Invertibility in Sequent Calculi

Peter Chapman

School of Computer Science, University of St Andrews
Email: pc@cs.st-andrews.ac.uk

Abstract. The invertibility of the rules of a sequent calculus is important for guiding proof search and can be used in some formalised proofs of Cut admissibility. We present sufficient conditions for when a rule is invertible with respect to a calculus. We illustrate the conditions with examples. It must be noted we give purely syntactic criteria; no guarantees are given as to the suitability of the rules.

1 Introduction

In this paper, we give an overview of some results about invertibility in sequent calculi. The framework is outlined in §2. The results are mainly concerned with multisuccedent calculi that have a single principal formula. We will use, as our running example throughout, the calculus **G3cp**. In §4, we look at the formalisation of single-succedent calculi; in §5, the formalisation in *Nominal Isabelle* for first-order calculi is shown; in §6 the results for modal logic are examined. We return to multisuccedent calculi in §7 to look at manipulating rule sets.

2 Formalising the Framework

2.1 Formulae and Sequents

A *formula* is either a propositional variable, the constant $\bot$, or a connective applied to a list of formulae. We thus have a type variable indexing formulae, where the type variable will be a set of connectives. In the usual way, we index propositional variables by use of natural numbers. So, formulae are given by the datatype:

datatype *'a form* = *At nat*
| *Compound 'a 'a form list*
| *ff*

For **G3cp**, we define the datatype *Gp*, and give the following abbreviations:

datatype *Gp* = *con* | *dis* | *imp*
type-synonym *Gp-form* = *Gp form*

abbreviation *con-form* (**infixl** ‹∧*› *80*) **where**
  $p \wedge* q \equiv Compound\ con\ [p,q]$

abbreviation *dis-form* (**infixl** ‹∨*› *80*) **where**
  $p \vee* q \equiv Compound\ dis\ [p,q]$

abbreviation *imp-form* (**infixl** ‹⊃› *80*) **where**
  $p \supset q \equiv Compound\ imp\ [p,q]$

A *sequent* is a pair of multisets of formulae. Sequents are indexed by the connectives used to index the formulae. To add a single formula to a multiset of formulae, we use the symbol $\oplus$, whereas to join two multisets, we use the symbol $+$.

2.2 Rules and Rule Sets

A *rule* is a list of sequents (called the premisses) paired with a sequent (called the conclusion). The two *rule sets* used for multisuccedent calculi are the axioms, and the uniprincipal rules (i.e. rules having one principal formula). Both are defined as inductive sets. There are two clauses for axioms, corresponding to $L\bot$ and normal axioms:

inductive-set *Ax* **where**
  id: $([], \wr\ At\ i\ \wr \Rightarrow* \wr\ At\ i\ \wr) \in Ax$
| *Lbot*: $([], \wr\ ff\ \wr \Rightarrow* \emptyset) \in Ax$

The set of uniprincipal rules, on the other hand, must not have empty premisses, and must have a single, compound formula in its conclusion. The function `mset` takes a sequent, and returns the multiset obtained by adding the antecedent and the succedent together:

inductive-set *upRules* **where**
  I: $[\![\ mset\ c \equiv \wr\ Compound\ R\ Fs\ \wr\ ;\ ps \neq []\ ]\!] \Longrightarrow (ps,c) \in upRules$

For **G3cp**, we have the following six rules, which we then show are a subset of the set of uniprincipal rules:

inductive-set *g3cp*
where
  conL: $([\wr\ A\ \wr + \wr\ B\ \wr \Rightarrow* \emptyset], \wr\ A \wedge* B\ \wr \Rightarrow* \emptyset) \in g3cp$
| *conR*: $([\emptyset \Rightarrow* \wr\ A\ \wr, \emptyset \Rightarrow* \wr\ B\ \wr], \emptyset \Rightarrow* \wr\ A \wedge* B\ \wr) \in g3cp$
| *disL*: $([\wr\ A\ \wr \Rightarrow* \emptyset, \wr\ B\ \wr \Rightarrow* \emptyset], \wr\ A \vee* B\wr \Rightarrow* \emptyset) \in g3cp$
| *disR*: $([\emptyset \Rightarrow* \wr\ A\ \wr + \wr\ B\ \wr], \emptyset \Rightarrow* \wr\ A \vee* B\ \wr) \in g3cp$
| *impL*: $([\emptyset \Rightarrow* \wr\ A\ \wr, \wr\ B\ \wr \Rightarrow* \emptyset], \wr\ A \supset B\ \wr \Rightarrow* \emptyset) \in g3cp$
| *impR*: $([\wr\ A\ \wr \Rightarrow* \wr\ B\ \wr], \emptyset \Rightarrow* \wr\ A \supset B\ \wr) \in g3cp$

lemma *g3cp-upRules*:
shows $g3cp \subseteq upRules$
⟨*proof*⟩

We have thus given the *active* parts of the **G3cp** calculus. We now need to extend these active parts with *passive* parts.

Given a sequent C, we extend it with another sequent S by adding the two antecedents and the two succedents. To extend an active part (Ps, C) with a sequent S, we extend every $P \in Ps$ and C with S:

overloading
  extend ≡ *extend*
  extendRule ≡ *extendRule*
begin

definition *extend*
  where *extend forms seq* ≡ (*antec forms* + *antec seq*) ⇒∗ (*succ forms* + *succ seq*)

definition *extendRule*
  where *extendRule forms R* ≡ (*map* (*extend forms*) (*fst R*), *extend forms* (*snd R*))

end

Given a rule set $\mathcal{R}$, the *extension* of $\mathcal{R}$, called $\mathcal{R}^\star$, is then defined as another inductive set:

inductive-set *extRules* :: *'a rule set* ⇒ *'a rule set* (‹-∗›)
  for *R* :: *'a rule set*
  where *I*: *r* ∈ *R* ⟹ *extendRule seq r* ∈ *R*∗

The rules of **G3cp** all have unique conclusions. This is easily formalised:

overloading *uniqueConclusion* ≡ *uniqueConclusion*
begin

definition *uniqueConclusion* :: *'a rule set* ⇒ *bool*
  where *uniqueConclusion R* ≡ ∀ *r1* ∈ *R*. ∀ *r2* ∈ *R*. (*snd r1* = *snd r2*) ⟶ (*r1* = *r2*)

end

lemma *g3cp-uc*:
shows *uniqueConclusion g3cp*
⟨*proof*⟩

2.3 Principal Rules and Derivations

A formula A is *left principal* for an active part R iff the conclusion of R is of the form $A \Rightarrow \emptyset$. The definition of *right principal* is then obvious. We have an inductive predicate to check these things:

inductive *rightPrincipal* :: *'a rule* ⇒ *'a form* ⇒ *bool*
  where
  up: *C* = (∅ ⇒∗ {|*Compound F Fs*|}) ⟹

rightPrincipal (*Ps*,*C*) (*Compound F Fs*)

As an example, we show that if $A \wedge B$ is principal for an active part in **G3cp**, then $\emptyset \Rightarrow A$ is a premiss of that active part:

lemma *principal-means-premiss*:
assumes *a*: *rightPrincipal r* (*A* ∧∗ *B*)
and *b*: *r* ∈ *g3cp*
shows (∅ ⇒∗ ⁅ *A* ⁆) ∈ *set* (*fst r*)
⟨*proof*⟩

A sequent is *derivable* at height 0 if it is the conclusion of a rule with no premisses. If a rule has m premisses, and the maximum height of the derivation of any of the premisses is n, then the conclusion will be derivable at height $n + 1$. We encode this as pairs of sequents and natural numbers. A sequent S is derivable at a height n in a rule system $\mathcal{R}$ iff (S, n) belongs to the inductive set `derivable` $\mathcal{R}$:

inductive-set *derivable* :: *'a rule set* ⇒ *'a deriv set*
for *R* :: *'a rule set*
where
base: ⟦([],*C*) ∈ *R*⟧ ⟹ (*C*,*0*) ∈ *derivable R*
| *step*: ⟦ *r* ∈ *R* ; (*fst r*)≠[] ; ∀ *p* ∈ *set* (*fst r*). ∃ *n* ≤ *m*. (*p*,*n*) ∈ *derivable R* ⟧
⟹ (*snd r*,*m* + *1*) ∈ *derivable R*

In some instances, we do not care about the height of a derivation, rather that the root is derivable. For this, we have the additional definition of `derivable'`, which is a set of sequents:

inductive-set *derivable'* :: *'a rule set* ⇒ *'a sequent set*
for *R* :: *'a rule set*
where
base: ⟦ ([],*C*) ∈ *R* ⟧ ⟹ *C* ∈ *derivable' R*
| *step*: ⟦ *r* ∈ *R* ; (*fst r*) ≠ [] ; ∀ *p* ∈ *set* (*fst r*). *p* ∈ *derivable' R* ⟧
⟹ (*snd r*) ∈ *derivable' R*

It is desirable to switch between the two notions. Shifting from derivable at a height to derivable is simple: we delete the information about height. The converse is more complicated and involves an induction on the length of the premiss list:

lemma *deriv-to-deriv*:
assumes (*C*,*n*) ∈ *derivable R*
shows *C* ∈ *derivable' R*
⟨*proof*⟩

lemma *deriv-to-deriv2*:
assumes *C* ∈ *derivable' R*
shows ∃ *n*. (*C*,*n*) ∈ *derivable R*
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*pro*

3 Formalising the Results

A variety of "helper" lemmata are used in the proofs, but they are not shown. The proof tactics themselves are hidden in the following proof, except where they are interesting. Indeed, only the interesting parts of the proof are shown at all. The main result of this section is that a rule is invertible if the premisses appear as premisses of *every* rule with the same principal formula. The proof is interspersed with comments.

lemma *rightInvertible*:
fixes $\Gamma\ \Delta :: 'a\ form\ multiset$
assumes *rules*: $R' \subseteq upRules \wedge R = Ax \cup R'$
  and *a*: $(\Gamma \Rightarrow* \Delta \oplus Compound\ F\ Fs,n) \in derivable\ R*$
  and *b*: $\forall\ r' \in R.\ rightPrincipal\ r'\ (Compound\ F\ Fs) \longrightarrow (\Gamma' \Rightarrow* \Delta') \in set\ (fst\ r')$
shows $\exists\ m{\leq}n.\ (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta', m) \in derivable\ R*$
⟨*proof*⟩

As an example, we show the left premiss of $R\wedge$ in **G3cp** is derivable at a height not greater than that of the conclusion. The two results used in the proof (`principal-means-premiss` and `rightInvertible`) are those we have previously shown:

lemma *conRInvert*:
assumes $(\Gamma \Rightarrow* \Delta \oplus (A \wedge* B),n) \in derivable\ (g3cp \cup Ax)*$
shows $\exists\ m{\leq}n.\ (\Gamma \Rightarrow* \Delta \oplus A,m) \in derivable\ (g3cp \cup Ax)*$
⟨*proof*⟩

We can obviously show the equivalent proof for left rules, too:

lemma *leftInvertible*:
fixes $\Gamma\ \Delta :: 'a\ form\ multiset$
assumes *rules*: $R' \subseteq upRules \wedge R = Ax \cup R'$
  and *a*: $(\Gamma \oplus Compound\ F\ Fs \Rightarrow* \Delta,n) \in derivable\ R*$
  and *b*: $\forall\ r' \in R.\ leftPrincipal\ r'\ (Compound\ F\ Fs) \longrightarrow (\Gamma' \Rightarrow* \Delta') \in set\ (fst\ r')$
shows $\exists\ m{\leq}n.\ (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta', m) \in derivable\ R*$⟨*proof*⟩⟨*proof*⟩

A rule is invertible iff every premiss is derivable at a height lower than that of the conclusion. A set of rules is invertible iff every rule is invertible. These definitions are easily formalised:

overloading
  invertible ≡ *invertible*
  invertible-set ≡ *invertible-set*
begin

definition *invertible*
  where *invertible r R* ≡
    $\forall\ n\ S.\ (r \in R \wedge (snd\ (extendRule\ S\ r),n) \in derivable\ R*) \longrightarrow$
    $(\forall\ p \in set\ (fst\ (extendRule\ S\ r)).\ \exists\ m \leq n.\ (p,m) \in derivable\ R*)$

definition *invertible-set*

where *invertible-set* $R \equiv \forall\ (ps,c) \in R.$ *invertible* $(ps,c)\ R$

end

A set of multisuccedent uniprincipal rules is invertible if each rule has a different conclusion. **G3cp** has the unique conclusion property (as shown in §2.2). Thus, **G3cp** is an invertible set of rules:

lemma *unique-to-invertible*:
assumes $R' \subseteq upRules \wedge R = Ax \cup R'$
and *uniqueConclusion* R'
shows *invertible-set* $R\langle proof \rangle$

lemma *g3cp-invertible*:
shows *invertible-set* $(Ax \cup g3cp)$
$\langle proof \rangle$

3.1 Conclusions

For uniprincipal multisuccedent calculi, the theoretical results have been formalised. Moreover, the running example demonstrates that it is straightforward to implement such calculi and reason about them. Indeed, it will be this class of calculi for which we will prove more results in §7.

4 Single Succedent Calculi

We must be careful when restricting sequents to single succedents. If we have sequents as a pair of multisets, where the second is restricted to having size at most 1, then how does one extend the active part of $L \supset$ from **G3ip**? The left premiss will be $A \supset B \Rightarrow A$, and the extension will be $\Gamma \Rightarrow C$. The `extend` function must be able to correctly choose to discard the C.

Rather than taking this route, we instead restrict to single formulae in the succedents of sequents. This raises its own problems, since now how does one represent the empty succedent? We introduce a dummy formula `Em`, which will stand for the empty formula:

datatype *'a form* = *At nat*
| *Compound 'a 'a form list*
| *ff*
| *Em*

When we come to extend a sequent, say $\Gamma \Rightarrow C$, with another sequent, say $\Gamma' \Rightarrow C'$, we only "overwrite" the succedent if C is the empty formula:

overloading
extend $\equiv$ *extend*
extendRule $\equiv$ *extendRule*
begin

definition *extend*
  where *extend forms seq* ≡
    if (*succ seq* = *Em*)
    then (*antec forms* + *antec seq*) ⇒∗ (*succ forms*)
    else (*antec forms* + *antec seq* ⇒∗ *succ seq*)

definition *extendRule*
  where *extendRule forms R* ≡ (*map* (*extend forms*) (*fst R*), *extend forms* (*snd R*))

end
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*pr*

Given this, it is possible to have right weakening, where we overwrite the empty formula if it appears as the succedent of the root of a derivation:

lemma *dpWeakR*:
assumes (Γ ⇒∗ *Em*,*n*) ∈ *derivable R*∗
and $R' \subseteq$ *upRules*
and $R = Ax \cup R'$
shows (Γ ⇒∗ *C*,*n*) ∈ *derivable R*∗ — Proof omitted⟨*proof*⟩

Of course, if $C = Em$, then the above lemma is trivial. The burden is on the user not to "use" the empty formula as a normal formula. An invertibility lemma can then be formalised:

lemma *rightInvertible*:
assumes $R' \subseteq$ *upRules* ∧ $R = Ax \cup R'$
and (Γ ⇒∗ *Compound F Fs*,*n*) ∈ *derivable R*∗
and ∀ $r' \in R$. *rightPrincipal* r' (*Compound F Fs*) ⟶ (Γ′ ⇒∗ *E*) ∈ *set* (*fst* r')
and *E* ≠ *Em*
shows ∃ *m*≤*n*. (Γ +Γ′ ⇒∗ *E*,*m*) ∈ *derivable R*∗
⟨*proof*⟩
lemma *leftInvertible*:
assumes $R' \subseteq$ *upRules* ∧ $R = Ax \cup R'$
and (Γ ⊕ *Compound F Fs* ⇒∗ δ,*n*) ∈ *derivable R*∗
and ∀ $r' \in R$. *leftPrincipal* r' (*Compound F Fs*) ⟶ (Γ′ ⇒∗ *Em*) ∈ *set* (*fst* r')
shows ∃ *m*≤*n*. (Γ +Γ′ ⇒∗ δ,*m*) ∈ *derivable R*∗
⟨*proof*⟩

G3ip can be expressed in this formalism:

inductive-set *g3ip*
where
  conL: ([⦇ *A* ⦈ + ⦇ *B* ⦈ ⇒∗ *Em*], ⦇ *A* ∧∗ *B* ⦈ ⇒∗ *Em*) ∈ *g3ip*
| *conR*: ([∅ ⇒∗ *A*, ∅ ⇒∗ *B*], ∅ ⇒∗ (*A* ∧∗ *B*)) ∈ *g3ip*
| *disL*: ([⦇ *A* ⦈ ⇒∗ *Em*, ⦇ *B* ⦈ ⇒∗ *Em*], ⦇ *A* ∨∗ *B*⦈ ⇒∗ *Em*) ∈ *g3ip*
| *disR1*: ([∅ ⇒∗ *A*], ∅ ⇒∗ (*A* ∨∗ *B*)) ∈ *g3ip*
| *disR2*: ([∅ ⇒∗ *B*], ∅ ⇒∗ (*A* ∨∗ *B*)) ∈ *g3ip*
| *impL*: ([⦇ *A* ⊃ *B* ⦈ ⇒∗ *A*, ⦇ *B* ⦈ ⇒∗ *Em*], ⦇ (*A* ⊃ *B*) ⦈ ⇒∗ *Em*) ∈ *g3ip*
| *impR*: ([⦇ *A* ⦈ ⇒∗ *B*], ∅ ⇒∗ (*A* ⊃ *B*)) ∈ *g3ip*
⟨*proof*⟩

As expected, $R\supset$ can be shown invertible:

lemma *impRInvert*:
assumes $(\Gamma \Rightarrow* (A \supset B), n) \in$ *derivable* $(Ax \cup g3ip)*$ **and** $B \neq Em$
shows $\exists\ m{\leq}n.\ (\Gamma \oplus A \Rightarrow* B, m) \in$ *derivable* $(Ax \cup g3ip)*$
⟨*proof*⟩

5 First-Order Calculi

To formalise first-order results we use the package *Nominal Isabelle*. The details, for the most part, are the same as in §2. However, we lose one important feature: that of polymorphism.

Recall we defined formulae as being indexed by a type of connectives. We could then give abbreviations for these indexed formulae. Unfortunately this feature (indexing by types) is not yet supported in *Nominal Isabelle*. Nested datatypes are also not supported. Thus, strings are used for the connectives (both propositional and first-order) and lists of formulae are simulated to nest via a mutually recursive definition:

nominal-datatype *form* = *At nat var list*
| *Cpd0 string form-list*
| *Cpd1 string «var»form* (‹- (∇ [-].-)›)
| *ff*
and *form-list* = *FNil*
| *FCons form form-list*

Formulae are quantified over a single variable at a time. This is a restriction imposed by *Nominal Isabelle*.

There are two new uniprincipal rule sets in addition to the propositional rule set: first-order rules without a freshness proviso and first-order rules with a freshness proviso. Freshness provisos are particularly easy to encode in *Nominal Isabelle*. We also show that the rules with a freshness proviso form a subset of the first-order rules. The function `set-of-prem` takes a list of premisses, and returns all the formulae in that list:

inductive-set *provRules* **where**
⟦ *mset c* = ⦇ *F* ∇ [*x*].*A* ⦈ ; *ps* ≠ [] ; *x* ♯ *set-of-prem* (*ps* − *A*)⟧
⟹ (*ps*,*c*) ∈ *provRules*

inductive-set *nprovRules* **where**
⟦ *mset c* = ⦇ *F* ∇ [*x*].*A* ⦈ ; *ps* ≠ [] ⟧
⟹ (*ps*,*c*) ∈ *nprovRules*

lemma *nprovContain*:
shows *provRules* ⊆ *nprovRules*
⟨*proof*⟩⟨*proof*⟩

Substitution is defined in the usual way:

nominal-primrec
  subst-form :: *var* ⇒ *var* ⇒ *form* ⇒ *form* (‹[-,-]-›)
and *subst-forms* :: *var* ⇒ *var* ⇒ *form-list* ⇒ *form-list* (‹[-,-]-›)
where
  [*z*,*y*](*At P xs*) = *At P* ([*z*;*y*]*xs*)
| *x*♯(*z*,*y*) ⟹ [*z*,*y*](*F* ∇ [*x*].*A*) = *F* ∇ [*x*].([*z*,*y*]*A*)
| [*z*,*y*](*Cpd0 F Fs*) = *Cpd0 F* ([*z*,*y*]*Fs*)
| [*z*,*y*]*ff* = *ff*
| [*z*,*y*]*FNil* = *FNil*
| [*z*,*y*](*FCons f Fs*) = *FCons* ([*z*,*y*]*f*) ([*z*,*y*]*Fs*)⟨*proof*⟩

Substitution is extended to multisets in the obvious way.

To formalise the condition "no specific substitutions", an inductive predicate is introduced. If some formula in the multiset Γ is a non-trivial substitution, then `multSubst` Γ:

definition *multSubst* :: *form multiset* ⇒ *bool* **where**
multSubst-def: *multSubst* Γ ≡ (∃ *A* ∈ (*set-mset* Γ). ∃ *x y B*. [*y*,*x*]*B* = *A* ∧ *y*≠*x*)

The notation $[z;y]xs$ stands for substitution of a variable in a variable list. The details are simple, and so are not shown.

Extending the rule sets with passive parts depends upon which kind of active part is being extended. The active parts with freshness contexts have additional constraints upon the multisets which are added:

inductive-set *extRules* :: *rule set* ⇒ *rule set* (‹ -*›)
  for *R* :: *rule set*
  where
  id: ⟦ *r* ∈ *R* ; *r* ∈ *Ax* ⟧ ⟹ *extendRule S r* ∈ *R*∗
| *sc*: ⟦ *r* ∈ *R* ; *r* ∈ *upRules* ⟧ ⟹ *extendRule S r* ∈ *R*∗
| *np*: ⟦ *r* ∈ *R* ; *r* ∈ *nprovRules* ⟧ ⟹ *extendRule S r* ∈ *R*∗
| *p*: ⟦ (*ps*,*c*) ∈ *R* ; (*ps*,*c*) ∈ *provRules* ; *mset c* = ⦇ *F* ∇ [*x*].*A* ⦈ ; *x* ♯ *set-of-seq S* ⟧
    ⟹ *extendRule S* (*ps*,*c*) ∈ *R*∗
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*pro*

The final clause says we can only use an S which is suitable fresh.

The only lemma which is unique to first-order calculi is the Substitution Lemma. We show the crucial step in the proof; namely that one can substitute a fresh variable into a formula and the resultant formula is unchanged. The proof is not particularly edifying and is omitted:

lemma *formSubst*:
shows *y* ♯ *x* ∧ *y* ♯ *A* ⟹ *F* ∇ [*x*].*A* = *F* ∇ [*y*].([*y*,*x*]*A*)⟨*proof*⟩

Using the above lemma, we can change any sequent to an equivalent new sequent which does not contain certain variables. Therefore, we can extend with any sequent:

lemma *extend-for-any-seq*:
fixes *S* :: *sequent*
assumes *rules*: *R1* ⊆ *upRules* ∧ *R2* ⊆ *nprovRules* ∧ *R3* ⊆ *provRules*

and *rules2*: $R = Ax \cup R1 \cup R2 \cup R3$
and *rin*: $r \in R$
shows *extendRule* $S\ r \in R*$
$\langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle$

We can then give the two inversion lemmata. The principal case (where the last inference had a freshness proviso) for the right inversion lemma is shown:

lemma *rightInvert*:
fixes $\Gamma\ \Delta$ :: *form multiset*
assumes *rules*: $R1 \subseteq upRules \wedge R2 \subseteq nprovRules \wedge R3 \subseteq provRules \wedge R = Ax \cup R1 \cup R2 \cup R3$
and *a*: $(\Gamma \Rightarrow* \Delta \oplus F\ \nabla\ [x].A,n) \in derivable\ R*$
and *b*: $\forall\ r' \in R.\ rightPrincipal\ r'\ (F\ \nabla\ [x].A) \longrightarrow (\Gamma' \Rightarrow* \Delta') \in set\ (fst\ r')$
and *c*: $\neg\ multSubst\ \Gamma' \wedge \neg\ multSubst\ \Delta'$
shows $\exists\ m \leq n.\ (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta', m) \in derivable\ R* \langle proof \rangle$

lemma *leftInvert*:
fixes $\Gamma\ \Delta$ :: *form multiset*
assumes *rules*: $R1 \subseteq upRules \wedge R2 \subseteq nprovRules \wedge R3 \subseteq provRules \wedge R = Ax \cup R1 \cup R2 \cup R3$
and *a*: $(\Gamma \oplus F\ \nabla\ [x].A \Rightarrow* \Delta,n) \in derivable\ R*$
and *b*: $\forall\ r' \in R.\ leftPrincipal\ r'\ (F\ \nabla\ [x].A) \longrightarrow (\Gamma' \Rightarrow* \Delta') \in set\ (fst\ r')$
and *c*: $\neg\ multSubst\ \Gamma' \wedge \neg\ multSubst\ \Delta'$
shows $\exists\ m \leq n.\ (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta', m) \in derivable\ R* \langle proof \rangle$

In both cases, the assumption labelled *c* captures the "no specific substitution" condition. Interestingly, it is never used throughout the proof. This highlights the difference between the object- and meta-level existential quantifiers.

Owing to the lack of indexing within datatypes, it is difficult to give an example demonstrating these results. It would be little effort to change the theory file to accommodate type variables when they are supported in *Nominal Isabelle*, at which time an example would be simple to write.

$\langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle \langle proof \rangle$

6 Modal Calculi

Some new techniques are needed when formalising results about modal calculi. A set of modal operators must index formulae (and sequents and rules), there must be a method for modalising a multiset of formulae and we need to be able to handle implicit weakening rules.

The first of these is easy; instead of indexing formulae by a single type variable, we index on a pair of type variables, one which contains the propositional connectives, and one which contains the modal operators:

datatype $('a, 'b)\ form =$ *At nat*
| *Compound* $'a\ ('a, 'b)\ form\ list$
| *Modal* $'b\ ('a, 'b)\ form\ list$
| *ff*

datatype-compat *form*

overloading
uniqueConclusion ≡ *uniqueConclusion*
modaliseMultiset ≡ *modaliseMultiset*
begin

definition *uniqueConclusion* :: (′*a*,′*b*) *rule set* ⇒ *bool*
where *uniqueConclusion R* ≡ ∀ *r1* ∈ *R*. ∀ *r2* ∈ *R*. (*snd r1* = *snd r2*) ⟶ (*r1* =*r2*)

Modalising multisets is relatively straightforward. We use the notation ! · Γ, where ! is a modal operator and Γ is a multiset of formulae:

definition *modaliseMultiset* :: ′*b* ⇒ (′*a*,′*b*) *form multiset* ⇒ (′*a*,′*b*) *form multiset*
where *modaliseMultiset a Γ* ≡ {# *Modal a* [*p*]. *p* ∈# *Γ* #}

end

Similarly to §5, two new rule sets are created. The first are the normal modal rules:

inductive-set *modRules2* **where**
⟦ *ps* ≠ [] ; *mset c* = ⦇ *Modal M Ms* ⦈ ⟧ ⟹ (*ps*,*c*) ∈ *modRules2*

The second are the *modalised context rules*. Taking a subset of the normal modal rules, we extend using a pair of modalised multisets for context. We create a new inductive rule set called `p-e`, for "prime extend", which takes a set of modal active parts and a pair of modal operators (say ! and •), and returns the set of active parts extended with ! · Γ ⇒ • · Δ:

inductive-set *p-e* :: (′*a*,′*b*) *rule set* ⇒ ′*b* ⇒ ′*b* ⇒ (′*a*,′*b*) *rule set*
for *R* :: (′*a*,′*b*) *rule set* **and** *M N* :: ′*b*
where
⟦ (*Ps*, *c*) ∈ *R* ; *R* ⊆ *modRules2* ⟧ ⟹ *extendRule* (*M*·*Γ* ⇒∗ *N*·*Δ*) (*Ps*, *c*) ∈ *p-e R M N*

We need a method for extending the conclusion of a rule without extending the premisses. Again, this is simple:

overloading *extendConc* ≡ *extendConc*
begin

definition *extendConc* :: (′*a*,′*b*) *sequent* ⇒ (′*a*,′*b*) *rule* ⇒ (′*a*,′*b*) *rule*
where *extendConc S r* ≡ (*fst r*, *extend S* (*snd r*))

end

The extension of a rule set is now more complicated; the inductive definition has four clauses, depending on the type of rule:

inductive-set *ext* :: (′*a*,′*b*) *rule set* ⇒ (′*a*,′*b*) *rule set* ⇒ ′*b* ⇒ ′*b* ⇒ (′*a*,′*b*) *rule set*

for $R\ R' :: ('a,'b)\ rule\ set$ **and** $M\ N :: 'b$
where
ax: $[\![\ r \in R\ ;\ r \in Ax\ ]\!] \Longrightarrow extendRule\ seq\ r \in ext\ R\ R'\ M\ N$
| up: $[\![\ r \in R\ ;\ r \in upRules]\!] \Longrightarrow extendRule\ seq\ r \in ext\ R\ R'\ M\ N$
| $mod1$: $[\![\ r \in p\text{-}e\ R'\ M\ N\ ;\ r \in R\ ]\!] \Longrightarrow extendConc\ seq\ r \in ext\ R\ R'\ M\ N$
| $mod2$: $[\![\ r \in R\ ;\ r \in modRules2\ ]\!] \Longrightarrow extendRule\ seq\ r \in ext\ R\ R'\ M\ N$

Note the new rule set carries information about which set contains the modalised context rules and which modal operators which extend those prime parts.

$\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle proof\rangle\langle pr$

We have two different inversion lemmata, depending on whether the rule was a modalised context rule, or some other kind of rule. We only show the former, since the latter is much the same as earlier proofs. The interesting cases are picked out:

lemma *rightInvert*:
fixes $\Gamma\ \Delta :: ('a,'b)\ form\ multiset$
assumes *rules*: $R1 \subseteq upRules \wedge R2 \subseteq modRules2 \wedge R3 \subseteq modRules2 \wedge$
$R = Ax \cup R1 \cup (p\text{-}e\ R2\ M1\ M2) \cup R3 \wedge$
$R' = Ax \cup R1 \cup R2 \cup R3$
and a: $(\Gamma \Rightarrow* \Delta \oplus Modal\ M\ Ms,n) \in derivable\ (ext\ R\ R2\ M1\ M2)$
and b: $\forall\ r' \in R'.\ rightPrincipal\ r'\ (Modal\ M\ Ms)\ R' \longrightarrow$
$(\Gamma' \Rightarrow* \Delta') \in set\ (fst\ r')$
and *neq*: $M2 \neq M$
shows $\exists\ m{\leq}n.\ (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta',m) \in derivable\ (ext\ R\ R2\ M1\ M2)$
$\langle proof\rangle\langle proof\rangle$

We guarantee no other rule has the same modal operator in the succedent of a modalised context rule using the condition $M \neq M_2$. Note this lemma only allows one kind of modalised context rule. In other words, it could not be applied to a calculus with the rules:

$$\frac{!\cdot\Gamma \Rightarrow A, \bullet\cdot\Delta}{\Gamma', !\cdot\Gamma \Rightarrow \bullet A, \bullet\cdot\Delta, \Delta'}\ R_1 \qquad \frac{\bullet\cdot\Gamma \Rightarrow A, !\cdot\Delta}{\Gamma', \bullet\cdot\Gamma \Rightarrow \bullet A, !\cdot\Delta, \Delta'}\ R_2$$

since, if $([\emptyset \Rightarrow A], \emptyset \Rightarrow \bullet A) \in \mathcal{R}$, then $R_1 \in$ p-e $\mathcal{R}\ !\ \bullet$, whereas $R_2 \in$ p-e $\mathcal{R}\ \bullet\ !$. Similarly, we cannot have modalised context rules which have more than one modalised multiset in the antecedent or succedent of the active part. For instance:

$$\frac{!\cdot\Gamma_1, \bullet\cdot\Gamma_2 \Rightarrow A, !\cdot\Delta_1, \bullet\cdot\Delta_2}{\Gamma', !\cdot\Gamma_1, \bullet\cdot\Gamma_2 \Rightarrow \bullet A, !\cdot\Delta_1, \bullet\cdot\Delta_2, \Delta'}$$

cannot belong to any `p-e` set. It would be a simple matter to extend the definition of `p-e` to take a *set* of modal operators, however this has not been done.

As an example, classical modal logic can be formalised. The (modal) rules for this calculus are then given in two sets, the latter of which will be extended with $\Box\cdot\Gamma \Rightarrow \Diamond\cdot\Delta$:

inductive-set *g3mod2*
where

diaR: ([∅ ⇒∗ ⁅ *A* ⁆], ∅ ⇒∗ ⁅ ◇ *A* ⁆) ∈ *g3mod2*
| *boxL*: ([⁅ *A* ⁆ ⇒∗ ∅], ⁅ □ *A* ⁆ ⇒∗ ∅) ∈ *g3mod2*

inductive-set *g3mod1*
where
boxR: ([∅ ⇒∗ ⁅*A*⁆],∅ ⇒∗ ⁅ □ *A* ⁆) ∈ *g3mod1*
| *diaL*: ([⁅*A*⁆ ⇒∗ ∅],⁅ ◇ *A* ⁆ ⇒∗ ∅) ∈ *g3mod1*
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩

We then show the strong admissibility of the rule:

$$\frac{\Gamma \Rightarrow \Box A, \Delta}{\Gamma \Rightarrow A, \Delta}$$

lemma *invertBoxR*:
assumes *R* = *Ax* ∪ *g3up* ∪ (*p-e g3mod1* □ ◇) ∪ *g3mod2*
and (*Γ* ⇒∗ *Δ* ⊕ (□ *A*),*n*) ∈ *derivable* (*ext R g3mod1* □ ◇)
shows ∃ *m*≤*n*. (*Γ* ⇒∗ *Δ* ⊕ *A*,*m*) ∈ *derivable* (*ext R g3mod1* □ ◇)
⟨*proof*⟩

where *principal* is the result which fulfils the principal formula conditions given in the inversion lemma, and *g3* is a result about rule sets.

⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*pro*

7 Manipulating Rule Sets

The removal of superfluous and redundant rules [1] will not be harmful to invertibility: removing rules means that the conditions of earlier sections are more likely to be fulfilled. Here, we formalise the results that the removal of such rules from a calculus $\mathcal{L}$ will create a new calculus $\mathcal{L}'$ which is equivalent. In other words, if a sequent is derivable in $\mathcal{L}$, then it is derivable in $\mathcal{L}'$. The results formalised in this section are for uniprincipal multisuccedent calculi.

When dealing with lists of premisses, a rule R with premisses P will be redundant given a rule R' with premisses P' if there exists some p such that $P = p\#P'$. There are other ways in which a rule could be redundant; say if $P = Q@P'$, or if $P = P'@Q$, and so on. The order of the premisses is not really important, since the formalisation operates on the finite set based upon the list. The more general "append" lemma could be proved from the lemma we give; we prove the inductive step case in the proof of such an append lemma. This is a height preserving transformation. Some of the proof is shown:

lemma *removeRedundant*:
assumes *r1* = (*p*#*ps*,*c*) ∧ *r1* ∈ *upRules*
and *r2* = (*ps*,*c*) ∧ *r2* ∈ *upRules*
and *R1* ⊆ *upRules* ∧ *R* = *Ax* ∪ *R1*
and (*T*,*n*) ∈ *derivable* (*R* ∪ {*r1*} ∪ {*r2*})∗
shows ∃ *m*≤*n*. (*T*,*m*) ∈ *derivable* (*R* ∪ {*r2*})∗
⟨*proof*⟩

Recall that to remove superfluous rules, we must know that Cut is admissible in the original calculus [1]. Again, we add the two distinguished premisses at the head of the premiss list; general results about permutation of lists will achieve a more general result. Since one uses Cut in the proof, this will in general not be height-preserving:

lemma *removeSuperfluous*:
assumes $r1 = ((\emptyset \Rightarrow\!* \lbag A \rbag) \# ((\lbag A \rbag \Rightarrow\!* \emptyset) \# ps), c) \wedge r1 \in upRules$
and $R1 \subseteq upRules \wedge R = Ax \cup R1$
and $(T,n) \in derivable\ (R \cup \{r1\})*$
and CA: $\forall\ \Gamma\ \Delta\ A.\ ((\Gamma \Rightarrow\!* \Delta \oplus A) \in derivable'\ R* \longrightarrow$
$(\Gamma \oplus A \Rightarrow\!* \Delta) \in derivable'\ R*) \longrightarrow$
$(\Gamma \Rightarrow\!* \Delta) \in derivable'\ R*$
shows $T \in derivable'\ R*\langle proof \rangle$

Combinable rules can also be removed. We encapsulate the combinable criterion by saying that if $(p\#P, T)$ and $(q\#P, T)$ are rules in a calculus, then we get an equivalent calculus by replacing these two rules by $((\text{extend } p\ q)\#P, T)$. Since the `extend` function is commutative, the order of p and q in the new rule is not important. This transformation is height preserving:

lemma *removeCombinable*:
assumes *a*: $r1 = (p \#\ ps, c) \wedge r1 \in upRules$
and *b*: $r2 = (q \#\ ps, c) \wedge r2 \in upRules$
and *c*: $r3 = (extend\ p\ q \#\ ps,\ c) \wedge r3 \in upRules$
and *d*: $R1 \subseteq upRules \wedge R = Ax \cup R1$
and $(T,n) \in derivable\ (R \cup \{r1\} \cup \{r2\})*$
shows $(T,n) \in derivable\ (R \cup \{r3\})*\langle proof \rangle$

8 Conclusions

Only a portion of the formalisation was shown; a variety of intermediate lemmata were not made explicit. This was necessary, for the *Isabelle* theory files run to almost 8000 lines. However, these files do not have to be replicated for each new calculus. It takes very little effort to define a new calculus. Furthermore, proving invertibility is now a quick process; less than 25 lines of proof in most cases.

theory *SequentInvertibility*
imports *MultiSequents SingleSuccedent NominalSequents ModalSequents SRCTransforms*
begin

end

References

1. A. Avron and I. Lev. Canonical propositional Gentzen-type systems. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy,*

June 18-23, 2001, Proceedings, volume 2083 of *Lecture Notes in Computer Science*. Springer, 2001.