks are not very surprising, but this feature is necessary to complement drag-and-drop actions.

- Clicking on a blue conjunction $A \wedge B$ transforms the item into two separate blue items A and B.
- Clicking on a red conjunction $A \wedge B$ splits the goal into two subgoals, whose conclusions are respectively A and B.
- Clicking on a blue disjunction $A \vee B$ splits the goal into two subgoals of same conclusion, with A (resp. B) added as a new hypothesis.
- Clicking on the left (resp. right)-hand subterm of a red disjunction $A \vee B$ replaces this red conclusion by A (resp. B).
- Clicking on a red implication $A \Rightarrow B$ breaks it into a new red conclusion B and a new blue hypothesis A.
- Clicking on a red universal quantifier $\forall x.A$ introduces a new object x and the conclusion becomes A.
- Clicking on a blue existential $\exists x.A$ introduces a new object x together with a blue hypothesis A.
- Clicking on a red equality $t = t$ solves the goal immediately.

One can see that these actions correspond essentially to the introduction rules of the head connective for the conclusion, and the elimination rule for the hypotheses.

It is possible to associate some more complex effects to click actions performed on locations deeper under connectives. This is the essence of proof-by-pointing, and [4] provides ample description. Since we here focus on drag-and-drop actions, we do not detail further more advanced PbP features. However we stress that these features are essentially compatible with what we describe in this work.

Adding New Items. Often in the course of a proof, one will want to add new items: either a new conjecture (blue item), or a new object (green item) that would be helpful to solve the current goal. These can be done respectively with the blue `+hyp` and the green `+expr` buttons, which appear in the screenshot of figure 5. When clicked, they prompt the user for the statement of the conjecture, or the name and expression defining the object. The `+hyp` button will also create a new subgoal requiring to prove the conjecture within the current context.

This mechanism and the syntax are for now very crude. The design of possible smoother tools is an important issue but left for future work[3].

[3]For instance [25] deals with a similar problem in the context of functional programming.

6 A Simple Example Involving Equality

In most interactive theorem provers, the most basic *rewrite* tactic allows the use of equality hypotheses, that is known equations of the form $t = u$, in order to replace some occurrences of t by u (or symmetrically, occurrences of u by t). This substitution can be performed in the conclusion or in hypotheses. Specifying the occurrences to be replaced with textual commands can be quite tedious, since it involves either dealing with some form of naming/numbering, or writing manually patterns which duplicate parts of the structure of terms.

In our setting we can provide this replacement operation through drag-and-drop. The user points at the occurrence(s) of t to be replaced, and then brings them to the corresponding side of the equality.

Figure 2 shows a very elementary example which is proving $1 + 1 = 2$ in the setting of Peano arithmetic. For any number n, we write $n \oplus 1$ to denote the application of the successor function to n; closed terms are directly written in decimal notation. The proof goes as follows[4]:

- We link the left-hand side $x+y\oplus 1$ of the second addition axiom with $1+1$ in the conclusion, which has the effect of rewriting $1 + 1$ into $(1 + 0)\oplus 1$:

$$
\begin{array}{lll}
 & \forall x.\forall y.\boxed{x + y \oplus 1} = (x + y) \oplus 1 \ \vdash\ \boxed{1 + 1} = 2 & \\
\rhd & \forall y.\boxed{1 + y \oplus 1} = (1 + y) \oplus 1 \ \vdash\ \boxed{1 + 1} = 2 & \mathsf{L}\forall\mathsf{i} \\
\rhd & \boxed{1 + 0 \oplus 1} = (1 + 0) \oplus 1 \ \vdash\ \boxed{1 + 1} = 2 & \mathsf{L}\forall\mathsf{i} \\
\equiv & \boxed{1 + 1} = (1 + 0) \oplus 1 \ \vdash\ \boxed{1 + 1} = 2 & \\
\rhd & (1 + 0) \oplus 1 = 2 & \mathsf{L}{=}_1
\end{array}
$$

- We link the right-hand side $x + 0$ of the first addition axiom with $1+0$ in the conclusion, which rewrites $1+0$ into 1:

$$
\begin{array}{lll}
 & \forall x.x = \boxed{x + 0} \ \vdash\ (\boxed{1 + 0}) \oplus 1 = 2 & \\
\rhd & 1 = \boxed{1 + 0} \ \vdash\ (\boxed{1 + 0}) \oplus 1 = 2 & \mathsf{L}\forall\mathsf{i} \\
\rhd & 1 \oplus 1 = 2 & \mathsf{L}{=}_2 \\
\equiv & 2 = 2 &
\end{array}
$$

We end up with the conclusion $2 = 2$, which is provable by a simple click. Notice how the orientation of the two rewritings is determined by which side of the equality is selected. Also, in this case, the rewritings correspond to backward proof steps, because the rewriting is performed in the conclusion. Similar rules ($\mathsf{F}{=}_1$ and $\mathsf{F}{=}_2$) are used to perform rewritings in hypotheses.

7 Drag-and-Dropping through Connectives

We mentioned in section 5 that it is possible to destruct logical connectives through click actions. In many cases however, this will not be necessary: because a drag-and-drop involves subterms of the items involved, one can often

[4]We use the symbol $\equiv$ to denote syntactic equality of two expressions modulo *pretty-printing*, e.g. decimal notation.

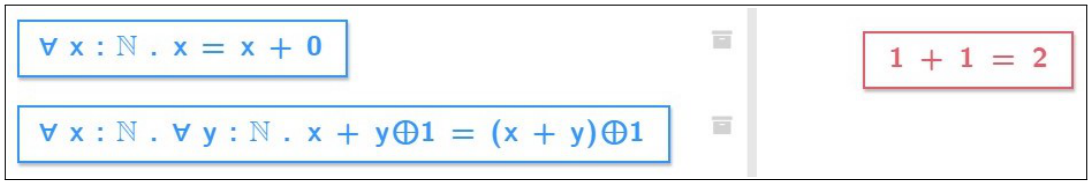

Figure 2. Proving 1 + 1 = 2 in Peano arithmetic.

directly use (resp. act on) the part of the hypothesis (resp. conclusion) which is of interest.

7.1 Conjunction and Disjunction

The conjunction is an easy to explain case. A hypothesis of the form $A \wedge B$ can be used directly both as evidence for A and as evidence for B. This is modeled by the rules $\mathsf{L}\wedge_1$ and $\mathsf{L}\wedge_2$. A very simple action is thus:

$$\begin{array}{lcll} \boxed{A} \wedge B \vdash \boxed{A} & \rhd & \boxed{A} \vdash \boxed{A} & \mathsf{L}\wedge_1 \\ & \rhd & \top & \mathsf{id} \end{array}$$

On the other hand, considering a conjunctive goal $A \wedge B$, one can simplify or solve one of the branches by a DnD action. This involves rules $\mathsf{R}\wedge_1$ and $\mathsf{R}\wedge_2$. For instance:

$$\begin{array}{lcll} \boxed{A} \vdash \boxed{A} \wedge B & \rhd & \boxed{A} \vdash \boxed{A} \wedge B & \mathsf{R}\wedge_1 \\ & \rhd & \top \wedge B & \mathsf{id} \\ & \rhd & B & \mathsf{neul} \end{array}$$

Red disjunctions work similarly to conjunctive goals, except that solving one branch will solve the entire goal. A nice consequence of this, which is hard to simulate with textual tactics, is that one can just simplify one branch of a disjunction without comitting to it:

$$\begin{array}{lcll} \boxed{A} \vdash (B \wedge \boxed{A}) \vee C & \rhd & (\boxed{A} \vdash B \wedge \boxed{A}) \vee C & \mathsf{R}\vee_1 \\ & \rhd & (B \wedge (\boxed{A} \vdash \boxed{A})) \vee C & \mathsf{R}\wedge_2 \\ & \rhd & (B \wedge \top) \vee C & \mathsf{id} \\ & \rhd & B \vee C & \mathsf{neur} \end{array}$$

Disjunctive hypotheses also have a backward behavior defined by the rules $\mathsf{L}\vee_1$ and $\mathsf{L}\vee_2$, although in most cases one will prefer the usual subgoal semantics associated with click actions. More interesting is their forward behavior with the rules $\mathsf{F}\vee_1$ and $\mathsf{F}\vee_2$, in particular when they interact with negated hypotheses. For instance:

$$\begin{array}{lcll} \boxed{A} \vee B * \neg\boxed{A} & \rhd & (\boxed{A} * \neg\boxed{A}) \vee B & \mathsf{F}\vee_1 \\ & \rhd & \neg(\boxed{A} \vdash \boxed{A}) \vee B & \mathsf{F}\Rightarrow_1 \\ & \rhd & \neg\top \vee B & \mathsf{id} \\ & \rhd & \bot \vee B & \mathsf{neul} \\ & \rhd & B & \mathsf{neul} \end{array}$$

We have noticed that on some examples, such actions could provide a significant speed-up with respect to traditional textual command provers. We give a more concrete example in section 11.

Notice that we used rules associated with implication, since negation can be defined by $\neg A \triangleq A \Rightarrow \bot$.

7.2 Implication

The implication connective is crucial, because it is not monotone. More precisely, the roles of hypotheses and conclusions are reversed on the left of an implication. We start with some very basic examples for the various elementary cases.

Using the right hand part of a hypothesis $A \Rightarrow B$ turns a conclusion B into A.

$$\begin{array}{lcll} A \Rightarrow \boxed{B} \vdash \boxed{B} & \rhd & A \wedge (\boxed{B} \vdash \boxed{B}) & \mathsf{L}\Rightarrow_2 \\ & \rhd & A \wedge \top & \mathsf{id} \\ & \rhd & A & \mathsf{neul} \end{array}$$

This can also be done under conjunctions and/or disjunctions:

$$A \Rightarrow \boxed{B} \vdash C \wedge (D \vee \boxed{B}) \quad \rhd^* \quad C \wedge (D \vee A)$$

An interesting point is what happens when using implications with several premisses. The curried and uncurried versions of the implication will behave exactly the same way:

$$A \Rightarrow B \Rightarrow \boxed{C} \vdash D \vee \boxed{C} \quad \rhd^* \quad D \vee (A \wedge B)$$

and:

$$A \wedge B \Rightarrow \boxed{C} \vdash D \vee \boxed{C} \quad \rhd^* \quad D \vee (A \wedge B)$$

As we have seen in Aristotle's example (section 4), blue implications can also be used in forward steps, where another hypothesis matches one of their premisses.

A first nice feature is the ability to strengthen a hypothesis by providing evidence for any of its premises:

$$B \Rightarrow \boxed{A} \Rightarrow C * \boxed{A} \quad \rhd^* \quad B \Rightarrow C$$

and again the same can be done for the uncurryfied version:

$$B \wedge \boxed{A} \Rightarrow C * \boxed{A} \quad \rhd^* \quad B \Rightarrow C.$$

The two aspects of the implication can be combined:

$$B \Rightarrow \boxed{A} \Rightarrow C * D \Rightarrow \boxed{A} \quad \rhd^* \quad B \Rightarrow D \Rightarrow C$$

or:

$$B \wedge \boxed{A} \Rightarrow C * D \Rightarrow \boxed{A} \quad \rhd^* \quad B \wedge D \Rightarrow C.$$

Note that there is almost no difference in the way one uses different versions of a hypothesis $A \Rightarrow B \Rightarrow C$, $A \wedge B \Rightarrow C$, but also $B \Rightarrow A \Rightarrow C$, in forward as well as in backward

steps[5]. This underlines, we hope, that our proposal makes the proof construction process much less dependent on arbitrary syntactical details, like the order of hypotheses or whether they come in curryfied form or not.

Also, the rules for implication combined with the rules for equality $\mathsf{L}{=_i}$ or $\mathsf{F}{=_i}$ naturally give access to *conditional rewriting*; we detail this in combination with quantifiers in the next section.

As for red implications, they also have a backward semantics with the rules $\mathsf{R}{\Rightarrow_1}$ and $\mathsf{R}{\Rightarrow_2}$, but most of the time one will want to destruct them immediately by click. An exception could be if one wants to simplify some part of an implicative, inductive goal before starting the induction.

7.3 Quantifiers

As the first example of this paper shows, the drag-and-drop actions work through quantifiers and can trigger instantiations of quantified variables. This is made possible by the rules $\mathsf{L}\forall\mathsf{i}$ and $\mathsf{F}\forall\mathsf{i}$, which allow the instantiation of a variable universally quantified in a hypothesis.

Symmetrically, a variable quantified existentially in a conclusion can also be instantiated. For instance:

$$\begin{array}{lll} \boxed{A(t)} \vdash \exists x.\boxed{A(x)} & \rhd & \boxed{A(t)} \vdash \boxed{A(t)} \quad \mathsf{L}\forall\mathsf{i} \\ & \rhd & \top \qquad\qquad\qquad\quad \mathsf{id} \end{array}$$

An interesting feature is the possibility to modify propositions under quantifiers. Consider the following possible goal:

$$\forall a.\exists b.A(f(a)+g(b))$$

where A, f and g can be complex expressions. Suppose we have a lemma allowing us to prove:

$$\forall a.\exists b.A(g(b)+f(a)).$$

Switching from one formulation to the other, involves one use of the commutativity property $\forall x.\forall y.x+y=y+x$. In our setting, the equality can be used under quantifiers in one single action:

$$\begin{array}{ll} & \forall x.\forall y.\boxed{x+y}=y+x \vdash \forall a.\exists b.A(\boxed{f(a)+g(b)}) \\ \rhd^* & \forall a.\exists b.A(g(b)+f(a)) \end{array}$$

Note also that it is possible to instantiate only some of the universally quantified variables in the items involved. In general, a universally quantified variable can be instantiated when the quantifier is in a negative position; for instance:

$$\forall x.\forall y.\boxed{P(y)} \Rightarrow R(x,y) \;*\; \boxed{P(a)} \quad \rhd^* \quad \forall x.R(x,a)$$

This last example illustrates how partial instantiation abstracts away the order in which quantifiers are declared, very much like the partial application presented in section 7.2.

Again, in some cases, only some existential quantifiers may be instantiated following a linkage:

$$\boxed{P(a)} \vdash \exists x.\exists y.\boxed{P(y)} \wedge R(x,y) \quad \rhd^* \quad \exists x.R(x,a)$$

When using an existential assumption, one can either destruct it through a click, or use or transform it through a DnD; for instance:

$$\exists x.\boxed{P(x)} \;*\; \forall y.\boxed{P(y)} \Rightarrow Q(y) \quad \rhd^* \quad \exists x.Q(x)$$

7.4 Dependency between Variables

Some more advanced examples yield simultaneous instantiations of existentially and universally quantified variables. In such cases, the system needs to check some dependency conditions. For instance, the following linkage is valid and solves the goal through one action:

$$\begin{array}{lll} & \exists y.\forall x.\boxed{R(x,y)} \vdash \forall x'.\exists y'.\boxed{R(x',y')} & \\ \rhd & \forall y.(\forall x.\boxed{R(x,y)} \vdash \forall x'.\exists y'.\boxed{R(x',y')}) & \mathsf{L}\exists\mathsf{s} \\ \rhd & \forall y.\forall x'.(\forall x.\boxed{R(x,y)} \vdash \exists y'.\boxed{R(x',y')}) & \mathsf{R}\forall\mathsf{s} \\ \rhd & \forall y.\forall x'.(\forall x.\boxed{R(x,y)} \vdash \boxed{R(x',y)}) & \mathsf{R}\exists\mathsf{i} \\ \rhd & \forall y.\forall x'.(\boxed{R(x',y)} \vdash \boxed{R(x',y)}) & \mathsf{L}\forall\mathsf{i} \\ \rhd & \forall y.\forall x'.\top & \mathsf{id} \\ \rhd^* & \top & \end{array}$$

But the contraposed situation is not provable; the system will refuse the following linkage:

$$\forall x.\exists y.\boxed{R(x,y)} \vdash \exists y'.\forall x'.\boxed{R(x',y')}$$

Indeed, there is no reduction path starting from this linkage ending with the id rule. This can be detected by the system because the unification of $R(x,y)$ and $R(x',y')$ here results in a cycle in the instantiations of variables[6]. The system thus refuses this action.

7.5 Conditional Rewriting

The example given in section 6, although very simple, already combines the rules for equality and for quantifiers. When also using implication, one obtains naturally some form of conditional rewriting. To take another simple example, suppose we have a hypothesis of the form:

$$\forall x.x \neq 0 \Rightarrow f(x)=g(x)$$

We can use this hypothesis for replacing a subterm $f(t)$ by $g(t)$, which will generate a side-condition $t \neq 0$:

$$\begin{array}{lll} & \forall x.x \neq 0 \Rightarrow \boxed{f(x)}=g(x) \vdash A(\boxed{f(t)}) & \\ \rhd & t \neq 0 \Rightarrow \boxed{f(t)}=g(t) \vdash A(\boxed{f(t)}) & \mathsf{L}\forall\mathsf{i} \\ \rhd & t \neq 0 \wedge (\boxed{f(t)}=g(t) \vdash A(\boxed{f(t)})) & \mathsf{L}{\Rightarrow_2} \\ \rhd & t \neq 0 \wedge A(g(t)) & \mathsf{L}{=_1} \end{array}$$

[5]When viewed as types through the Curry-Howard isomorphism, $A \Rightarrow B \Rightarrow C$, $A \wedge B \Rightarrow C$, $B \wedge A \Rightarrow C$ and $B \Rightarrow A \Rightarrow C$ are *isomorphic types*; and Roberto di Cosmo [10] has also precisely underlined that type isomorphisms should help to free the programmer from arbitrary syntactical choices.

[6]Also notice that this example requires to use full (first-order) unification, not only matching.

One could similarly do such a rewrite in a hypothesis. Furthermore, the conditional rewrite can also be performed under quantifiers; for instance:

$$\begin{array}{lll} & \forall x.x \neq 0 \Rightarrow \boxed{f(x)} = g(x) \ \vdash \ \exists y.A(\boxed{f(y)}) & \mathsf{R}\exists\mathsf{s} \\ \rhd & \exists y.(\forall x.x \neq 0 \Rightarrow \boxed{f(x)} = g(x) \ \vdash \ A(\boxed{f(y)})) & \mathsf{L}\forall\mathsf{i} \\ \rhd & \exists y.(y \neq 0 \wedge (\boxed{f(y)} = g(y) \ \vdash \ A(\boxed{f(y)}))) & \mathsf{L}{\Rightarrow_2} \\ \rhd & \exists y.(y \neq 0 \wedge A(g(t))) & \mathsf{L}{=_1} \end{array}$$

8 Correctness

All examples up to now followed the scheme for DnD actions sketched in section 4:

- Given a blue item A and a red item B, backward proof steps produce a new conclusion C by applying a sequence of rewrite rules $A \vdash B \ \rhd^* \ C$.
- Given two blue items A and B, forward proof steps produce a new hypothesis C by applying a sequence of rewrite rules $A * B \ \rhd^* \ C$.

Thus for such actions to be logically correct, we have to make sure that our rewrite system satisfies the following property:

Property 1 (Correctness)**.**

- *If $A \vdash B \ \rhd^* \ C$, then $A, C \vdash B$ is provable.*
- *If $A * B \ \rhd^* \ C$, then $A, B \vdash C$ is provable.*

We first need a few definitions to handle the fact that rewrite rules apply at any depth inside formulas:

Definition 8.1 (Context)**.** A context, written $A\square$, is a proposition containing exactly one occurrence of a specific propositional variable $\square$ which is not used elsewhere.

Given another proposition B, we write $A[\boxed{B}]$ for the proposition obtained by replacing $\square$ in $A\square$ by B. Note that this replacement is not a substitution because it allows variable capture. For instance $\forall x.\boxed{P(x)}$ is the proposition $\forall x.P(x)$.

Definition 8.2 (Path)**.** A path is a proposition where one subformula has been selected. Formally, a path is a pair $(A\square, B)$ formed by one context and one proposition:

- $A\square$ is called the *context* of the path,
- B is called the *selection* of the path.

The path $(A\square, B)$ can be viewed as the proposition $A[\boxed{B}]$. For readability, we will generally also write $A[\boxed{B}]$ for the path $(A\square, B)$.

Definition 8.3 (Inversions)**.** Given a context $A\square$, the number of inversions in $A\square$, written $inv(A\square)$, is the number of subterms of $A\square$ which are of the form $C\square \Rightarrow D$; that is the number of times the hole is on the left-hand side of an implication.

For instance:

$$\begin{array}{rcl} inv(D \wedge \square) & = & 0 \\ inv((D \wedge \square) \Rightarrow E) & = & 1 \\ inv((\square \Rightarrow C) \Rightarrow D) & = & 2 \end{array}$$

Definition 8.4 (Polarity of a context)**.** We will write $A^+\square$ to specify that a context is *positive*, meaning that $inv(A^+\square)$ is even. Symmetrically, $A^-\square$ will be used for *negative* contexts, meaning that $inv(A^-\square)$ is odd.

The following simple covariance and contravariance property will be used extensively later on:

Property 2. *If $\Gamma, A \vdash B$ is provable, then so are $\Gamma, C^+[\boxed{A}] \vdash C^+[\boxed{B}]$ and $\Gamma, D^-[\boxed{B}] \vdash D^-[\boxed{A}]$.*

For each rule, interpreting $\vdash$ as $\Rightarrow$ and $*$ as $\wedge$ in the right-hand side is enough to show that the rule satisfies property 1 locally. But for rewritings taking place at occurrences deeper inside a proposition, we need to consider the polarity of their context. Using the notation of definition 8.4, we can state:

Lemma 8.5.

- *If $C^+[\boxed{A \vdash B}] \ \rhd \ D$ then $D \vdash C^+[\boxed{A \Rightarrow B}]$ is provable.*
- *If $C^-[\boxed{A \vdash B}] \ \rhd \ D$ then $C^-[\boxed{A \Rightarrow B}] \vdash D$ is provable.*
- *If $C^+[\boxed{A * B}] \ \rhd \ D$ then $C^+[\boxed{A \wedge B}] \vdash D$ is provable.*
- *If $C^-[\boxed{A * B}] \ \rhd \ D$ then $D \vdash C^-[\boxed{A \wedge B}]$ is provable.*

Remark 1. *For some rules, like $\mathsf{R}{\Rightarrow_1}$, the left-hand and right-hand propositions are equivalent:*

$$A \Rightarrow B \Rightarrow C \ \Leftrightarrow \ A \wedge B \Rightarrow A$$

Such rules are called invertible *and their names are tagged by $*$. This point will be relevant in section 10.2.*

An easy but important technical point is that rewrite rules preserve the polarity of contexts around redexes, in the following precise sense:

Property 3. *If $C[\boxed{A \vdash B}] \ \rhd \ C'[\boxed{A' \vdash B'}]$ (resp. $C[\boxed{A * B}] \ \rhd \ C'[\boxed{A' * B'}]$) then $C\square$ and $C'\square$ have the same polarity.*

If $C[\boxed{A \vdash B}] \ \rhd \ C'[\boxed{A' * B'}]$ (resp. $C[\boxed{A * B}] \ \rhd \ C'[\boxed{A' \vdash B'}]$) then $C\square$ and $C'\square$ have opposite polarities.

Combining lemma 8.5 and property 3, we obtain the central correctness result about the rewrite rules:

Lemma 8.6.

- *If $C^+[\boxed{A \vdash B}] \ \rhd^* \ D$ then $D \vdash C^+[\boxed{A \Rightarrow B}]$ is provable.*
- *If $C^-[\boxed{A \vdash B}] \ \rhd^* \ D$ then $C^-[\boxed{A \Rightarrow B}] \vdash D$ is provable.*
- *If $C^+[\boxed{A * B}] \ \rhd^* \ D$ then $C^+[\boxed{A \wedge B}] \vdash D$ is provable.*
- *If $C^-[\boxed{A * B}] \ \rhd^* \ D$ then $D \vdash C^-[\boxed{A \wedge B}]$ is provable.*

We do not detail the proof here, but it relies crucially on the covariance and contravariance property 2.

Finally, property 1 is obtained as the special case where the rewriting starts in the (positive) empty context.

9 Linkages

In demonstrating correctness, we focused solely on the two *items* involved in a DnD action. But every DnD action also specifies the *selection* of a subterm in each item. We call *linkage* the combined data of the two items together with the selection, since the intent is to *link* the subterms to make them interact in some way.

Remark 2. *In this article we only consider linkages between two subterms, but as noted in section 6, rewriting is an example of action that can benefit from allowing multiple selections*[7].

Each kind of DnD action is mapped in the system to a specific form of linkage, which is designed to hold all the information necessary for the correct execution of the action. In this way the system can automatically search for linkages of a certain form, and propose to the user all well-defined actions associated to these linkages.

Remark 3. *In the future, one can imagine several DnD actions associated to a given linkage. In this case, the user could be queried to choose the action to be performed (typically with a pop-up menu). However with the actions considered in this article, such ambiguities never arise.*

On the "items axis", we already distinguished between backward and forward linkages, written respectively $A \vdash B$ and $A * B$. If the items are unspecified, we will write $A \mathbin{@} B$.

Using the "selection axis", we can specify a further distinction that was informal up to now: that of *logical* action and *rewrite* action.

- *Logical* linkages link two subformulas. Thus they have the form $B\boxed{A} \mathbin{@} C\boxed{A'}$.
- *Rewrite* linkages link one side of an equality with a first-order term. Using liberally the notations from definitions 8.1 and 8.2, they thus have the form $B\boxed{\overline{t} = u} \mathbin{@} C\boxed{t'}$ (or symmetrically $B\boxed{u = \overline{t}} \mathbin{@} C\boxed{t'}$).

In both cases, we impose the following condition:

Condition 1 (Unification)**.** *The linked subterms (A and A' or t and t') must be* unifiable *with respect to the variables quantified in their contexts $B\Box$, $C\Box$. We do not detail all the constraints of this unification problem, but the essential idea is that a variable is unifiable if and only if its quantifier is* instantiable*. This in turn can be determined by the polarity of the context surrounding the quantifier. One also needs to check that the unifier does not create circular dependencies between variables.*

There are also additional restrictions on the polarities of contexts. Assuming we work in a goal $\Gamma \vdash C$:

Condition 2 (Polarity)**.** *The following conditions are necessary for the linkage $B\boxed{A} \mathbin{@} D\boxed{A'}$ to be a logical linkage:*

1. $B\boxed{A} \in \Gamma$,
2. $D\boxed{A'} \begin{cases} \equiv C & \text{if } @ \text{ is } \vdash \\ \in \Gamma & \text{if } @ \text{ is } * \end{cases}$
3. $(inv(B\Box), inv(D\Box)) \in \begin{cases} \{(0,0),(1,1),(0,2)\} & \text{if } @ \text{ is } \vdash \\ \{(0,1),(1,0)\} & \text{if } @ \text{ is } * \end{cases}$

Similarly, the following conditions are necessary for the linkage $B\boxed{\overline{t} = u} \mathbin{@} D\boxed{t'}$ to be a rewrite linkage. Either $B\boxed{\overline{t} = u} \in \Gamma$, then:

1. $D\boxed{t'} \begin{cases} \equiv C & \text{if } @ \text{ is } \vdash \\ \in \Gamma & \text{if } @ \text{ is } * \end{cases}$
2. *$B\Box$ is positive*

or $B\boxed{\overline{t} = u} \equiv C$, then $@$ is $\vdash$ and:

1. $D\boxed{t'} \in \Gamma$
2. *$B\Box$ is negative*

Remark 4. *The main reason for limiting the number of inversions in a path to 2 is that we place ourselves in intuitionistic logic. In classical logic, one could, for instance, imagine the following behavior:*

$$(\boxed{A} \Rightarrow B) \Rightarrow C \;\vdash\; \boxed{A} \;\triangleright^*\; C \Rightarrow A$$

But this would not be valid intuitionistically.

Definition 9.1 (Valid linkage)**.** We say that a linkage $\mathcal{L}$ is *valid* if it satisfies conditions 1 and 2.

One understands that for logical linkages, condition 2 guarantees that there is one positive and one negative occurrence among A and A'. For rewrite linkages, it guarantees that the equality is in negative position.

One can also check that all the examples given up to here were based on valid linkages.

10 Describing DnD Actions

We are now equipped to specify how logical and rewrite linkages translate deterministically to the backward and forward proof steps shown in all examples.

First some remarks can be made about the rewrite rules of figure 3:

- The set of rewrite rules is obviously non-confluent.
- It is also terminating, because the number of connectives or quantifiers under $*$ or $\vdash$ decreases[8].

As for the rules of figure 4, they are both terminating *and* confluent. Indeed they define a function that eliminates redundant occurrences of the units $\top$ and $\bot$.

Here is a high-level overview of the complete procedure followed to generate a proof step:

1. **Selection:** the user selects two subterms in two items of the current goal;
2. **Linkage:** this either gives rise to a logical linkage $B\boxed{A} \mathbin{@} C\boxed{A'}$ (resp. a rewrite linkage $B\boxed{\overline{t} = u} \mathbin{@} C\boxed{t'}$), or does not correspond to a known form of linkage. In this case the procedure stops here, and the system does not propose any action to the user;

[7] In fact, a restricted kind of multi-occurrence rewrite is already available in the current prototype of Actema: one just needs to enter *selection mode*, by either toggling the dedicated button, or holding down the `shift` key.

[8] Except for the Fcomm rule which is just meant to make the $*$ connective symmetric; formally, the only infinite reduction paths end with an infinite iteration of Fcomm.

3. **Unification:** the system tries to unify the selected subterms A and A' (resp. t and t'), which either yields a substitution σ, or fails. In this case we stop like in the previous step;
4. **Linking:** the system then chooses a rewriting starting from the linkage. Thanks to theorem 10.2, this rewriting always ends with a proposition of the form $D\boxed{\sigma(A) \vdash \sigma(A')}$ (resp. $D\boxed{\boxed{\sigma(t)} = u \ @ \ C_0\boxed{\sigma(t')}}$);
5. **Interaction:** thus one can apply the id rule (resp. an equality rule in $\{\mathsf{L{=}_1}, \mathsf{L{=}_2}, \mathsf{F{=}_1}, \mathsf{F{=}_2}\}$);
6. **Unit elimination:** in the case of a logical action, this creates an occurrence of $\top$, which is eliminated using the rules of figure 4;
7. **Goal modification:** the two previous steps produced a formula E. In the case of a forward linkage, a hypothesis E is added to the goal; in the case of a backward linkage, the goal's conclusion becomes E. In both cases, the logical correctness is guaranteed by property 1.

10.1 Productivity

An important property of the linking step 4 is that there is always a rewriting sequence that brings together the selected subterms, which ensures that we can proceed to the interaction step 5.

Because the rewrite rules are terminating, the important point is to show that one can always apply a rule until one reaches an interaction rule on the linkage. In other words, it is possible to find at least one rule which preserves conditions 1 and 2 on linkages:

Lemma 10.1 (Valid Progress)**.** *If a linkage $\mathcal{L}$ is valid, then either:*

1. $\mathcal{L} \in \{\boxed{A} \vdash \boxed{A}, \boxed{t} = u \vdash A\boxed{t}, u = \boxed{t} \vdash A\boxed{t}, \boxed{t} = u * A\boxed{t}, u = \boxed{t} * A\boxed{t}\}$;
2. *or $\mathcal{L} \rhd D\boxed{\mathcal{L}'}$ with $\mathcal{L}'$ valid.*

The proof is not fundamentally difficult, but understandably verbose. The two main points are:

- The rules involving a connective always preserve validity.
- When one can apply a rule involving a quantifier $\forall x$ (resp. $\exists x$), one checks whether the substitution instantiates x or not. In the first case one performs the instantiation rule $\mathsf{L\forall_i}$ or $\mathsf{F\forall_i}$ (resp. $\mathsf{R\exists_i}$); in the second case the corresponding s rule.

Then we can state the following *productivity theorem*, which is a direct consequence of the previous lemma and the fact that the rewrite rules terminate:

Theorem 10.2 (Productivity)**.** *If $\mathcal{L}$ is a valid linkage, then there is a sequence of reductions with one of the following*

Backward

$$
\begin{array}{rcll}
A \vdash A & \rhd & \top & \mathsf{id} \\
t = u \vdash A & \rhd & A\,[t \setminus u] & \mathsf{L{=}_1} \\
u = t \vdash A & \rhd & A\,[t \setminus u] & \mathsf{L{=}_2} \\
\\
(B \wedge C) \vdash A & \rhd & B \vdash A & \mathsf{L\wedge_1} \\
(C \wedge B) \vdash A & \rhd & B \vdash A & \mathsf{L\wedge_2} \\
A \vdash (B \wedge C) & \rhd & (A \vdash B) \wedge C & \mathsf{R\wedge_1} \\
A \vdash (C \wedge B) & \rhd & C \wedge (A \vdash B) & \mathsf{R\wedge_2} \\
\\
(B \vee C) \vdash A & \rhd & (B \vdash A) \wedge (C \Rightarrow A) & \mathsf{L\vee_1}^* \\
(C \vee B) \vdash A & \rhd & (C \Rightarrow A) \wedge (B \vdash A) & \mathsf{L\vee_2}^* \\
A \vdash (B \vee C) & \rhd & (A \vdash B) \vee C & \mathsf{R\vee_1} \\
A \vdash (C \vee B) & \rhd & C \vee (A \vdash B) & \mathsf{R\vee_2} \\
\\
(C \Rightarrow B) \vdash A & \rhd & C \wedge (B \vdash A) & \mathsf{L\Rightarrow_2} \\
A \vdash (B \Rightarrow C) & \rhd & (A * B) \Rightarrow C & \mathsf{R\Rightarrow_1}^* \\
A \vdash (C \Rightarrow B) & \rhd & C \Rightarrow (A \vdash B) & \mathsf{R\Rightarrow_2}^* \\
\\
(\forall x.B) \vdash A & \rhd & B\,[x \setminus t] \vdash A & \mathsf{L\forall i} \\
(\forall x.B) \vdash A & \rhd & \exists x.(B \vdash A) & \mathsf{L\forall s} \\
A \vdash (\forall x.B) & \rhd & \forall x.(A \vdash B) & \mathsf{R\forall s}^* \\
\\
(\exists x.B) \vdash A & \rhd & \forall x.(B \vdash A) & \mathsf{L\exists s}^* \\
A \vdash (\exists x.B) & \rhd & A \vdash B\,[x \setminus t] & \mathsf{R\exists i} \\
A \vdash (\exists x.B) & \rhd & \exists x.(A \vdash B) & \mathsf{R\exists s}
\end{array}
$$

Forward

$$
\begin{array}{rcll}
A * (t = u) & \rhd & A\,[t \setminus u] & \mathsf{F{=}_1} \\
A * (u = t) & \rhd & A\,[t \setminus u] & \mathsf{F{=}_2} \\
\\
A * (B \wedge C) & \rhd & A * B & \mathsf{F\wedge_1} \\
A * (C \wedge B) & \rhd & A * B & \mathsf{F\wedge_2} \\
\\
A * (B \vee C) & \rhd & (A * B) \vee C & \mathsf{F\vee_1} \\
A * (C \vee B) & \rhd & C \vee (A * B) & \mathsf{F\vee_2} \\
\\
A * (B \Rightarrow C) & \rhd & (A \vdash B) \Rightarrow C & \mathsf{F\Rightarrow_1} \\
A * (C \Rightarrow B) & \rhd & C \Rightarrow (A * B) & \mathsf{F\Rightarrow_2} \\
\\
A * (\forall x.B) & \rhd & A * B\,[x \setminus t] & \mathsf{F\forall i} \\
A * (\forall x.B) & \rhd & \forall x.(A * B) & \mathsf{F\forall s} \\
\\
A * (\exists x.B) & \rhd & \exists x.(A * B) & \mathsf{F\exists s}^* \\
\\
B * A & \rhd & A * B & \mathsf{Fcomm}
\end{array}
$$

In the rules $\{\mathsf{L\forall s}, \mathsf{L\exists s}, \mathsf{R\forall s}, \mathsf{R\exists s}, \mathsf{F\forall s}, \mathsf{F\exists s}\}$, x is not free in A.

Figure 3. Linking rules

Units

$$
\begin{array}{lrcll}
\langle \circ, \dagger \rangle \in \{\langle \wedge, \top \rangle, \langle \vee, \bot \rangle, \langle \Rightarrow, \top \rangle\} & \dagger \circ A & \rhd & A & \mathsf{neul} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \top \rangle, \langle \vee, \bot \rangle\} & A \circ \dagger & \rhd & A & \mathsf{neur} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \bot \rangle, \langle \vee, \top \rangle\} & \dagger \circ A & \rhd & \dagger & \mathsf{absl} \\
\langle \circ, \dagger \rangle \in \{\langle \wedge, \bot \rangle, \langle \vee, \top \rangle, \langle \Rightarrow, \top \rangle\} & A \circ \dagger & \rhd & \dagger & \mathsf{absr} \\
\langle \diamond, \dagger \rangle \in \{\langle \forall, \top \rangle, \langle \exists, \bot \rangle\} & \diamond x.\dagger & \rhd & \dagger & \mathsf{absq} \\
 & \bot \Rightarrow A & \rhd & \top & \mathsf{efq}
\end{array}
$$

Figure 4. Unit elimination rules

forms:

$$\mathcal{L} \rhd^* D^+\boxed{A} \vdash \boxed{A}$$

$$\mathcal{L} \rhd^* D\boxed{t = u \ @ \ A\boxed{t}} \qquad \mathcal{L} \rhd^* D\boxed{u = \boxed{t} \ @ \ A\boxed{t}}$$

10.2 Choosing the Best Derivation

A last point to deal with is non-confluence and in particular choosing between first simplifying the head connective on the right or the left of $*$ or $\vdash$. For instance in $\boxed{A} \vee B \vdash B \vee \boxed{A}$ one can apply either $\mathsf{L}\vee_1$ or $\mathsf{R}\vee_2$.

Interestingly, an answer is provided by *focusing*. It has been noticed by Andreoli [3] that, in bottom-up proof search, one should apply the invertible logical rules first. In our framework, this translates into first applying the invertible rewrite rules (the ones marked by a *). In the case of the example above, this means performing $\mathsf{L}\vee_1$ first, which leads to the following behavior:

$$\boxed{A} \vee B \ \vdash \ B \vee \boxed{A} \rhd^* B \Rightarrow B \vee A.$$

This is indeed the "right" choice, since applying $\mathsf{R}\vee_2$ first would lead to a dead-end:

$$\boxed{A} \vee B \ \vdash \ B \vee \boxed{A} \rhd^* B \vee (B \Rightarrow A).$$

When two invertible rules can be applied, the order is irrelevant. There are cases where two non-invertible rules can be applied. The vast majority of them commute in terms of provability, but not necessarily in the shape of the resulting formula[9]. Therefore our specification still leaves room for some choices. Currently, we have a heuristic prioritizing of the rules that sticks to what is presented in examples. One could also choose to leave the disambiguation to the user, e.g. by looking at the *orientation* of drag-and-drops. This is the solution chosen in [7] and [9].

11 A More Advanced Example

It is too early to perform a detailed case study comparing our approach to interactive theorem proving with others – tactic based, declarative, *etc.*.. This is due primarily to the fact that our prototype is not mature enough; it cannot handle lemmas and implements a limited formalism. However some examples allow to get a glimpse of specificities and possible advantages of proofs by actions.

One such example is a small logical riddle, which we borrow from a textual educational system, Edukera [27]. One considers a population of people, with at least one element h, together with a single function Mother and one predicate Rich. The aim is to show that the two following assumptions are incompatible:

(1) $\forall x.\neg\mathsf{Rich}(x) \vee \neg\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(x)))$,
(2) $\forall x.\neg\mathsf{Rich}(x) \Rightarrow \mathsf{Rich}(\mathsf{Mother}(x))$.

The original goal thus corresponds to the illustration of figure 5.

It is quite natural to approach this problem in a forward manner, by starting from the hypotheses to establish new facts. And a first point illustrated by this example is that DnD actions allow to do this in a smooth and precise manner. A possible first step is to bring h to the first hypothesis, to obtain a new fact:

(3) $\neg\mathsf{Rich}(h) \vee \neg\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(h)))$.

Double clicking on this new fact yields two cases:

(4) $\neg\mathsf{Rich}(h)$,
(4') $\neg\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(h)))$.

Let us detail how one solves the second one.

By bringing $\neg\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(h)))$ on the premise of $\forall x.\boxed{\neg\mathsf{Rich}(x)} \Rightarrow \mathsf{Rich}(\mathsf{Mother}(x))$ one obtains

(6) $\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(\mathsf{Mother}(h))))$.

The next step is a good example where the DnD is useful. By bringing this new fact to the right-hand part of

(1) $\forall x.\neg\mathsf{Rich}(x) \vee \neg\boxed{\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(x)))}$

one immediately obtains a new fact

(7) $\neg\mathsf{Rich}(\mathsf{Mother}(h))$.

In other proof systems, this last step requires a somewhat intricate tactic line and/or writing down at least the statement of the new fact.

One can then finish the case by combining (7) and (2) which yields $\mathsf{Rich}(\mathsf{Mother}(\mathsf{Mother}(h)))$, which contradicts

[9] In fact the only rules which do not give equivalent results when commuted are the critical pairs $\mathsf{F}\vee_i$ / $\mathsf{F}\Rightarrow_2$ for $i \in \{1, 2\}$, as was noted independently in [9].

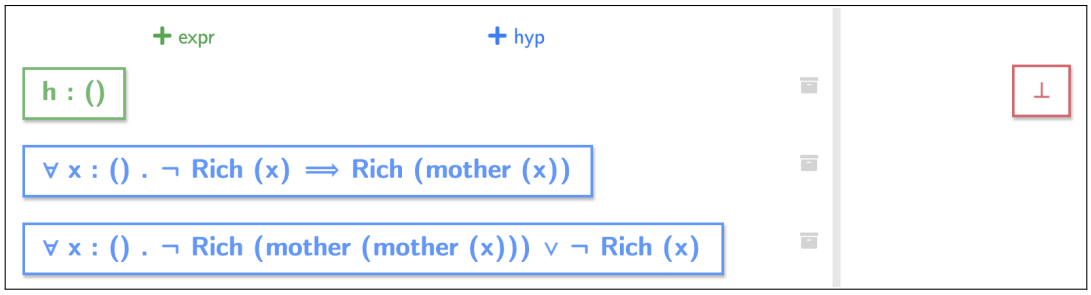

Figure 5. The beginning of an example due to Edukera.

(4′). These two last steps each correspond to a simple DnD. The other case, ¬Rich(h), is quite similar.

Such a simple example is not sufficient to provide significant metrics. Note however that once a user has understood the proof, the riddle is routinely solved in less than a minute in Actema, which seems out of reach for about any user in a tactic based prover. At least as important is the fact that the proof can be performed without typing any text, especially no intermediate statement.

12 Related Work

Subformula Linking. Although the primary motivation is very practical, it benefitted a lot from recent proof theory, especially Deep Inference. A key step was the discovery of the work of Kaustuv Chaudhuri [7] who had noticed how formula linking in deep inference could be used for proof construction in linear logic. His calculus of structures was very important for designing the rewrite rules which underly our system. In more recent work [9] he also deals with intuitionistic logic. Interestingly, some ideas like forward proof steps or the use of colors appeared independently in his and our work.

A difference is that we give the possibility to link first-order terms in addition to propositions, which is the basis for rewrite actions. One can imagine to design new kind of transformations in the future.

Window Inference. We have already mentioned Proof-by-Pointing, which was part of the CtCoq and Pcoq efforts [2] to design a graphical user interface for the Coq proof assistant. Another contemporary line of work was the one based on *window inference*, initially pioneered by P.J. Robinson and J. Staples. In [26], window inference is described as a general proof-theoretical framework, which aims to accomodate for the pervasive use of *equivalence transformations* throughout mathematics and computer science.

Window inference has been used both for general-purpose logics like HOL [14], and in more specialized settings like program refinement [15]. It naturally lends itself to integration in a graphical user interface ([19], [20]), where the user can *focus* on a subexpression by clicking on it. One is then presented with a new *graphical* window, holding the selected expression as well as an extended set of hypotheses exposing information inferrable from the context of the expression. The user can pick from a list of valid transformations to be applied to the expression, before closing the window. This propagates the transformations to the parent window by replacing the old subexpression by the new one, without modifying the surrounding context.

This process is quite reminiscent of the rewriting produced by our DnD actions. One key difference is that window inference rules can be applied stepwise, while we choose to hide the sequence of rules that justifies a DnD. The window inference approach gives to the user a precise control of the transformations to be performed and thus could inspire interesting extensions of our work.

Tangible Functional Programming. We noticed an interesting connection with the work of Conal Elliott on tangible functional programming [11]. His concept of *deep application* of λ-terms seems related to the notion of subformula linking, when viewing function and product types as implications and conjunctions through the formulae-as-types interpretation. He also devised a system of basic combinators which are composed sequentially to compute the result of a DnD, though it follows a more complex dynamic than our rewrite rules. Even if the mapping between proofs and programs is not exact in this case, it suggests a possible interesting field of application for the Curry-Howard correspondance, in the realm of graphical proving/programming environments.

Other Gestural Proof Systems. There are other proof systems which include drag-and-drop features. Two of them are the KeY Prover [1] and TAS [20]. TAS is a window inference system tailored for program refinement, and uses DnD actions between an expression and a transformation, in order to apply the latter to the former. As for the KeY Prover, its usage of DnD overlaps only a very small portion of usecases that we hinted at in section 11, namely the instantiation of quantifiers with objects.

We can also mention the recent work of Zhan et al. [29]. They share with us the vision of a proof assistant mainly driven by gestural actions, which requires far less textual

inputs from the user. However, they only consider point-and-click actions, and rely on a text-heavy presentation at two levels:

1. the proof state, which is a structured proof text in the style of Isar [24];
2. the proof commands, which can only be performed through choices in textual menus.

Explicit Proof Objects. Finally let us mention various recent implementations proposing various ways to construct proofs graphically: Building Blocks [17], the Incredible Proof Machine [5], Logitext[10] and Click & coLLecT [6]. But these systems focus more on explicating the proof object than on making its construction easier.

13 Conclusion and Perspectives

This work started as a very practical effort. Discovering and understanding the links with more theoretically grounded approaches, and especially deep inference, made us aware that there may be more proof theoretical depth to this idea than we first thought. But, most importantly, adapting the logical rules and tools of deep inference to the practical question we encountered, allowed us to structure our proposal and to define the "right" behavior for the system. We were able to extend the deep inference approach to the use of equalities 6, which may be an originality of this work. It seems imaginable to proceed similarly with other mathematical relations.

More generally, we hope that our treatment of equality can be the start for providing graphical or gestural tools to perform algebraic transformations of expressions (be there in the conclusion or in hypotheses). As mentioned above, Window Inference could serve as an inspiration here. This seems promising to us, since describing such a transformation is notoriously tedious when using textual commands.

Even a small prototype allowed us to experiment on some non-trivial examples and to make some first encouraging experiences. In various cases, like the one described in section 11, we have observed shorter or more straightforward proofs than in textual provers. Another nice point is that some syntactical details, like the name of proof tactics become irrelevant in the gestural setting. More generally, we feel that using such a system, one may indeed develop a good intuition for the behavior of the logical items. But this is obviously a user interface or user experience question which is too early to quantify. Also, some novel questions appear when implementing such a graphical system: what are the good user interface choices, how to obtain a good look-and-feel, what visual feedback the system should provide…

On the other hand, we should acknowledge that certain styles of proofs, where a large number of subcases can be immediately solved through the same short textual tactic sequence, may be less well suited for the gestural approach (the SSReflect [13] dialect for Coq is very well suited for such cases).

Among future lines of work, it will be interesting to explore how some automation fits into this framework. One example is the *point-and-shoot* paradigm of [4]. But the DnD feature could open up new possibilities, like having the system perform some automated deduction to prove equivalences or implications between the two squared formulas (which would thus no longer be required to be strictly equal or unifiable).

Another obvious and important point to be tackled next is to provide a smooth way to invoke a library of lemmas in a graphical proof. We believe this could raise some interesting questions.

An also promising line of work is to extend our approach to classical logic. A point being that the graphical setting could smoothly handle multiple conclusions with less spurious overhead than text commands.

An important difference with the days of the pioneering work on proof-by-pointing is that developers can now rely on powerful and standardized libraries, which make the construction of user interfaces much faster and easier, giving new room for experimentation and proposals. But bringing everything together in simple commands remains a complicated theoretical and development task.

[10] http://logitext.mit.edu/main

Acknowledgments

We are grateful to Kaustuv Chaudhuri and Dale Miller for stimulating discussions, and to Sébastien Najjar of the Dioxygen company for his work on the front-end of the Actema prototype. Useful comments and references were provided by anonymous referees.

References

[1] Wolfgang Ahrendt and Sarah Grebing. 2016. Using the KeY Prover. In *Deductive Software Verification – The KeY Book*, Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich (Eds.). Vol. 10001. Springer International Publishing, Cham, 495–539. https://doi.org/10.1007/978-3-319-49812-6_15 Series Title: Lecture Notes in Computer Science.

[2] Ahmed Amerkad, Yves Bertot, Loïc Pottier, and Laurence Rideau. 2001. *Mathematics and Proof Presentation in Pcoq.* Technical Report RR-4313. INRIA. https://hal.inria.fr/inria-00072274

[3] Jean-Marc Andreoli. 1992. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation* 2, 3 (1992), 297–347. https://academic.oup.com/logcom/article-lookup/doi/10.1093/logcom/2.3.297

[4] Yves Bertot, Gilles Kahn, and Laurent Théry. 1994. Proof by pointing. In *Theoretical Aspects of Computer Software*, Masami Hagiya and John C. Mitchell (Eds.). Vol. 789. Springer Berlin Heidelberg, 141–160. https://doi.org/10.1007/3-540-57887-0_94 Series Title: Lecture Notes in Computer Science.

[5] Joachim Breitner. 2016. Visual Theorem Proving with the Incredible Proof Machine. In *Interactive Theorem Proving*, Jasmin Christian

Blanchette and Stephan Merz (Eds.). Vol. 9807. Springer International Publishing, 123–139. https://doi.org/10.1007/978-3-319-43144-4_8 Series Title: Lecture Notes in Computer Science.

[6] Etienne Callies and Olivier Laurent. 2021. Click and coLLecT An Interactive Linear Logic Prover. In *5th International Workshop on Trends in Linear Logic and Applications (TLLA 2021)*. Rome (virtual), Italy. https://hal-lirmm.ccsd.cnrs.fr/lirmm-03271501

[7] Kaustuv Chaudhuri. 2013. Subformula Linking as an Interaction Method. In *Interactive Theorem Proving*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Vol. 7998. Springer Berlin Heidelberg, 386–401. https://doi.org/10.1007/978-3-642-39634-2_28 Series Title: Lecture Notes in Computer Science.

[8] Kaustuv Chaudhuri. 2020. Interactive Proof Building with Direct Manipulation for Linear Logic (and Cousins). (2020). Invited Talk at the *Linearity & TLLA* workshop.

[9] Kaustuv Chaudhuri. 2021. Subformula Linking for Intuitionistic Logic with Application to Type Theory. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 200–216. https://doi.org/10.1007/978-3-030-79876-5_12

[10] Roberto Di Cosmo. 1995. *Isomorphisms of types: from λ-calculus to information retrieval and language design*. Birkhauser. http://www.ens.fr/users/dicosmo/Publications/ISObook.html ISBN-0-8176-3763-X.

[11] Conal M. Elliott. 2007. Tangible Functional Programming. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming* (Freiburg, Germany) *(ICFP '07)*. Association for Computing Machinery, New York, NY, USA, 59–70. https://doi.org/10.1145/1291151.1291163

[12] Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1 – 101. https://doi.org/10.1016/0304-3975(87)90045-4

[13] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. 2016. *A Small Scale Reflection Extension for the Coq system*. Research Report RR-6455. Inria Saclay Ile de France. https://hal.inria.fr/inria-00258384

[14] J. Grundy. 1991. Window Inference In The HOL System. In *1991 International Workshop on the HOL Theorem Proving System and Its Applications*. 177–189. https://doi.org/10.1109/HOL.1991.596285

[15] Jim Grundy. 1992. A Window Inference Tool for Refinement. In *5th Refinement Workshop*, Cliff B. Jones, Roger C. Shaw, and Tim Denvir (Eds.). Springer London, London, 230–254.

[16] Alessio Guglielmi. 1999. *A Calculus of Order and Interaction*. Technical Report. Technische Universität Dresden. https://www.researchgate.net/publication/2807151_A_Calculus_of_Order_and_Interaction

[17] Sorin Lerner, Stephen R. Foster, and William G. Griswold. 2015. Polymorphic Blocks: Formalism-Inspired UI for Structured Connectors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, Bo Begole, Jinwoo Kim, Kori Inkpen, and Woontack Woo (Eds.). ACM, 3063–3072. https://doi.org/10.1145/2702123.2702302

[18] Chuck Liang and Dale Miller. 2009. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science* 410, 46 (2009), 4747–4768. https://doi.org/10.1016/j.tcs.2009.07.041

[19] Thomas Långbacka, Rimvydas Rukšėnas, and Joakim von Wright. 1995. TkWinHOL: A tool for Window Inference in HOL. In *Higher Order Logic Theorem Proving and Its Applications (Lecture Notes in Computer Science)*, E. Thomas Schubert, Philip J. Windley, and James Alves-Foss (Eds.). Springer, Berlin, Heidelberg, 245–260. https://doi.org/10.1007/3-540-60275-5_69

[20] Christoph Lüth and Burkhart Wolff. 2000. TAS — A Generic Window Inference System. In *Theorem Proving in Higher Order Logics*, Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Mark Aagaard, and John Harrison (Eds.). Vol. 1869. Springer Berlin Heidelberg, Berlin, Heidelberg, 406–423. https://doi.org/10.1007/3-540-44659-1_25 Series Title: Lecture Notes in Computer Science.

[21] Alberto Martelli and Ugo Montanari. 1982. An Efficient Unification Algorithm. *ACM Trans. Program. Lang. Syst.* 4, 2 (April 1982), 258–282. https://doi.org/10.1145/357162.357169

[22] Dale A. Miller. 1987. A compact representation of proofs. *Studia Logica* 46 (1987), 347–370.

[23] Robin Milner. 1984. The use of machines to assist in rigorous proof. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 312, 1522 (1984), 411–422. https://doi.org/10.1098/rsta.1984.0067 arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.1984.0067

[24] Tobias Nipkow. 2002. Structured Proofs in Isar/HOL. In *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers (Lecture Notes in Computer Science, Vol. 2646)*, Herman Geuvers and Freek Wiedijk (Eds.). Springer, 259–278. https://doi.org/10.1007/3-540-39185-1_15

[25] Cyrus Omar, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. 2021. Filling typed holes with live GUIs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 511–525. https://doi.org/10.1145/3453483.3454059

[26] Peter J. Robinson and John Staples. 1993. Formalizing a Hierarchical Structure of Practical Mathematical Reasoning. *Journal of Logic and Computation* 3, 1 (Feb. 1993), 47–61. https://doi.org/10.1093/logcom/3.1.47

[27] Benoit Rognier and Guillaume Duhamel. 2016. Présentation de la plateforme edukera. In *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*.

[28] Lutz Straßburger. 2019. The problem of proof identity, and why computer scientists should care about Hilbert's 24th problem. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 377, 2140 (2019), 20180038. https://doi.org/10.1098/rsta.2018.0038

[29] Bohua Zhan, Zhenyan Ji, Wenfan Zhou, Chaozhu Xiang, Jie Hou, and Wenhui Sun. 2019. *Design of point-and-click user interfaces for proof assistants*.