Implementing the Clausal Normal Form Transformation with Proof Generation

Hans de Nivelle

Max Planck Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
nivelle@mpi-sb.mpg.de

Abstract. We explain how we intend to implement the clausal normal form transformation with proof generation. We present a convenient data structure for sequent calculus proofs, which will be used for representing the generated proofs. The data structure allows easy proof checking and generation of proofs. In addition, it allows convenient implementation of proof normalization, which is necessary in order to keep the size of the generated proofs acceptable.

1 Introduction

In [2], a method for generating explicit proofs from the clausal normal form transformation was presented, which does not make use of choice axioms. It is our intention to implement this method. In this paper we introduce the data structure for the representation of proofs that we intend to use, and we give a general algorithm scheme, with which one can translate formulas and obtain correctness proofs at the same time.

In [2], natural deduction was used for showing that it is in principle possible to generate explicit proofs. It is however in practice better to use sequent calculus, because sequent calculus allows proof reductions that reduce the size of generated proofs. In order to be able to keep the sizes of the resulting proofs acceptable, it is necessary to normalize proofs in such a way that repeated building up of contexts is avoided.

In the preceeding paper [1], which was still proposing to use choice axioms, it was explained how to do this in type theory. An intermediate calculus was introduced, called the *replacement calculus*, which allows for proof normalization. After normalization, the resulting proof could be translated into type theory through a simple replacement schema. If one uses sequent calculus instead of natural deduction, the standard reductions of sequent calculus can do the proof normalization. It turns out that proof normalization in the replacement calculus corresponds to a restricted form of cut elimination in sequent calculus. Therefore, if one uses sequent calculus instead of natural deduction, the replacement calculus can be omitted alltogether.

In the next section we introduce sequent calculus. After that, we introduce the data structure that we will use for representing sequent calculus proofs. Then we will give a general scheme for translating formulas and generating proofs at the same time. In the last section, we show that our sequent proof data structure is convenient for implementing the kind of proof reduction that we need.

2 Sequent Calculus

Definition 1. *A* sequent *is an object of form $\Gamma \vdash \Delta$, where both Γ and Δ are multisets.*

We give the rules of sequent calculus. We assume that α-equivalent formulas are not distinguished. We also give equality rules, although equality plays no rule in the CNF-transformation.

$$\text{(axiom)}\ \frac{}{\Gamma, A \vdash \Delta, A}$$

$$\text{(cut)}\ \frac{\Gamma, A \vdash \Delta \qquad \Gamma \vdash \Delta, A}{\Gamma \vdash \Delta}$$

Structural Rules:

$$\text{(weakening left)}\ \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \text{(weakening right)}\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}$$

$$\text{(contraction left)}\ \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \text{(contraction right)}\ \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}$$

Rules for the truth constants:

$$(\top\text{-left})\ \frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta} \qquad (\top\text{-right})\ \frac{}{\Gamma \vdash \Delta, \top}$$

$$(\bot\text{-left})\ \frac{}{\Gamma, \bot \vdash \Delta} \qquad (\bot\text{-right})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \bot}$$

Rules for $\neg$:

$$(\neg\text{-left})\ \frac{\Gamma \vdash \Delta, A}{\Gamma, \neg A \vdash \Delta} \qquad (\neg\text{-right})\ \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, \neg A}$$

Rules for $\wedge, \vee, \leftarrow, \leftrightarrow$:

$$(\wedge\text{-left})\ \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \qquad (\wedge\text{-right})\ \frac{\Gamma \vdash \Delta, A \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$$

$$(\vee\text{-left})\ \frac{\Gamma,\ A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \qquad (\vee\text{-right})\ \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$$

$$(\to\text{-left})\ \frac{\Gamma \vdash \Delta, A \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \qquad\qquad (\to\text{-right})\ \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}$$

$$(\leftrightarrow\text{-left})\ \frac{\Gamma, A \to B,\ B \to A \vdash \Delta}{\Gamma, A \leftrightarrow B \vdash \Delta}$$

$$(\leftrightarrow\text{-right})\ \frac{\Gamma \vdash \Delta, A \to B \qquad \Gamma \vdash \Delta, B \to A}{\Gamma \vdash \Delta, A \leftrightarrow B}$$

Rules for the quantifiers:

$$(\forall\text{-left })\ \frac{\Gamma, P[x := t] \vdash \Delta}{\Gamma, \forall x\ P \vdash \Delta} \qquad\qquad (\forall\text{-right })\ \frac{\Gamma \vdash \Delta, P[x := y]}{\Gamma \vdash \Delta, \forall x\ P}$$

$$(\exists\text{-left })\ \frac{\Gamma, P[x := y] \vdash \Delta}{\Gamma, \exists x\ P \vdash \Delta} \qquad\qquad (\exists\text{-right })\ \frac{\Gamma \vdash \Delta, P[x := t]}{\Gamma \vdash \Delta, \exists x\ P}$$

The t is an arbitrary term. The y is a variable which is not free in Γ, Δ, P

Rules for equality:

$$(\text{refl})\ \frac{}{\Gamma \vdash \Delta,\ t \approx t}$$

$$(\text{repl-left })\ \frac{t_1 \approx t_2,\ \Gamma[t_1] \vdash \Delta}{t_1 \approx t_2,\ \Gamma[t_2] \vdash \Delta} \qquad\qquad (\text{repl-right })\ \frac{t_1 \approx t_2,\ \Gamma \vdash \Delta[t_1]}{t_1 \approx t_2,\ \Gamma \vdash \Delta[t_2]}.$$

The last rules mean: If $t_1 \approx t_2$ appears among the premisses, then an arbitrary occurrence of t_1 can be replaced by t_2. The replacement can take place either on the left or on the right. Only one replacement at the same time is possible.

3 Proof Trees

We introduce a concise sequent calculus format, which allows for easy proof checking and implementation of proof reductions. It is closely related to the embedding of sequent calculus in LF, which is introduced in [5].

We first prove a simple lemma that shows that one should avoid explicitly mentioning the formulas occurring in the proof:

Lemma 1. *Consider the sequents* $(\neg\neg)^n A \vdash A$, *for* $n \geq 0$.

If one has a proof representation method that explicitly mentions the formulas in a sequent, then the proofs have size $O(n^2)$.

Proof. Because one will have to represent all subformulas
$A, \neg A, (\neg)^2 A, (\neg)^3 A, \ldots, (\neg\neg)^n A$.

Nevertheless, the proof has a length of only n steps. If one does not mention the formulas, one can obtain a representation of size n. In our representation, we avoid explicitly mentioning formulas by assigning labels to them. Whenever a new formula is constructed, it will be clear what the new formula is, from the way it is constructed, so that we will not have to mention it.

Definition 2. *We redefine a sequent as an object of form $\Gamma \vdash \Delta$, where both Γ and Δ are sets of labelled formulas. So we have $\Gamma = \{\alpha_1{:}A_1, \ldots, \alpha_p{:}A_p\}$ and $\Delta = \{\beta_1{:}B_1, \ldots, \beta_q{:}B_q\}$, where $\alpha_i = \alpha_j$ implies $i = j$ and $\beta_i = \beta_j$ implies $i = j$.*

In case there is no A', s.t. $\alpha{:}A' \in \Gamma$, the notation $\Gamma + \alpha{:}A$ denotes $\Gamma \cup \{\alpha{:}A\}$. Otherwise, $\Gamma + \alpha{:}A$ is undefined. (even when $A = A'$)

In case there is an A, s.t. $\alpha{:}A \in \Gamma$, the notation $\Gamma - \alpha$ denotes $\Gamma \backslash \{\alpha{:}A\}$. Otherwise $\Gamma - \alpha$ is not defined.

In case there is an A, s.t. $\alpha{:}A \in \Gamma$, the notation $\Gamma[\alpha]$ denotes A. Otherwise $\Gamma[\alpha]$ is not defined.

For Δ, we define $\Delta + \beta{:}B, \quad \Delta - \beta, \quad \Delta[\beta]$ in the same way as for Γ.

Proofs are checked *top-down*, i.e. from the goal sequent towards the axioms. For each node in the proof tree, the node states the label of the conslusion in the derived sequent, and what labels the premisses should receive in the child sequents. During checking, the conclusion is removed from the sequent (if it exists, and has the right form), and replaced by the children, after which proof checking continues.

Definition 3. *We recursively define* proof trees *and when a proof tree* accepts *a labelled sequent. In the following list, we implicitly assume that α, β are labels. We will omit the definedness conditions. So we will assume that $\Delta[\alpha] = \Delta[\beta]$ means: $F[\alpha]$ and $F[\beta]$ are both defined and $F[\alpha] = F[\beta]$.*

- $\mathrm{ax}(\alpha, \beta)$ *is a proof tree. It is a proof of $\Gamma \vdash \Delta$, if $\Gamma[\alpha]$ is an α-variant of $\Delta[\beta]$.*
- *If π_1, π_2 are proof trees, and A is a formula, then* $\mathrm{cut}(A, \pi_1, \alpha, \pi_2, \beta)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if π_1 is a proof of $\Gamma + \alpha{:}A \vdash \Delta$ and π_2 is a proof of $\Gamma \vdash \Delta + \beta{:}A$.*
- *If π is a proof tree, then* $\mathrm{weakenleft}(\alpha, \pi)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if π is a proof of $\Gamma - \alpha \vdash \Delta$.*
- *If π is a proof tree, then* $\mathrm{weakenright}(\beta, \pi)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if π is a proof of $\Gamma \vdash \Delta - \beta$.*
- *If π is a proof tree, then* $\mathrm{contrleft}(\alpha_1, \pi, \alpha_2)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if π is a proof of $\Gamma + \alpha_1{:}A[\alpha_2] \vdash \Delta$.*
- *If π is a proof tree, then* $\mathrm{contrright}(\beta_1, \pi, \beta_2)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if π is a proof of $\Gamma \vdash \Delta + \beta_1{:}\Delta[\beta_2]$.*
- *If π is a proof tree, then* $\mathrm{trueleft}(\alpha, \pi)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if $\Gamma[\alpha] = \top$, and π is a proof of $\Gamma - \alpha \vdash \Delta$.*

- trueright(β) *is a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta] = \top$.
- falseleft(α) *is a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha] = \bot$.
- falseright(β, π) *is a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta] = \bot$ *and* π *is a proof of* $\Gamma \vdash \Delta - \beta$.
- *If* π *is a proof tree, then* negleft(α, π, β) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $\neg A$, *and* π *is a proof of* $\Gamma \vdash \Delta + \beta{:}A$.
- *If* π *is a proof tree, then* negright(β, π, α) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $\neg A$, *and* π *is a proof of* $\Gamma + \alpha{:}A \vdash \Delta$.
- *If* π *is a proof tree and* $\alpha_1 \neq \alpha_2$, *then* andleft($\alpha, \pi, \alpha_1, \alpha_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $A \wedge B$, *and*
  π *is a proof of* $(\Gamma - \alpha) + \alpha_1{:}A + \alpha_2{:}B \vdash \Delta$.
- *If* π_1, π_2 *are proof trees, then* andright($\beta, \pi_1, \beta_1, \pi_2, \beta_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $A \wedge B$,
  π_1 *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_1{:}A$, *and*
  π_2 *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_2{:}B$.
- *If* π_1, π_2 *are proof trees, then* orleft($\alpha, \pi_1, \alpha_1, \pi_2, \alpha_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\alpha]$ *has form* $A \vee B$,
  π_1 *is a proof of* $(\Gamma - \alpha) + \alpha_1{:}A \vdash \Delta$, *and*
  π_2 *is a proof of* $(\Gamma - \alpha) + \alpha_2{:}B \vdash \Delta$.
- *If* π *is a proof tree and* $\beta_1 \neq \beta_2$, *then* orright($\beta, \pi, \beta_1, \beta_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $A \vee B$, *and*
  π *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_1{:}A + \beta_2{:}B$.
- *If* π *is a proof tree, then* impliesleft($\alpha, \pi_1, \beta_1, \pi_2, \alpha_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $A \rightarrow B$, *and*
  π_1 *is a proof of* $(\Gamma - \alpha) \vdash \Delta + \beta_1{:}A$, *and*
  π_2 *is a proof of* $(\Gamma - \alpha) + \alpha_2{:}B \vdash \Delta$.
- *If* π *is a proof tree and* $\alpha_1 \neq \beta_2$, *then* impliesright($\beta, \pi_1, \alpha_1, \pi_2, \beta_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $A \rightarrow B$,
  π_1 *is a proof of* $\Gamma + \alpha_1{:}A \vdash (\Delta - \beta) + \beta_2{:}B$.
- *If* π *is a proof tree and* $\alpha_1 \neq \alpha_2$, *then* equivleft($\alpha, \pi, \alpha_1, \alpha_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $A \leftrightarrow B$, *and*
  π *is a proof of* $(\Gamma - \alpha) + \alpha_1{:}(A \rightarrow B) + \alpha_2{:}(B \rightarrow A) \vdash \Delta$.
- *If* π_1, π_2 *are proof trees, then* equivright($\beta, \pi_1, \beta_1, \pi_2, \beta_2$) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $A \leftrightarrow B$,
  π_1 *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_1{:}(A \rightarrow B)$, *and*
  π_2 *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_2{:}(B \rightarrow A)$.
- *If* π *is a proof tree and* t *is a term, then* forallleft(α, π, α_1, t) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $\forall x\ P$ *and* π *is a proof of* $(\Gamma - \alpha) + \alpha_1{:}P[x := t] \vdash \Delta$.
- *If* π *is a proof tree and* y *is a variable, then* forallright(β, π, β_1, y) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Delta[\beta]$ *has form* $\forall x\ P$,
  y *is not free in* Γ, Δ *or* P, *and*
  π *is a proof of* $\Gamma \vdash (\Delta - \beta) + \beta_1{:}P[x := y]$.
- *If* π *is a proof tree and* y *is a variable, then* existsleft(α, π, α_1, y) *is also a proof tree. It is a proof of* $\Gamma \vdash \Delta$ *if* $\Gamma[\alpha]$ *has form* $\exists x\ P$,
  y *is not free in* Γ, Δ *or* P, *and*
  π *is a proof of* $(\Gamma + \alpha) + \alpha_1{:}P[x := y] \vdash \Delta$

– *If π is a proof tree and t is a term, then* existsright(β, π, β_1, t) *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if $\Delta[\beta]$ has form $\exists x\ P$ and*
  π is a proof of $\Gamma \vdash (\Delta - \beta) + \beta_1{:}P[x := t]$.
– *If t is a term, then* eqrefl(β, t) *is a proof tree. It is a proof of $\Gamma \vdash \Delta$ if*
  $\Delta[\beta] = (t \approx t)$.
– *If π is a proof tree and ρ is a position, then* replleft$(\alpha_1, \alpha_2, \pi, \rho, \alpha_3)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if $\Gamma[\alpha_1]$ has form $t_1 \approx t_2$, if $\Gamma[\alpha_2]$ has form $A_\rho[t_2]$, and π is a proof of $(\Gamma - \alpha_2) + \alpha_3{:}A_\rho[t_1] \vdash \Delta$.*
– *If π is a proof tree and ρ is a position, then* replright$(\alpha, \beta_1, \pi, \rho, \beta_2)$ *is also a proof tree. It is a proof of $\Gamma \vdash \Delta$ if $\Gamma[\alpha]$ has form $t_1 \approx t_2$, if $\Delta[\beta_1]$ has form $B_\rho[t_1]$, and π is a proof of $\Gamma \vdash (\Delta - \beta_1) + \beta_2{:}B_\rho[t_2]$.*

As an example, consider the following proof:

$$\dfrac{\dfrac{\alpha_1{:}A,\ \alpha_2{:}B \vdash \beta_1{:}B \qquad\qquad \alpha_1{:}A,\ \alpha_2{:}B \vdash \beta_2{:}A}{\alpha_1{:}A,\ \alpha_2{:}B \vdash \beta{:}B \wedge A}}{\alpha{:}A \wedge B \vdash \beta{:}B \wedge A}$$

It can be represented by the following proof term:

$$\mathrm{andleft}(\alpha, \mathrm{andright}(\beta, \mathrm{ax}(\alpha_2, \beta_1), \beta_1, \mathrm{ax}(\alpha_1, \beta_2), \beta_2), \alpha_1, \alpha_2).$$

Following [5], we consider a rule as a binder that binds the labels that it introduces in the subproofs where the label is introduced. For example, andleft$(\alpha, \pi, \alpha_1, \alpha_2)$ introduces the labels α_1, α_2 in π. Therefore, it can be viewed as binding any occurrences of α_1, α_2 in π. Likewise, we consider forallright(β, π, β_1, y), as a binder that binds any occurrences of y in π.

Viewing the rules as binders makes it possible to define a notion of α-equivalence of proofs. This has the advantage that label conflicts can be resolved by renaming labels. Without α-equivalence, a rule introducing some formula with label α cannot be applied on a labelled sequent already containing α. However, if we use α-equivalence, we can rename α into a new label α' and continue proof checking. As an example, the proof tree given a few lines above would not be a proof of $\alpha{:}A \wedge B,\ \alpha_1{:}A \vdash \beta{:}B \wedge A$. Using α-equivalence, we can replace α_1 in the proof tree by some α_1' and the sequent will be accepted. The main advantage of this is that proof checking becomes monotone:

Lemma 2. *If π is a proof tree, which is a proof of some labelled sequent $\Gamma \vdash \Delta$, and $\Gamma \subseteq \Gamma'$, $\Delta \subseteq \Delta'$, then π is also a proof of $\Gamma' \vdash \Delta'$.*

The following property is important for proof reductions. It is assumed that substitution is capture avoiding:

Lemma 3. *Let π a proof of labelled sequent $\Gamma \vdash \Delta$ containing a free variable x. Let t be some term. Then $\pi[x := t]$ is a proof of $(\Gamma \vdash \Delta)[x := t]$.*

The following property is important, because it makes it possible to use proof terms as schemata, i.e. as objects that can be instantiated.

Theorem 1. *Let π be a proof of a labelled sequent $\Gamma \vdash \Delta$. Let $A(x_1, \ldots, x_n)$ be an n-ary atom occurring in $\Gamma \vdash \Delta$, s.t. $x_1, \ldots, x_n$ are its free variables. Let $F(x_1, \ldots, x_n, y_1, \ldots, y_m)$ be a formula having at least free variables $x_1, \ldots, x_n$, and with possible other free variables $y_1, \ldots, y_m$. Assume that no occurrence of $A(x_1, \ldots, x_n)$ in $\Gamma \vdash \Delta$ is in the scope of a quantifier that binds one of the y_j and that no occurrence of $A(x_1, \ldots, x_n)$ in a cut formula occurring in π is in the scope of a quantifier that binds one of the y_j. Let π' be obtained from π by substituting $A(x_1, \ldots, x_n) := F(x_1, \ldots, x_n, y_1, \ldots, y_m)$ in every cut formula in π. Then π' is a proof of $(\Gamma \vdash \Delta)[A(x_1, \ldots, x_n) := F(x_1, \ldots, x_n, y_1, \ldots, y_m)]$.*

Note that when π is cut free, then $\pi' = \pi$. The reason that Theorem 1 holds, is the fact that the cut rule is the only rule that explicitly mentions formulas.

In case variables from $y_1, \ldots, y_m$ are caught, it is always possible to obtain an α-variant of $\Gamma \vdash \Delta$ and π, s.t. no capture takes place. The same holds for any cut formula in π.

As an example, consider the sequent $\forall x(A(x) \wedge B) \vdash (\forall x A(x)) \wedge B$, which clearly has a cut free proof. Lemma 1 allows to substitute $P(y, z)$ for B, (because y and z are not caught), but it does not allow to substitute $P(x, y, z)$ for B. The sequent can be renamed into $\forall x_1(A(x_1) \wedge B) \vdash (\forall x_1 A(x_1)) \wedge B$.

4 The Negation Normal Form Transformation

We describe in detail how we intend to implement the negation normal form transformation with proof generation.

Definition 4. *Formula F is in* negation normal form *(NNF) if* **(1)** *F does not contain $\rightarrow$ or $\leftrightarrow$,* **(2)** *negation is applied only on atoms in F,* **(3)** *if F contains $\top$ (or $\bot$,) then $F = \top$, (or $\bot$).*

A formula can be easily transformed into NNF by two rewrite systems. The first rewrite system removes $\rightarrow$ and $\leftrightarrow$, and it pushes the negations inwards. The second rewrite system moves $\bot$ and $\top$ upwards until they either disappear, or reach the top of the formula. The rewrite systems could be combined into one rewrite system, but that would be inefficient, because the two rewrite systems are more efficient with different rewrite systems. The first rewrite system is given by the following table:

$$\begin{array}{ll}
A \rightarrow B & \Rightarrow \neg A \vee B \\
A \leftrightarrow B & \Rightarrow (\neg A \vee B) \wedge (A \vee \neg B) \\
\\
\neg\neg A & \Rightarrow A \\
\neg(A \vee B) & \Rightarrow \neg A \wedge \neg B \\
\neg(A \wedge B) & \Rightarrow \neg A \vee \neg B \\
\neg(\forall x\ P(x)) & \Rightarrow \exists x\ \neg P(x) \\
\neg(\exists x\ P(x)) & \Rightarrow \forall x\ \neg P(x)
\end{array}$$

The following algorithm normalizes a formula under the set of rules.

Algorithm 1
formula nnf12(formula$F$)
begin
  while there are a rule $A \Rightarrow B$ *and a substitution* Θ, *s.t.*
  $A\Theta = F$ *do*
    $F := B\Theta$
  if F *is an atom,* $A = \bot$ *or* $A = \top$, *then return* F
  if F *has form* $\neg A$, *with* A *an atom,* $A = \bot$, *or* $A = \top$, *then return* $\neg A$.
  if F *has form* $A \wedge B$, *then return* $\mathrm{nnf12}(A) \wedge \mathrm{nnf12}(B)$
  if F *has form* $A \vee B$, *then return* $\mathrm{nnf12}(A) \vee \mathrm{nnf12}(B)$
  if F *has form* $\forall x\ P(x)$, *then return* $\forall x\ \mathrm{nnf12}(P(x))$
  if F *has form* $\exists x\ P(x)$, *then return* $\exists x\ \mathrm{nnf12}(P(x))$
end

The algorithm implements a particular rewrite strategy, namely *outside-inside* normalization. It assumes that the rewrite front starts at the outside and then moves inward. When the formula has been normalized at one point, then this point does not need to be reconsidered anymore. There second part of the rewrite system needs exactly the opposite strategy, *inside-outside* normalization. If one would combine the systems, one would have to look for possible rewrites everywhere in the formula, which is less efficient.

Definition 5. *A* justified sequent *is a pair of form* $(\alpha{:}A \vdash \beta{:}B,\ \pi)$, *s.t.* $\alpha \neq \beta$ *and* π *is a proof of* $\alpha{:}A \vdash \beta{:}B$. *A* justified rewrite rule *is a justified sequent* $(\alpha{:}A \vdash \beta{:}B,\ \pi)$, *s.t.* $A \Rightarrow B$ *is a rewrite rule.*

There is no formal distinction between a justified sequent and a justified rewrite rule, but we give them different names because their roles are different.

We now modify the rewrite algorithm, so that it will output a proof at the same time with its result. It will do this by returning a justified sequent.

Algorithm 2 *Function* $\mathrm{nnf12}(F, \alpha)$ *returns a justified sequent* $(\alpha{:}F \vdash \beta{:}F',\ \pi)$, *s.t.* $F' = \mathrm{nnf12}(F)$, *and* β *is some new label.*

justifiedsequent nnf12(formula F, label α)
begin
    array of justifiedsequent Π;
    Initialize Π to the empty (zero length) array.
    while there are a justified rewrite rule $(\alpha'{:}A' \vdash \beta'{:}B',\ \pi')$ *and*
            a substitution Θ, s.t. $A'\Theta = F$ *do*
    begin
        Let γ be a new label, not occurring in Π, and distinct from α.
        Assign $\pi' := \pi'[\alpha' := \alpha,\ \beta' := \gamma]$. *(so that π' now proves* $\alpha{:}A' \vdash \gamma{:}B$*)*
        Append $(\alpha{:}A'\Theta \vdash \gamma{:}B'\Theta,\ \pi')$ *to Π. (the length of Π is increased by 1,*
            there is no need to modify π' because of Theorem 1)
        Assign $F := B\Theta$.
        Assign $\alpha := \gamma$.
    end
    If F is an atom, $F = \bot$ or $F = \top$, then
        return applycut(Π).
    If F has form $\neg A$, where A is an atom, $A = \bot$ or $A = \top$, then
        return applycut(Π).
    If F has form $A_1 \wedge A_2$, then
    begin
        Let $\alpha_1, \alpha_2, \beta$ *be new, distinct labels.*
        Assign B_1, β_1, π_1 *from* $(\alpha_1{:}A_1 \vdash \beta_1{:}B_1,\ \pi_1) :=$ nnf12(A_1, α_1)
        Assign B_2, β_2, π_2 *from* $(\alpha_2{:}A_2 \vdash \beta_2{:}B_2,\ \pi_2) :=$ nnf12(A_2, α_2)
        Append $\alpha{:}A_1 \wedge A_2 \vdash \beta{:}B_1 \wedge B_2$,
            andleft(α,
                andright(β,
                    weakenleft(α_2, π_1), β_1,
                    weakenleft(α_1, π_2), β_2),
                α_1, α_2)) *to Π.*
        return applycut(Π)
    end
    If F has form $A_1 \vee A_2$, then
    begin
        Let $\alpha_1, \alpha_2, \beta$ *be new, distinct labels.*
        Assign B_1, β_1, π_1 *from* $(\alpha_1{:}A_1 \vdash \beta_1{:}B_1,\ \pi_1) :=$ nnf12(A_1, α_1)
        Assign B_2, β_2, π_2 *from* $(\alpha_2{:}A_2 \vdash \beta_2{:}B_2,\ \pi_2) :=$ nnf12(A_2, α_2)
        Append $(\alpha{:}A_1 \vee A_2 \vdash \beta{:}B_1 \vee B_2$,
            orright(β,
                orleft(α,
                    weakenright(β_2, π_1), α_1,
                    weakenright(β_1, π_2), α_2),
                β_1, β_2)) *to Π.*
        return applycut(Π).
    end

If F *has form* $\forall x\ P(x)$, *then*
begin
  Let α_1 *and* β *be a new, distinct labels.*
  Assign $Q(x), \beta_1, \pi_1$ *from* $(\alpha_1{:}\,P(x) \vdash \beta_1{:}\,Q(x),\ \pi_1) := \mathrm{nnf12}(P(x), \alpha_1)$
  Append $(\alpha{:}\forall x\ P(x) \vdash \beta{:}\forall x\ Q(x),$
    $\mathrm{forallright}(\beta, \mathrm{forallleft}(\alpha, \pi_1, \alpha_1, x), \beta_1, x)\ )$ *to* Π.
  return $\mathrm{applycut}(\Pi)$
end

If F *has form* $\exists x\ P(x)$, *then*
begin
  Let α_1 *and* β *be new, distinct labels.*
  Assign $Q(x), \beta_1, \pi_1$ *from* $(\alpha_1{:}\,P(x) \vdash \beta_1{:}\,Q(x),\ \pi_1) := \mathrm{nnf12}(P(x), \alpha_1)$
  Append $(\alpha{:}\,\exists x\ P(x) \vdash \beta{:}\,\exists x\ Q(x),$
    $\mathrm{existsleft}(\alpha, \mathrm{existsright}(\beta, \pi_1, \beta_1, x), \alpha_1, x)\ )$ *to* Π.
  return $\mathrm{applycut}(\Pi)$
end
end

Function $\mathrm{applycut}(\Pi)$ *combines the proofs* π_i *of* $\alpha_i{:}\,A_i \vdash \beta_i{:}\,B_i$ *into one proof by using the cut rule. It must be the case that* $\beta_{i+1} = \alpha_i$, *and* $B_{i+1} = A_i$, *for* $1 \leq i < |\Pi|$.

(justifiedsequent) applycut(array of justifiedsequent Π)
begin
  Σ *is a variable of type labelled sequent.*
  π *is a variable of type proof tree.*
  Assign $(\Sigma, \pi) := \Pi_1$

  for $i := 2$ *to* $|\Pi|$ *do*
  begin
    Assign $(\alpha{:}\,A \vdash \beta{:}\,B) := \Sigma$
    Assign $(\beta{:}\,B \vdash \gamma{:}\,C,\ \rho) := \Pi_i$
    Assign $\Sigma := \alpha{:}\,A \vdash \gamma{:}\,C$
    Assign $\pi := \mathrm{cut}(B, \mathrm{weakenleft}(\alpha, \rho), \beta, \mathrm{weakenright}(\gamma, \pi), \beta)$
  end
  return $(\Sigma,\ \pi)$
end

We now come to the second part of the rewrite system that will ensure the third condition of Definition 4.

$$
\begin{array}{ll}
A \lor \bot \Rightarrow A & A \land \bot \Rightarrow \bot \\
A \lor \top \Rightarrow \top & A \land \top \Rightarrow A \\
\bot \lor A \Rightarrow A & \bot \land A \Rightarrow \bot \\
\top \lor A \Rightarrow \top & \top \land A \Rightarrow A \\
 & \\
\forall x \ \bot \ \Rightarrow \bot & \exists x \ \bot \ \Rightarrow \bot \\
\forall x \ \top \ \Rightarrow \top & \exists x \ \top \ \Rightarrow \top
\end{array}
$$

In order to obtain a normal form, Algorithm 1 cannot be used, because the outside-inside strategy does generally not result in a normal form. Instead, an inside-outside rewrite strategy has to be used:

Algorithm 3
formula nnf3 (formula F)
begin
  if F *is an atom,* $A = \bot$ *or* $A = \top$, *then* $G := F$
  if F *has form* $\neg A$, *with* A *an atom,* $A = \bot$, *or* $A = \top$, *then* $G := F$
  if F *has form* $A \land B$, *then* $G := \mathrm{nnf3}(A) \land \mathrm{nnf3}(B)$
  if F *has form* $A \lor B$, *then* $G := \mathrm{nnf3}(A) \lor \mathrm{nnf3}(B)$
  if F *has form* $\forall x\ P(x)$, *then* $G := \forall x\ \mathrm{nnf3}(P(x))$
  if F *has form* $\exists x\ P(x)$, *then* $G := \exists x\ \mathrm{nnf3}(P(x))$

  while there are a rule $A \Rightarrow B$ *and a substitution* Θ, *s.t.* $A\Theta = G$ *do*
    $G := B\Theta$
end

Algorithm 3 differs from Algorithm 1 only in the fact that rewriting on the current level is attempted only after the subterms have been normalized.

Algorithm 2 can be easily modified correspondingly, by moving the while-loop in the beginning towards the end. It can be also easily adopted to situations where more complicated rewrite strategies are needed.

5 Subformula Replacement

Some steps in the clausal normal form transformation can cause exponential blowup of the formula. The problematic steps are the replacement of $A \leftrightarrow B$ by $(\neg A \lor B) \land (A \lor \neg B)$, and the factoring of conjunctions over disjunctions performed by the following rules: $(A \land B) \lor C \Rightarrow (A \lor C) \land (B \lor C)$, $A \lor (B \land C) \Rightarrow (A \lor B) \land (A \lor C)$.
Expansion of $\leftrightarrow$ would cause exponential blowup on the following sequence of formulas

$$(a_1 \leftrightarrow (a_2 \leftrightarrow \cdots (a_{n-1} \leftrightarrow a_n))),\ n > 0.$$

Factoring would cause exponential blowup on the following sequence of formulas

$$(a_1 \land b_1) \lor \cdots \lor (a_n \land b_n),\ n > 0.$$

In order to avoid this, it is possible to use subformula replacement. For example, in the last formula, one can introduce new symbols $x_1, \ldots, x_n$, and replace it by the equisatisfiable set of formulas

$$x_1 \vee \cdots \vee x_n,\ x_1 \leftrightarrow (a_1 \wedge b_1), \ldots, x_n \leftrightarrow (a_n \wedge b_n).$$

Subformula replacement as such is not first-order, but it can be easily dealt with within first-order logic, by observing that the new names are abbreviations of certain formulas. During the CNF-transformation, we allow to add premisses of the following form to the set of premisses:

$$\forall x_1 \cdots x_n\ X(x_1, \ldots, x_n) \leftrightarrow F(x_1, \ldots, x_n).$$

X is a new symbol that does not yet occur in the premisses and also not in $F(x_1, \ldots, x_n)$. When the resolution prover succeeds, one obtains a proof π of a sequent $\Gamma, D_1, \ldots, D_k \vdash \bot$, in which Γ is the set of original first-order formulas, and $D_1, \ldots, D_k$ are the introduced premisses, which are all of form

$$\forall x_1 \cdots x_{n_j}\ X_j(x_1, \ldots, x_{n_j}) \leftrightarrow F_j(x_1, \ldots, x_{n_j}),\ \text{for } 1 \leq j \leq k.$$

A new symbol X_j can occur in $F_{j'}$ only when $j' > j$, and it cannot occur in Γ. By substituting the X_j away and applying Theorem 1, the proof π can be transformed into a proof π' of $\Gamma, E_1, \ldots, E_k \vdash \bot$ in which each E_j has form

$$\forall x_1 \cdots x_{n_j}\ F(x_1, \ldots, x_{n_j}) \leftrightarrow F(x_1, \ldots, x_{n_j}).$$

These are simple tautologies which can be proven and cut away.

6 Antiprenexing

The purpose of anti-prenexing (also called miniscoping) is to obtain smaller Skolem terms. In many formulas, not everything that is in the scope of a quantifier, does also depend on this quantifier. If one systematically factors such subformulas out of the scope of the quantifier, one can often reduce dependencies between quantifiers. For details, we refer to [4], here we give only a few examples:

Example 1. Without anti-prenexing, $\forall x\ \exists y[\ p(x) \wedge q(y)]$ skolemizes into $\forall x\ [p(x) \wedge q(f(x))]$. Antiprenexing reduces the formula to $(\ \forall x\ p(x)\ ) \wedge (\ \exists y\ q(y)\ )$, which Skolemizes into $(\ \forall x\ p(x)\ ) \wedge q(c)$.

Without anti-prenexing, $\forall x\ \exists y_1 y_2\ [p(y_1) \wedge q(x, y_2)]$ skolemizes into $\forall x\ [\ p(f_1(x)) \wedge q(x, f_2(x))\ ]$. Antiprenexing reduces the formula to $\forall x\ [\exists y_1\ p(y_1) \wedge \exists y_2\ q(x, y_2)]$, which Skolemizes into $\forall x\ [p(c_1) \wedge q(f_2(x))]$.

Without anti-prenexing, $\forall x\ \exists y\ [p(x) \wedge q(y) \wedge r(x)]$ skolemizes into $\forall x[p(x) \wedge q(f(x)) \wedge r(x)]$. Antiprenexing can reduce the formula to $\forall x\ [p(x) \wedge r(x) \wedge \exists y\ q(y)]$, which can be Skolemized into $\forall x\ [p(x) \wedge r(x) \wedge q(c)]$.

As far as we can see, all replacements can be handled by the following 'rewrite system':

$$
\begin{array}{llll}
A \vee B & \Rightarrow B \vee A & A \wedge B & \Rightarrow B \wedge A \\
A \vee (B \vee C) & \Rightarrow A \vee B \vee C & A \wedge (B \wedge C) & \Rightarrow A \wedge B \wedge C \\
\\
\forall x\ (P(x) \wedge Q) & \Rightarrow (\forall x\ P(x)) \wedge Q & \exists x\ (P(x) \wedge Q) & \Rightarrow (\exists x\ P(x)) \wedge Q \\
\forall x\ (P \wedge Q(x)) & \Rightarrow P \wedge \forall x\ Q(x) & \exists x\ (P \wedge Q(x)) & \Rightarrow P \wedge \exists x\ Q(x) \\
\forall x\ (P(x) \vee Q) & \Rightarrow (\forall x\ P(x)) \vee Q & \exists x\ (P(x) \vee Q) & \Rightarrow (\exists x\ P(x)) \vee Q \\
\forall x\ (P \vee Q(x)) & \Rightarrow P \vee \forall x\ Q(x) & \exists x\ (P \vee Q(x)) & \Rightarrow P \vee \exists x\ Q(x) \\
\\
\forall x\ P & \Rightarrow P & \exists x\ P & \Rightarrow P \\
\forall x \forall y\ P(x, y) & \Rightarrow \forall y \forall x\ P(x, y) & \exists x \exists y\ P(x, y) & \Rightarrow \exists y \exists x\ P(x, y)
\end{array}
$$

The system is not a rewrite system in the usual sense, because an additional strategy is needed for deciding when a certain rule should be applied. Straightforward normalization would not terminate due to the presence of permutation rules. If one would remove the permutation rules, one would often not obtain the best possible result. For example, in the last formula of the example, $(p(x) \wedge q(y)) \wedge r(x)$ first has to be permuted into $(p(x) \wedge r(x)) \wedge q(y)$, before the rule $\exists x(P \wedge Q(x)) \Rightarrow P \wedge \exists x\ Q(x)$ can be applied.

Despite the fact that the decision making is more complicated than was the case for the NNF, Algorithm 2 can be still modified for anti-prenexing, because the decision making plays no role in the proof generation. For the proof generation, only correctness of the rules matters, and all rules can be easily proven correct.

7 Proof Reductions

Proof reductions are important, because they make it possible to obtain modularity and flexibility. For a detailed motivation, we refer to [1]. There, a special calculus called *replacement calculus* was introduced which allows for certain reductions that remove repeated building up of the same context in a proof. In sequent calculus, the standard reductions of cut elimination correspond to the reductions of the replacement calculus, so there is no need anymore for the replacement calculus. For the purpose of proof simplification, one should implement all standard reductions of cut elimination (see [3]), except for the permutation of a cut with a contraction, because this permutation is the cause of increasement in proof length.

The proof reductions are needed in order to combine the repeated building up of contexts. Suppose that one has a big formula of form $F[A_1]$, that A_1 is first rewritten into A_2, and after that into A_3. Algorithm 2 lifts a proof of $A_1 \vdash A_2$ to a proof of $F[A_1] \vdash F[A_2]$. After that, it lifts a proof of $A_2 \vdash A_3$ to a proof of $F[A_2] \vdash F[A_3]$, which is then combined, using cut, into a proof of $F[A_1] \vdash F[A_3]$.

However, it would be more efficient to first apply cut on $A_1 \vdash A_2$ and $A_2 \vdash A_3$, resulting in $A_1 \vdash A_3$, and lift this proof to $F[A_1] \vdash F[A_3]$.

Combination of context lifting can be done only if one knows in advance the order in which the replacements will be made, and when they are near to each other. This was the case for the NNF-transformation, and Algorithm 2 makes use of this fact, both for the outside-inside strategy, and for the inside-outside strategy.

If one does not know the order of replacements in advance, then Algorithm 2 will not avoid repeated lifting into the same context. This would be the case for anti-prenexing. In that case, one has to rely on proof reductions. Using the standard reductions of cut elimination, the cut on the top level can be permuted with the rules that build up the context, until it either disappears, or reaches a contraction.

Using proof terms, the reductions can be easily implemented by a rewrite system on proof terms. We give a few examples of the reductions involved, and give the corresponding rewrite rules:

$$\frac{\dfrac{\Gamma \vdash \Delta,\ \beta_1{:}A \qquad\qquad \Gamma \vdash \Delta,\ \beta_2{:}B}{\Gamma \vdash \Delta,\ \beta{:}A \wedge B} \qquad\qquad \dfrac{\Gamma,\ \alpha_1{:}A,\ \alpha_2{:}B \vdash \Delta}{\Gamma,\ \alpha{:}A \wedge B \vdash \Delta}}{\Gamma \vdash \Delta}$$

is replaced by

$$\frac{\Gamma \vdash \Delta,\ \beta_2{:}B \qquad\qquad \dfrac{\Gamma \vdash \Delta,\ \beta_1{:}A \qquad\qquad \Gamma,\ \alpha_1{:}A,\ \alpha_2{:}B \vdash \Delta}{\Gamma,\ \alpha_2{:}B \vdash \Delta}}{\Gamma \vdash \Delta.}$$

The corresponding rewrite rule is

$$\mathrm{cut}(A \wedge B, \mathrm{andleft}(\alpha, \pi, \alpha_1, \alpha_2), \alpha, \mathrm{andright}(\beta, \pi_1, \beta_1, \pi_2, \beta_2), \beta) \Rightarrow$$
$$\mathrm{cut}(B, \mathrm{cut}(A, \pi, \alpha_1, \pi_1, \beta_1), \alpha_2, \pi_2, \beta_2).$$

The following proof fragment

$$\frac{\dfrac{\Gamma \vdash \Delta,\ \beta_1{:}P[x := y]}{\Gamma \vdash \Delta,\ \beta{:}\forall x\ P(x)} \qquad\qquad \dfrac{\Gamma,\ \alpha_1{:}P[x := t] \vdash \Delta}{\Gamma,\ \alpha{:}\forall x\ P(x) \vdash \Delta}}{\Gamma \vdash \Delta}$$

reduces into

$$\frac{\Gamma \vdash \Delta,\ \beta_1\!: P[x := t] \qquad\qquad \Gamma,\ \alpha_1\!: P[x := t] \vdash \Delta}{\Gamma \vdash \Delta}$$

The corresponding rewrite rule is

$$\mathrm{cut}(\forall x\ P(x), \mathrm{forallleft}(\alpha, \pi_2, \alpha_1, t), \alpha, \mathrm{forallright}(\beta, \pi_1, \beta_1, y), \beta) \Rightarrow$$
$$\mathrm{cut}(P[x := t], \pi_2, \alpha_1, \pi_1[y := t], \beta_1).$$

8 Conclusions

We have shown that implementing the CNF-transformation with proof generation is possible. We have given a data structure (inspired by [5]) for the represention of sequent calculus proofs, which is concise, and which allows for implementation of proof reductions. We have given a general translation algorithm, based on rewriting, that covers nearly all of the transformations involved.

Proof generation will not be feasible for formulas that are propositionally complex. Such formulas will have exponentially large proofs, (because probably $\mathcal{NP} \neq$ co-$\mathcal{NP}$.)

References

1. Hans de Nivelle. Extraction of proofs from the clausal normal form transformation. In Julian Bradfield, editor, *Proceedings of the 16th International Workshop on Computer Science Logic (CSL 2002)*, volume 2471 of *Lecture Notes in Artificial Intelligence*, pages 584–598, Edinburgh, Scotland, UK, September 2002. Springer.
2. Hans de Nivelle. Translation of resolution proofs into short first-order proofs without choice axioms. In Franz Baader, editor, *Proceedings of the 19th International Conference on Computer Aided Deduction (CADE 19)*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 365–379, Miami, USA, July 2003. Springer Verlag.
3. Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.
4. Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.
5. Frank Pfenning. Structural cut elimination. In Dexter Kozen, editor, *Proceedings 10th Annual IEEE Symposion on Logic in Computer Science*, pages 156–166. IEEE Computer Society Press, 1995.