Proof Trees as Formulas

Claus-Peter Wirth

Dept. of Computer Sci., Universität des Saarlandes, D–66123 Saarbrücken, Germany
wirth@logic.at

Abstract. This draft is for internal communication and not completely self-contained.

1 Motivation

On the one hand, proof trees like sequent or tableau trees are very useful for formalizing a logic and for administrating the state of proof attempts in an implementation. Moreover, with proof trees it is easily possible to have a simple rule-based local means of specifying the possible deductive inference steps. Such a specification consisting of a set of local inference rules is not only well-suited for communicating a logic between human beings. It can also be used to include the logic into a logical framework, provided that the framework directly and conveniently supports the typical inferences of the logic.

On the other hand, in spite of their conceptual clearness, these proof trees still put off possible users of theorem provers who do not have an education that includes logic calculi, like many mathematicians and most engineers and students of computer science. Experience shows that while a growing group of possible users of a theorem prover are well acquainted with formulas expressing proof obligations and have a clear intuitive understanding of their semantics, they are not thinking in terms of a formal calculus, they do not want to think in terms of proof trees, and it takes them quite a while until they find it natural to formulate and understand proofs in the form of formal proof trees; and with the more efficient matrix representations of these proof trees, only a very small number of scientists feel comfortable.

Bringing both hands together, it would thus be nice to combine the advantages of proof trees with the possibility to present proof trees in the form of formulas that resemble the structure of the input formulas as closely as possible. Inspired by [1] (which combines and significantly extends ideas from [3–5]), we want to present here a first step into this direction.

For simplicity we restrict ourselves to classical logic and use the convenient sequent style presentation in this paper, although an implementation should use structure-sharing matrix techniques.

2 Introduction

The basic idea is roughly as follows.

Instead of applying inference rules to an original sequent (i.e. a disjunctive list of formulas), the user points at a certain position in a formula in order to set the focus of his proof to the subformula pointed at. The sequent calculus in the background applies the minimum number of calculus steps in order to make this position to a top position of a formula of a sequent of the proof tree and the resulting sequent is presented to the user in an appealing format. Optionally, all formulas in the sequent with the exception of the one pointed at may be further decomposed as long as α-, δ-, or γ-steps (according to Smullyan's classification) are applicable to them. Note that, while the β-steps cannot be handled this way because they split the sequent in twain, the automatic decomposition of α-, δ-, and γ-formulas cannot provide any difficulties when the δ-steps are liberalized ones, and when the γ-steps are limited to a multiplicity of 1 and use free variables. Now the user can globally instantiate free variables in order to get complementary formulas (in which case the whole sequent will be replaced with the single-formula sequent '`true`'), use equations &c. for contextually rewriting those formulas pointed at, or make a case analysis (implemented as Cut), or replace a redundant formula pointed at with the formula '`false`'. He can also set additional pointers to subformulas in order to trigger further decomposing calculus steps.

So far this looks very much like a traditional sequent calculus with a nice mouse-clicking user interface. The additional feature here, however, is the possibility to remove the previously set pointers either completely or otherwise to a parent position. In this case, the sequent calculus steps are undone in a way that respects the changes that have taken place between setting and removing the pointer. E.g., if a sequent has turned into `true`, this branch of the proof tree will be cut off when undoing the sequent calculus steps. A proof is successfully finished when all pointers have been removed and the current sequent reads '`true`'. If the result is not '`true`' because the original formula was not valid, our proof attempt may nevertheless be useful because the resulting sequent is equivalent to the input sequent, and hopefully simpler.

3 Automatic Simplification

We define a *sequent* to be a list of formulas. Note that we will make use of the ordering of the formulas in the list insofar as the number of a formula—counting from right to left—will be used to identify it even after transformations and addition of new formulas to the left end of the sequent.

We use upper case Greek letters to denote sequents, upper case Latin letters to denote formulas, and lower case Latin letters to denote terms.

The *conjugate* of a formula A (written: $\overline{A}$) is the formula B if A is of the form $\neg B$, and the formula $\neg A$ otherwise. In this paper we add one additional exception for formulas of the form $(A_0 \vee \ldots \vee A_{n+1})$ in which the '$\vee$' is annotated to be *soft* w.r.t. conjugation, as introduced by our γ-rules of Section 4.4. In this case we recursively define $\overline{(A_0 \vee \ldots \vee A_{n+1})} = \left(\overline{A_0} \wedge \ldots \wedge \overline{A_{n+1}}\right)$, where also the '$\wedge$' is annotated to be soft, with an analogous definition of the conjugate.

We assume that formulas are immediately normalized w.r.t. the following confluent and terminating rewrite system. Note that, although preservation of the original formula structure is very important for user interaction and is a major subject of this paper, we are only interested in preservation of the formula structure *modulo* a simple form of simplification.

$$\boxed{\begin{array}{rcl}\mathsf{false} \lor A & \longrightarrow & A \\ A \lor \mathsf{false} & \longrightarrow & A \\ \mathsf{true} \lor A & \longrightarrow & \mathsf{true} \\ A \lor \mathsf{true} & \longrightarrow & \mathsf{true} \\ A \lor A & \longrightarrow & A \\ A \lor \neg A & \longrightarrow & \mathsf{true} \\ \neg A \lor A & \longrightarrow & \mathsf{true}\end{array}} \quad \boxed{\begin{array}{rcl}\mathsf{false} \land A & \longrightarrow & \mathsf{false} \\ A \land \mathsf{false} & \longrightarrow & \mathsf{false} \\ \mathsf{true} \land A & \longrightarrow & A \\ A \land \mathsf{true} & \longrightarrow & A \\ A \land A & \longrightarrow & A \\ A \land \neg A & \longrightarrow & \mathsf{false} \\ \neg A \land A & \longrightarrow & \mathsf{false}\end{array}} \quad \boxed{\begin{array}{rcl}\mathsf{false} \Rightarrow A & \longrightarrow & \mathsf{true} \\ A \Rightarrow \mathsf{false} & \longrightarrow & \neg A \\ \mathsf{true} \Rightarrow A & \longrightarrow & A \\ A \Rightarrow \mathsf{true} & \longrightarrow & \mathsf{true} \\ A \Rightarrow A & \longrightarrow & \mathsf{true} \\ A \Rightarrow \neg A & \longrightarrow & \neg A \\ \neg A \Rightarrow A & \longrightarrow & A\end{array}}$$

$$\boxed{\begin{array}{rcl}\mathsf{false} \Leftarrow A & \longrightarrow & \neg A \\ A \Leftarrow \mathsf{false} & \longrightarrow & \mathsf{true} \\ \mathsf{true} \Leftarrow A & \longrightarrow & \mathsf{true} \\ A \Leftarrow \mathsf{true} & \longrightarrow & A \\ A \Leftarrow A & \longrightarrow & \mathsf{true} \\ A \Leftarrow \neg A & \longrightarrow & A \\ \neg A \Leftarrow A & \longrightarrow & \neg A\end{array}} \quad \boxed{\begin{array}{rcl}\mathsf{false} = A & \longrightarrow & \neg A \\ A = \mathsf{false} & \longrightarrow & \neg A \\ \mathsf{true} = A & \longrightarrow & A \\ A = \mathsf{true} & \longrightarrow & A \\ A = A & \longrightarrow & \mathsf{true} \\ A = \neg A & \longrightarrow & \mathsf{false} \\ \neg A = A & \longrightarrow & \mathsf{false}\end{array}} \quad \begin{array}{c}\boxed{\begin{array}{rcl}\neg\mathsf{false} & \longrightarrow & \mathsf{true} \\ \neg\mathsf{true} & \longrightarrow & \mathsf{false} \\ \neg\neg A & \longrightarrow & A\end{array}} \\ \\ \boxed{(t{=}t) \longrightarrow \mathsf{true}}\end{array}$$

Actually, we write $\land$ and $\lor$ as lists of associative operators and the last three rewrite rules for each of these operators are to be understood modulo associativity and commutativity.

Moreover, the resulting sequents are always immediately rewritten at most one step via

$$\begin{array}{rcl}\Gamma\ \mathsf{true}\ \Pi & \longrightarrow & \mathsf{true} \\ \Gamma\ A\ \Delta\ \neg A\ \Pi & \longrightarrow & \mathsf{true} \\ \Gamma\ \neg A\ \Delta\ A\ \Pi & \longrightarrow & \mathsf{true}\end{array}$$

The latter two rules should also apply when the two occurrences of A actually differ but can be unified by setting free γ-variables that do not occur in other leaf sequents. They even should trigger automatic α-, γ-, and δ-decomposition in order to be applicable.

Finally, we rewrite inference steps in proof trees according to the confluent and terminating rewrite system:

$$\begin{array}{ccc}\dfrac{\Gamma}{\mathsf{true}} & \longrightarrow & \dfrac{\mathsf{true}}{} \\ \dfrac{\Gamma}{\Pi_0\ \cdots\ \Pi_{m-1}\quad \mathsf{true}\quad \Delta_0\ \cdots\ \Delta_{n-1}} & \longrightarrow & \dfrac{\Gamma}{\Pi_0\ \cdots\ \Pi_{m-1}\quad \Delta_0\ \cdots\ \Delta_{n-1}}\end{array}$$

Note that in the last rewrite rule each of the 'Π_i', 'Δ_j', 'true' denotes a separate sequent, not a part of a long concatenated sequent. Note that in the good old days when trees grew upwards, Gerhard Gentzen would have inverted the inference rules such that passing the line means consequence. In our case, trees grow downwards and passing the line principally means reduction, but actually equivalence transformation.

We will call this whole rewrite system *automatic simplification*.

While the latter tree rewriting part of automatic simplification may be hard wired, the formula rewriting part should be programmable by the logics programmer, but not by the user of the theorem prover.

4 Inference Rules

It should be clear that any inference rule now comes as a single decomposition rule ($\downarrow$) as usual, plus a list of rules ($\uparrow$) for recomposition whose applicability is tested in the precedence from left to right when the step has to be undone.

Note that the $\downarrow$ and $\uparrow$ do not indicate any direction of consequence. All our inference rules will be equivalence transformations.

In general, we will use "$\uparrow$ id" as an abbreviation for the rule

$$\uparrow \frac{\Xi'}{\Xi'}$$

that asks just to hand up the lower sequent as upper sequent on recomposition, provided that this rule is applicable, which is the case when there is exactly one lower sequent

4.1 α-Rules

Let us consider some α-rules first. As double negation does not occur according to automatic simplification, all our α-rules have at least two side formulas.

$$\downarrow \frac{\Gamma\ (A_0 \vee \ldots \vee A_{n+1})\ \Pi}{\Gamma\ A_0\ \ldots\ A_{n+1}\ \Pi} \qquad\qquad \uparrow \frac{\Xi'\ (A'_0 \vee \ldots \vee A'_{n+1})\ \Pi'}{\Xi'\ A'_0\ \ldots\ A'_{n+1}\ \Pi'}$$

Note that, due to automatic simplification, this means that when we undo the decomposition of an α-formula and the lower sequent has been simplified to 'true', we do not apply the $\uparrow$-rule since the rewriting of automatic simplification will already have removed the whole inference step.

Note that the recomposition step ($\uparrow$) will trigger automatic simplification for those A'_i that have become false.

In any case, on recomposition we only put the formula back to its old position in order to be able to preserve the structure of the original sequent when removing all pointers.

Technically, the whole decomposition and recomposition can be achieved the following way: When we apply the $\downarrow$-rule we annotate the inference step with $n{+}1$ and with the number that indicates the position (say p) of the principal formula say $(A_0 \vee \ldots \vee A_{n+1})$ in the upper sequent (numbering from right to left). When we apply the corresponding $\uparrow$-rule we remove the formulas number $p{+}n{+}1, \ldots, p$, join them with the operator $\vee$ into an $\vee$-formula, and insert it at position p. Thus, we require in the above rule as well as in the below rules that the number of formulas of Π is equal to the number of formulas of Π'.

In a matrix representation the technical realization would be as follows. $\Gamma\ (A_0 \vee \ldots \vee A_{n+1})\ \Pi$ would be represented as a list of positions (*path*) in the original sequent say $g_m, \ldots, g_1, a, h_p, \ldots, h_0$, and the new path would be

$$g_m, \ldots, g_1, a0, \ldots, a(n{+}1), h_p, \ldots, h_1,$$

representing the lower sequent.

More α-rules:

$$\downarrow \frac{\Gamma\ \neg(A_0 \wedge \ldots \wedge A_{n+1})\ \Pi}{\Gamma\ \overline{A_0}\ \ldots\ \overline{A_{n+1}}\ \Pi} \qquad\qquad \uparrow \frac{\Xi'\ \neg(\overline{A'_0} \wedge \ldots \wedge \overline{A'_{n+1}})\ \Pi'}{\Xi'\ A'_0\ \ldots\ A'_{n+1}\ \Pi'}$$

$$\downarrow\frac{\Gamma\ (A{\Rightarrow}B)\ \Pi}{\Gamma\ \overline{A}\ B\ \Pi} \qquad\qquad \uparrow\frac{\Xi'\ (\overline{A'}{\Rightarrow}B')\ \Pi'}{\Xi'\ A'\ B'\ \Pi'}$$

$$\downarrow\frac{\Gamma\ (A{\Leftarrow}B)\ \Pi}{\Gamma\ A\ \overline{B}\ \Pi} \qquad\qquad \uparrow\frac{\Xi'\ (A'{\Leftarrow}\overline{B'})\ \Pi'}{\Xi'\ A'\ B'\ \Pi'}$$

4.2 Liberalized δ-Rules

$$\downarrow\frac{\Gamma\ \ \forall x.\,A\ \ \Pi}{\Gamma\ \ A\{x{\mapsto}x^{\delta}\}\ \ \Pi}\ \ \mathcal{V}_{\mathrm{free}}(A)\times\{x^{\delta}\} \qquad\qquad \uparrow\mathrm{id}$$

Note that this rule requires a semantics for free variables (as the one in [7]) to make sense. The x^{δ} is a new free δ-variable, which serves as a special parameter in an equivalence transformation. The $\mathcal{V}_{\mathrm{free}}(A)\times\{x^{\delta}\}$ to the right has to be added to the current variable-condition, which takes care of the soundness. Roughly speaking, this means that the free γ-variables in A must not be instantiated with terms that contain x^{δ}.

More γ-rules:

$$\downarrow\frac{\Gamma\ \ \neg\exists x.\,A\ \ \Pi}{\Gamma\ \ \overline{A\{x{\mapsto}x^{\delta}\}}\ \ \Pi}\ \ \mathcal{V}_{\mathrm{free}}(A)\times\{x^{\delta}\} \qquad\qquad \uparrow\mathrm{id}$$

4.3 Unconditional Contextual Rewriting

Let s and t be either both terms (of the same type) or both formulas. Assume that the formula B is listed in the sequent $\Gamma\Pi$. Let $A[t]$ denote the formula $A[s]$ with some occurrences of s replaced with t. Now we can apply the rule

$$\downarrow\frac{\Gamma\ A[s]\ \Pi}{\Gamma\ A[t]\ \Pi} \qquad\qquad \uparrow\mathrm{id}$$

in any of the following cases:

1. B is one of the formulas $(s{\neq}t)$ or $(t{\neq}s)$.
2. B is s and t is false.
3. B is $\neg s$ and t is true.

The latter two kinds of rewriting should actually be part of automatic simplification for the case that $A[s]$ is the formula focused on.

4.4 γ-Rules

$$\downarrow\frac{\Gamma\ \ \exists x.\,A\ \ \Pi}{\Gamma\ \ \exists x.\,A\ \ A\{x{\mapsto}x^{\gamma}\}\ \ \Pi} \qquad\qquad \uparrow\frac{\Xi'\ \ (A'_0{\vee}A'_1)\ \ \Pi'}{\Xi'\ \ A'_0\ \ A'_1\ \ \Pi'}$$

Note that in a matrix representation or indexed formula tree we have to change the subformula $\exists x.\,A$ destructively into $(\exists x.\,A \vee A\{x{\mapsto}x^\gamma\})$ before we then go down into $A\{x{\mapsto}x^\gamma\}$.

More γ-rules:

$$\downarrow\frac{\Gamma \quad \neg\forall x.\,A \quad \Pi}{\Gamma \quad \neg\forall x.\,A \quad \overline{A\{x{\mapsto}x^\gamma\}} \quad \Pi} \qquad\qquad \uparrow\frac{\Xi' \quad (A'_0{\vee}A'_1) \quad \Pi'}{\Xi' \quad A'_0 \quad A'_1 \quad \Pi'}$$

Note that the newly introduced '$\vee$' are *soft w.r.t. conjugation* in the sense that the complement operator should turn it into a soft '$\wedge$' and recurse on the arguments. Otherwise the formula say

$$\forall x.\,A \;\Rightarrow\; B$$

may look like

$$\neg(\neg\forall x.\,A \vee A') \;\Rightarrow\; B$$

after setting and subsequently removing a focus to A, whereas it should read

$$\left(\forall x.\,A \wedge \overline{A'}\right) \;\Rightarrow\; B.$$

4.5 β-Rules

A naïve version of a β-rule is the following one:

$$\downarrow\frac{\Gamma\,(A_0{\wedge}\ldots{\wedge}A_{m+1})\,\Pi}{\Gamma\,A_0\,\Pi \;\;\cdots\;\; \Gamma\,A_{m+1}\,\Pi} \qquad\qquad \uparrow\text{id} \qquad\qquad \uparrow\frac{\Gamma' \;\left(A'_0{\wedge}\ldots{\wedge}A'_{m'}\right)\Pi'}{\Gamma'\,A'_0\,\Pi' \;\;\cdots\;\; \Gamma'\,A'_{m'}\,\Pi'}$$

Here we must point out a serious problem. The β-rules are the trouble-makers in logic. Not only do they (under assistance of the γ-rules) render classical first-order logic undecidable; they also made problems in Section 2 with automatic decomposition; and now they even may not admit structure preserving $\uparrow$-rules:

Note that the above $\uparrow$-rule does not tell us what to do when at least two of the lower sequents are not '`true`' and they differ in more than the former side-formulas, i.e. when they differ in Γ' or Π' !

When we only admit a single focus or a list of pointers that are on a single path in the original sequent, then we know that all transformations (besides instantiations and automatic simplification, which are global and change both sides simultaneously) will take place outside Γ and Π.

Another even more impractical solution with which we can also transform formulas in $\Gamma\Pi$ is to undo all proof steps in this branch of the proof tree on recombination unless the modified branches turned out to be '`true`'. (This can be realized simply by taking the upper sequent and applying all global variable instantiations to it.)

Note that it is not the problem here to combine two sequents. This can always be achieved according to

$$\uparrow\frac{\Gamma\;\left(\bigvee\Delta_1 \wedge \bigvee\Delta_2\right)}{\Xi_1 \qquad \Xi_2}$$

where Γ lists the formulas common to Ξ_1 and Ξ_2 and where Δ_1 and Δ_2 list the remaining formulas of Ξ_1 and Ξ_2, resp.. The problem, however, is that with such a $\uparrow$-β-rule we lose all the positional correspondences and, when we remove all pointers, end up with a completely different sequent, which has no structural correspondence to the input sequent, not even modulo simplification.

Thus, in order to be useful in practice, we need a solution that is a little more involved. We define the *intersection* and the *difference* of two sequents $A_m \ldots A_0$ and $B_n \ldots B_0$ $(m,n \in \mathbf{N})$ as the sequents $C_k \ldots C_0$ and $D_m \ldots D_0$, resp., where $k := \min\{m,n\}$, and for $i \preceq k$ and $j \preceq m$:

$$C_i := \begin{cases} A_i & \text{if } A_i = B_i \\ \mathsf{false} & \text{otherwise} \end{cases}$$

$$D_j := \begin{cases} A_j & \text{if } n \prec j \text{ or } B_j = \mathsf{false} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

The *intersection* of several sequents is their pairwise intersection, which is associative and commutative. The *formula* of the sequent $A_m \ldots A_0$ is the formula $(A_m \vee \ldots \vee A_0)$, normalized w.r.t. automatic simplification, and with the '$\vee$' being soft w.r.t. conjugation.

Note the following: Let Φ be the intersection of Γ with other sequents $\Pi_0, \ldots, \Pi_q$. Let Δ be the difference of Γ and Φ. Then $\Delta\Phi$ is equivalent to Γ. Thus, due to distributivity of $\vee$ over $\wedge$ we get the following β-rule:

$$\downarrow \frac{\Gamma\ (A_0 \wedge \ldots \wedge A_{m+1})\ \Pi}{\Gamma\, A_0\, \Pi \quad \cdots \quad \Gamma\, A_{m+1}\, \Pi}$$

$$\uparrow \mathrm{id} \qquad\qquad \uparrow \frac{\Gamma'' \ \ \big((A''_0 \wedge \ldots \wedge A''_{m'})\big)\ \Pi''}{\Xi'_0\ \Gamma'_0\ A'_0\ \Pi'_0 \quad \cdots \quad \Xi'_{m'}\ \Gamma'_{m'}\ A'_{m'}\ \Pi'_{m'}}$$

> where the numbers of formulas of each of Γ, Γ'_0, …, $\Gamma'_{m'}$, Γ'' as well as the numbers of formulas of each of Π, Π'_0, …, $\Pi'_{m'}$, Π'' are the same, resp., where $\Gamma''\Pi''$ is the intersection of $\Gamma'_0\Pi'_0$, …, $\Gamma'_{m'}\Pi'_{m'}$, and where A''_j is the formula of the sequent resulting from appending A'_j, Ξ'_j, and the difference of $\Gamma'_j\Pi'_j$ and $\Gamma''\Pi''$, for $j \preceq m'$.

Note that due to automatic simplification on recomposition, none of the lower sequents is true and we have $m' \succeq 0$.

With the same condition we get the following β-rule:

$$\downarrow \frac{\Gamma\ \neg(A_0 \vee \ldots \vee A_{m+1})\ \Pi}{\Gamma\ \overline{A_0}\ \Pi \quad \cdots \quad \Gamma\ \overline{A_{m+1}}\ \Pi}$$

$$\uparrow \mathrm{id} \qquad\qquad \uparrow \frac{\Gamma'' \ \ \neg\Big(\overline{A''_0} \vee \ldots \vee \overline{A''_{m'}}\Big)\ \Pi''}{\Xi'_0\ \Gamma'_0\ A'_0\ \Pi'_0 \quad \cdots \quad \Xi'_{m'}\ \Gamma'_{m'}\ A'_{m'}\ \Pi'_{m'}}$$

With the analogous conditions we get the following β-rules:

$$\downarrow \frac{\Gamma\ \neg(A_0 \Rightarrow A_1)\ \Pi}{\Gamma\, A_0\, \Pi \qquad \Gamma\ \overline{A_1}\ \Pi} \qquad \uparrow \mathrm{id} \qquad \uparrow \frac{\Gamma'' \ \ \neg\Big(A''_0 \Rightarrow \overline{A''_1}\Big)\ \ \Pi''}{\Xi'_0\ \Gamma'_0\ A'_0\ \Pi' \qquad \Xi'_1\ \Gamma'_1\ A'_1\ \Pi'_1}$$

$$\downarrow\frac{\Gamma\ \neg(A_0{\Leftarrow}A_1)\ \Pi}{\Gamma\ \overline{A_0}\ \Pi \qquad\quad \Gamma\ A_1\ \Pi} \qquad\qquad \uparrow\text{id} \qquad \uparrow\frac{\Gamma''\ \ \neg\left(\overline{A''_0}{\Leftarrow}A''_1\right)\ \ \Pi''}{\Xi'_0\ \Gamma'_0\ A'_0\ \Pi'_0 \qquad\quad \Xi'_1\ \Gamma'_1\ A'_1\ \Pi'_1}$$

Finally, note that there are more powerful, asymmetric versions of β-rules which have shown to be useful in practical applications, e.g. with the QUODLIBET system, [2]. We present only one for the $\wedge$-formula that folds down to the right:

$$\downarrow\frac{\Gamma\ (A_0{\wedge}A_1)\ \Pi}{\Gamma\ A_0\ \Pi \qquad\quad \Gamma\ (A_1{\Leftarrow}A_0)\ \Pi} \qquad\qquad \uparrow\text{id} \qquad \uparrow\frac{\Gamma''\ \ \left(A''_0{\wedge}A''_1\right)\ \ \Pi''}{\Xi'_0\ \Gamma'_0\ A'_0\ \Pi'_0 \qquad\quad \Xi'_1\ \Gamma'_1\ A'_1\ \Pi'_1}$$

4.6 Cut

$$\downarrow\frac{\Pi}{A\ \Pi \quad \overline{A}\ \Pi} \qquad\qquad \uparrow\text{id} \qquad\qquad \uparrow\frac{(A''_0{\wedge}A''_1)\ \ \Pi''}{\Xi'_0\ \Pi'_0 \qquad\quad \Xi'_1\ \Pi'_1}$$

where the numbers of formulas of each of Π, Π'_0, Π'_1, Π'' are the same, where Π'' is the intersection of Π'_0 and Π'_1, and where A''_j is the formula of the sequent resulting from appending A'_j, Ξ'_j, and the difference of Π'_j and Π'', for $j \preceq 1$.

4.7 Lemma Application

Suppose we want to apply the sequent $A_0\ \cdots\ A_m$ $(m{\in}\mathbf{N})$ as a lemma. Note that this will only be an equivalence transformation relative to the validity of the lemma. We present here a version with maximal downfolding (as used in QUODLIBET, e.g.). Every level of user-controlled downfolding is obviously no problem.

$$\downarrow\frac{\Pi}{\overline{A_0}\ \Pi \qquad \overline{A_1}\ A_0\ \Pi\ \ \cdots\ \ \overline{A_m}\ A_{m-1}\ldots A_0\ \Pi}$$

$$\uparrow\text{id} \qquad\qquad\qquad \uparrow\frac{(A''_0{\wedge}\ldots{\wedge}A''_{m'})\ \ \Pi''}{\Xi'_0\ \Pi'_0\ \ \cdots\ \ \Xi'_{m'}\ \Pi'_{m'}}$$

where the numbers of formulas of each of Π, Π'_0, $\ldots$, $\Pi'_{m'}$, Π'' are the same, where Π'' is the intersection of Π'_0, $\ldots$, $\Pi'_{m'}$, and where A''_j is the formula of the sequent resulting from appending A'_j, Ξ'_j, and the difference of Π'_j and Π'', for $j \preceq m'$.

5 Examples

5.1 The Example of [3]

Let us replay the example of [3] here. We use automatic α-decomposition of not focused formulas. We start with the one formula sequent

$$\neg((A \wedge B) \vee (A \Rightarrow B)) \Rightarrow \neg((B \vee C) \Rightarrow (A \vee C))$$

Clicking to the $\wedge$ we get $\quad (A \boxed{\wedge} B),\ \neg A,\ B,\ \neg((B \vee C) \Rightarrow (A \vee C))$

A user might prefer a representation of the form

Focused: $(A \boxed{\wedge} B)$ $\qquad$ Rewrite Rules: $A = \mathsf{true}$, $B = \mathsf{false}$, ...

No matter how it is presented to the user, by the automatic simplification part of unconditional contextual rewriting we immediately get

$$(A \boxed{\wedge} \mathsf{false}),\ \neg A,\ B,\ \neg((B \vee C) \Rightarrow (A \vee C))$$

which is immediately simplified by automatic simplification into

$$\boxed{\mathsf{false}},\ \neg A,\ B,\ \neg((B \vee C) \Rightarrow (A \vee C))$$

Removing the single set pointer (focus) we get

$$\neg(\mathsf{false} \vee (A \Rightarrow B)) \Rightarrow \neg((B \vee C) \Rightarrow (A \vee C))$$

which is immediately simplified to $\quad \neg(A \Rightarrow B) \Rightarrow \neg((B \vee C) \Rightarrow (A \vee C))$

Next we focus on the last implication $\quad \neg A,\ B,\ \neg((B \vee C) \boxed{\Rightarrow} (A \vee C))$

and automatic simplification immediately produces $\quad \neg A,\ B,\ \boxed{\mathsf{false}}$

Removing the single set pointer (focus) we get $\quad \neg(A \Rightarrow B) \Rightarrow \mathsf{false}$

which is automatically simplified to $\quad A \Rightarrow B$

This series of equivalence transformations closely follows that of the example of [3] with exactly the same result. The slight differences improve the user-friendliness, namely that the focused formula does not jump out of its position and is always presented with positive polarity (the '$\neg$' does not disappear with the focus on '$\Rightarrow$' above).

Moreover, in our case here, we have an additional possibility, namely not to remove the the first pointer before setting the second one. This results in

$$\boxed{\mathsf{false}},\ \neg A,\ B,\ \neg((B \vee C) \boxed{\Rightarrow} (A \vee C))$$

and then in $\quad \boxed{\mathsf{false}},\ \neg A,\ B,\ \boxed{\mathsf{false}}$

This seems more natural because the basic idea of the whole transformation is to use $\neg A$ and B for rewriting everywhere.

5.2 First-Order Example

Suppose we want to prove $$\forall v.\left(\begin{array}{c}\forall u.\ \mathsf{Q}(u,v)\\ \vee\ \forall w.\ \mathsf{Q}(v,w)\end{array}\right)\Rightarrow\exists x.\ \forall y.\ \mathsf{Q}(x,y)$$

When the user clicks on the last Q he gets:

$$\neg\forall v.\left(\begin{array}{c}\forall u.\ \underline{\mathsf{Q}(u,v)}\\ \vee\ \forall w.\ \underline{\mathsf{Q}(v,w)}\end{array}\right),\ \exists x.\ \forall y.\ \mathsf{Q}(x,y),\ \boxed{\mathsf{Q}(x_0^\gamma,y_0^\delta)}$$

with the variable-condition $x_0^\gamma \xrightarrow[R]{} y_0^\delta$. Note that the two occurrences of Q that are of complementary polarity to the Q in the focus should be highlighted by the system. We have underlined them here. So suppose the user clicks the first one. He gets:

$$\neg\forall u.\ \underline{\mathsf{Q}(u,v_0^\gamma)},\ \neg\boxed{\mathsf{Q}(u_0^\gamma,v_0^\gamma)},$$
$$\neg\forall v.\left(\begin{array}{c}\forall u.\ \underline{\mathsf{Q}(u,v)}\\ \vee\ \forall w.\ \underline{\mathsf{Q}(v,w)}\end{array}\right),\ \exists x.\ \forall y.\ \underline{\mathsf{Q}(x,y)},\ \boxed{\mathsf{Q}(x_0^\gamma,y_0^\delta)}$$

Note that the system does not apply any unification right in the moment because v_0^γ occurs also in another sequent, namely in

$$\neg\forall w.\ \underline{\mathsf{Q}(v_0^\gamma,w)},\ \neg\forall v.\left(\begin{array}{c}\forall u.\ \underline{\mathsf{Q}(u,v)}\\ \vee\ \forall w.\ \underline{\mathsf{Q}(v,w)}\end{array}\right),\ \exists x.\ \forall y.\ \mathsf{Q}(x,y),\ \boxed{\mathsf{Q}(x_0^\gamma,y_0^\delta)}$$

Nevertheless, the user cannot do anything anymore because the last sequent-rewriting rule of automatic simplification (cf. Section 3) triggers automatic γ-decomposition of the last formula, removing this branch. Then—the v_0^γ now occurring only in one sequent—the system unifies the two focuses and replies with `true`, which the user only sees when the proof is done. Note that, although this proof looks very simple because it took only two quite arbitrary clicks by the user, the theorem is already quite hard to prove in natural deduction because it is not intuitionistically valid.

Acknowledgments: I am indebted to Serge Autexier who got me down the trees.

References

[1] Serge Autexier (2002). *Theory and Architecture of an Hierarchical Contextual Reasoning Framework.* Ph.D. thesis. To appear.

[2] Ulrich Kühler (2000). *A Tactic-Based Inductive Theorem Prover for Data Types with Partial Operations.* Ph.D. thesis, Infix, Sankt Augustin.

[3] Leonard G. Monk (1988). *Inference Rules Using Local Contexts.* J. Automated Reasoning **4**, pp. 445–462, Kluwer.

[4] Peter J. Robinson, John Staples (1993). *Formalizing a Hierarchical Structure of Practical Mathematical Reasoning.* J. Logic Computat. **3**, pp. 47–61, Oxford Univ. Press.

[5] Lincoln A. Wallen (1990). *Automated Proof Search in Non-Classical Logics.* MIT Press. Cf., however, `http://www.ags.uni-sb.de/~cp/p/wallen/all.txt` for some obsolete aspects of this fascinating book.

[6] Claus-Peter Wirth (1997). *Positive/Negative-Conditional Equations: A Constructor-Based Framework for Specification and Inductive Theorem Proving.* Ph.D. thesis, Verlag Dr. Kovač, Hamburg.

[7] Claus-Peter Wirth (2000). *Descente Infinie + Deduction.* Report 737/2000, FB Informatik, Univ. Dortmund. Extd. version, Dec. 13, 2002. `http://www.ags.uni-sb.de/~cp/p/tab99/new.html (Feb. 01, 2002).`

5.3 Conditional Rewriting?

Let $D[C]$ denote a huge formula in that C has a single positive occurrence. Consider
$$\neg(A \Rightarrow (B \Rightarrow C)),\ D[C]$$
In the system of [1] it is possible to use conditional rewriting with implications to rewrite C to to $(A \wedge B)$, yielding something like $D[(A \wedge B)],\ \neg(A \Rightarrow (B \Rightarrow C)),\ D[C]$
We could do this step only in case of the original sequent having the equivalent form
$$\neg((A \wedge B) \Rightarrow C),\ D[C]$$
These things seem wondrous.

5.4 Rewriting with β-blocks

Let C and D be formulas. Assume that the formula B is listed in the sequent $\Gamma\Pi$. Let $A[D]$ denote the formula $A[C]$ with some occurrences of C replaced with D. Now we can apply the rule
$$\downarrow\frac{\Gamma\ A[C]\ \Pi}{\Gamma\ A[D]\ A[C]\ \Pi} \qquad\qquad \uparrow\frac{\Xi'\ (A'_0 \vee A'_1)\ \Pi'}{\Xi'\ A'_0\ A'_1\ \Pi'}$$
in any of the following cases:

1. B is a β-block for $\{C^-, D^+\}$ and all occurrences of C in $A[C]$ have positive polarity.
2. B is a β-block for $\{C^+, D^-\}$ and all occurrences of C in $A[C]$ have negative polarity.

β-blocks are inductively defined as follows: $\neg(D \Rightarrow C)$, $\neg(C \Leftarrow D)$, $(D \wedge \overline{C})$, $(\overline{C} \wedge D)$ $\neg(\overline{D} \vee C)$ and $\neg(C \vee \overline{D})$, are β-blocks for C^- and D^+.

Every formula A can be seen as the singleton β-set $\{A^+\}$.

Then we may rewrite elements of β-sets as follows to several new elements as follows:
$$\boxed{\begin{array}{rcl}
(A_0 \wedge \ldots \wedge A_{n+1})^+ & \to & A_0^+, \ldots, A_{n+1}^+ \\
(A_0 \vee \ldots \vee A_{n+1})^- & \to & A_0^-, \ldots, A_{n+1}^- \\
(A \Rightarrow B)^- & \to & A^+, B^- \\
(A \Leftarrow B)^- & \to & A^-, B^+ \\
(\neg A)^- & \to & A^+ \\
(\neg A)^+ & \to & A^-
\end{array}}$$
Now B is a β-block for the β-set M if $\{B^+\}$ and M rewrite to the same set. Actually, when rewriting $\{B^+\}$ (but not for M), we additionally may allow the following steps here:
$$\boxed{\begin{array}{rcl}
(A_0 \wedge \ldots \wedge A_{n+1})^- & \to & (A_0 \wedge \ldots \wedge A_{i-1} \wedge A_{i+1} \wedge \ldots \wedge A_{n+1})^- \\
(A_0 \vee \ldots \vee A_{n+1})^+ & \to & (A_0 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_{n+1})^+ \\
(A \Rightarrow B)^+ & \to & A^- \\
(A \Rightarrow B)^+ & \to & B^+ \\
(A \Leftarrow B)^+ & \to & A^+ \\
(A \Leftarrow B)^+ & \to & B^-
\end{array}}$$