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:

\begin{verbatim}
datatype 'a form = At nat \\
    | Compound 'a 'a form list \\
    | ff
\end{verbatim}

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

\begin{verbatim}
datatype Gp = con | dis | imp 

type-synonym Gp-form = Gp form
\end{verbatim}
abbreviation **con-form** (infixl \(\land\) 80) where  
\(p \land q \equiv \text{Compound con } [p,q]\)

abbreviation **dis-form** (infixl \(\lor\) 80) where  
\(p \lor q \equiv \text{Compound dis } [p,q]\)

abbreviation **imp-form** (infixl \(\supset\) 80) where  
\(p \supset q \equiv \text{Compound imp } [p,q]\)

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 \(\oplus\).

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  
\(id\): (\(\langle\rangle, [\langle At i \rangle \Rightarrow * \langle At i \rangle] \rangle \in Ax\)  
\(Lbot\): (\(\langle\rangle, \langle ff \rangle \Rightarrow * \langle\rangle \rangle \in Ax\)

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  
\(I\): (\([\langle mset c \equiv \langle Compound R Fs \rangle \Rightarrow * \langle\rangle \rangle, ps \neq \langle\rangle \] \Rightarrow (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  
\(conL\): (\(\langle\rangle A \oplus B \Rightarrow * \langle\rangle, \langle A \land B \rangle \Rightarrow * \langle\rangle \rangle \in g3cp\)
\(conR\): (\(\langle\rangle \Rightarrow * \langle A \rangle \oplus B, \langle\rangle \Rightarrow * \langle A \land B \rangle \rangle \in g3cp\)
\(disL\): (\(\langle\rangle A \Rightarrow * \langle\rangle, \langle A \lor B \rangle \Rightarrow * \langle\rangle \rangle \in g3cp\)
\(disR\): (\(\langle\rangle \Rightarrow * \langle A \oplus B \rangle, \langle\rangle \Rightarrow * \langle A \lor B \rangle \rangle \in g3cp\)
\(impL\): (\(\langle\rangle \Rightarrow * \langle A \rangle \oplus B, \langle A \Rightarrow * \langle\rangle \rangle \Rightarrow * \langle\rangle \rangle \in g3cp\)
\(impR\): (\(\langle\rangle A \Rightarrow * \langle B \rangle \rangle \Rightarrow * \langle\rangle, \langle\rangle A \Rightarrow * \langle B \rangle \rangle \Rightarrow * \langle\rangle \rangle \in g3cp\)

**lemma** \(g3cp-upRules: shows g3cp \subseteq upRules\)

**proof** –
{ 
    fix ps c
    assume (ps,c) ∈ g3cp
    then have (ps,c) ∈ upRules by (induct) auto
}
thus g3cp ⊆ upRules by auto
qed

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 ∈ Ps and C with S:

overloading
extent ≡ extent
extendRule ≡ extendRule
begin
definition extend
where extent forms seq ≡ (antec forms + antec seq) ⇒∗ (succ forms + succ seq)
definition extendRule
where extendRule forms R ≡ (map (extent forms) (fst R), extent forms (snd R))
end

Given a rule set R, the extension of R, called R*, 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
apply (auto simp add:uniqueConclusion-def Ball-def)
apply (rule g3cp.cases) apply auto by (rotate-tac 1,rule g3cp.cases,auto)
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:

\[
\text{inductive } \text{rightPrincipal} :: \ 'a \text{ rule } \Rightarrow \ 'a \text{ form } \Rightarrow \text{ bool}
\]

where

\[
\text{up: } C = (\emptyset \Rightarrow \ast \text{ Compound } FFs) \implies \text{rightPrincipal} (Ps, C) (\text{Compound } FFs)
\]

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

**lemma principal-means-premiss:**

assumes \( a: \text{rightPrincipal } r (A \land \ast B) \)

and \( b: r \in g3cp \)

shows \( (\emptyset \Rightarrow \ast \text{ Compound } A I ) \in \text{set } (fst r) \)

proof

- from \( a \) and \( b \) obtain \( Ps \) where req: \( r = (Ps, \emptyset \Rightarrow \ast \text{ Compound } A I ) \)
  - by (cases \( r \)) auto
  - with \( b \) have \( Ps = [\emptyset \Rightarrow \ast \text{ Compound } A I , \emptyset \Rightarrow \ast \text{ Compound } B I ] \)
    - apply (cases \( r \)) by (rule g3cp.cases) auto
    - with req show \( (\emptyset \Rightarrow \ast \text{ Compound } A I ) \in \text{set } (fst r) \) by auto

qed

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 \( R \) iff \( (S, n) \) belongs to the inductive set \( \text{derivable } R \):

\[
\text{inductive-set } \text{derivable} :: \ 'a \text{ rule set } \Rightarrow \ 'a \text{ deriv set}
\]

for \( R :: \ 'a \text{ rule set} \)

where

\[
\text{base: } [([], C) \in R] \implies (C, 0) \in \text{derivable } R
\]

| step: | \( r \in R; (fst r) \neq []; \forall p \in \text{set } (fst r). \exists n \leq m. (p, n) \in \text{derivable } R \]

\implies (snd \( r \), \( m + 1 \)) \in \text{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 \( \text{derivable}' \), which is a set of sequents:

\[
\text{inductive-set } \text{derivable'} :: \ 'a \text{ rule set } \Rightarrow \ 'a \text{ sequent set}
\]

for \( R :: \ 'a \text{ rule set} \)

where

\[
\text{base: } [([], C) \in R] \implies C \in \text{derivable}' R
\]

| step: | \( r \in R; (fst r) \neq []; \forall p \in \text{set } (fst r). p \in \text{derivable'} R \]

\implies (snd \( r \)) \in \text{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 \((C,n) \in \text{derivable } R\)
shows \(C \in \text{derivable'} R\)
using assms by (induct) auto

lemma deriv-to-deriv2:
assumes \(C \in \text{derivable'} R\)
shows \(\exists \ n. \ (C,n) \in \text{derivable } R\)
using assms proof (induct)
  case (base C)
  then have \((C,0) \in \text{derivable } R\) by auto
  then show \(?case\) by blast
next
  case (step r)
  then obtain \(ps \ c\) where \(r = (ps,c)\) and \(ps \neq []\) by (cases r) auto
  with step(\(3\)) have \(aa: \forall \ p \in \text{set } ps. \ \exists \ n. \ (p,n) \in \text{derivable } R\) by auto
  then have \(\exists \ m. \ \forall \ p \in \text{set } ps. \ \exists \ n \leq m. \ (p,n) \in \text{derivable } R\)
  proof (induct ps) — induction on the list
    case Nil
    then show \(?case\) by auto
  next
    case (Cons a as)
    then have \(\exists \ m. \ \forall \ p \in \text{set } as. \ \exists \ n \leq m. \ (p,n) \in \text{derivable } R\) by auto
    then obtain \(m\) where \(\forall \ p \in \text{set } as. \ \exists \ n \leq m. \ (p,n) \in \text{derivable } R\) have
      \(\exists \ n. \ (a,n) \in \text{derivable } R\) by auto
    then obtain \(m'\) where \(a,m') \in \text{derivable } R\) by blast
    ultimately have \(\forall \ p \in \text{set } (a \# as). \ \exists \ n \leq (\text{max } m \ m'). \ (p,n) \in \text{derivable } R\)
    by auto — max returns the maximum of two integers
    then show \(?case\) by blast
  qed
then obtain \(m\) where \(\forall \ p \in \text{set } ps. \ \exists \ n \leq m. \ (p,n) \in \text{derivable } R\) by blast
with \(r = (ps,c)\) and \(r \in R\) have \((c,m+1) \in \text{derivable } R\) using \(ps \neq []\), and
  derivable.step[where \(r=(ps,c)\) and \(R=R\) and \(m=m\)] by auto
then show \(?case\) using \(r = (ps,c)\) by auto
qed

### 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.
The height of derivations is decided by the length of the longest branch. Thus, we need to use strong induction: i.e. $\forall m \leq n$. If $P(m)$ then $P(n + 1)$.

**proof** (induct $n$ arbitrary; $\Gamma \vdash \Delta$ rule:nat-less-induct)

**case** ($1 \in \Gamma \vdash \Delta$)

then have $IH: \forall m < n. \forall \Gamma \vdash \Delta. \ (\Gamma \Rightarrow \Delta \oplus \text{Compound } FFs, m) \in \text{derivable } R^*$

and

$a': (\Gamma \Rightarrow \Delta \oplus \text{Compound } FFs,n) \in \text{derivable } R^*$

and

$b': \forall r' \in R. \text{rightPrincipal } r' (\text{Compound } FFs) \longrightarrow (\Gamma \Rightarrow \Delta') \in \text{set } (\text{fst } r')$

by auto

**show** $\exists$case

**proof** (cases $n$) — Case analysis on $n$

**case** $\emptyset$

then obtain $r$ where $\text{extendRule } S \ r = ([],\Gamma \Rightarrow \Delta \oplus \text{Compound } FFs)$

and $r \in \text{Ax } \lor r \in R'$ by auto — At height 0, the premises are empty

moreover

{ assume $r \in \text{Ax}$

then obtain $i$ where $([],\emptyset \vdash \emptyset \vdash \emptyset) = r \lor$

$r = ([],\emptyset \vdash \emptyset)$

using $\text{characteriseAx}$[where $r=r$] by auto

moreover — Case split on the kind of axiom used

{ assume $r = ([],\emptyset \vdash \emptyset)$

then have $At i \in \# \Gamma \land At i \in \# \Delta$ by auto

then have $At i \in \# \Gamma' \land At i \in \# \Delta' \text{ by auto}$

then have $\ (\Gamma + \Gamma' \Rightarrow \Delta + \Delta',0) \in \text{derivable } R^*$ using rules by auto

} moreover

{ assume $r = ([],\emptyset \vdash \emptyset)$

then have $\# \Gamma$ by auto

then have $\# \Gamma' \text{ by auto}$

then have $\ (\Gamma + \Gamma' \Rightarrow \Delta + \Delta',0) \in \text{derivable } R^*$ using rules by auto

} ultimately have $\ (\Gamma + \Gamma' \Rightarrow \Delta + \Delta',0) \in \text{derivable } R^*$ by blast

moreover

{ assume $r \in R'$ — This leads to a contradiction

then obtain $Ps \ C$ where $Ps \neq []$ and $r = (Ps,C)$ by auto

moreover obtain $S$ where $r = ([],S)$ by blast — Contradiction

ultimately have $\ (\Gamma + \Gamma' \Rightarrow \Delta + \Delta',0) \in \text{derivable } R^*$ using rules by simp
ultimately show $\exists \, m \leq n. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{derivable } R*$ by blast

In the case where $n = n' + 1$ for some $n'$, we know the premisses are empty, and every premiss is derivable at a height lower than $n'$:

case $(\text{Suc } n')$
then have $(\Gamma \Rightarrow \Delta \oplus \text{Compound } F \, Fs, n'+1) \in \text{derivable } R*$ using $a'$ by simp
then obtain $Ps$ where $(Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \, Fs) \in R*$ and $Ps \neq []$ and
\[ \forall \, p \in \text{set } Ps. \exists \, n \leq n'. \, (p, n) \in \text{derivable } R* \text{ by auto} \]
then obtain $r \, S$ where $r \in Ax \lor r \in R'$
and extendRule $S \, r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \, Fs)$ by auto
moreover
\[ \{ \text{assume } r \in Ax \quad — \text{Gives a contradiction} \} \]
then have $\text{fst } r = []$ apply (cases $r$) by (rule $Ax\. cases$) auto
moreover obtain $x \, y$ where $r = (x, y)$ by (cases $r$)
then have $x \neq []$ using ($Ps \neq []$)
and (extendRule $S \, r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \, Fs)$) by auto
ultimately have $\exists \, m \leq n. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{derivable } R*$ by auto

moreover
\[ \{ \text{assume } r \in R' \} \]
obtain $ps \, c$ where $r = (ps, c)$ by (cases $r$) auto
have (rightPrincipal $r$ (Compound $F \, Fs)) \lor 
\neg (\text{rightPrincipal } r \, (\text{Compound } F \, Fs))
by blast — The formula is principal, or not

If the formula is principal, then $\Gamma' \Rightarrow \Delta'$ is amongst the premisses of $r$:

\[ \{ \text{assume } \text{rightPrincipal } r \, (\text{Compound } F \, Fs) \} \]
then have $(\Gamma' \Rightarrow \Delta') \in \text{set } Ps$ using $b'$ by auto
then have extend $S \, (\Gamma' \Rightarrow \Delta') \in \text{set } Ps$
using (extendRule $S \, r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \, Fs)$)
by (simp)
moreover have $S = (\Gamma \Rightarrow \Delta)$ by (cases $S$) auto
ultimately have $(\Gamma + \Gamma' \Rightarrow \Delta + \Delta') \in \text{set } Ps$ by (simp add:extend-def)
then have $\exists \, m \leq n'. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{derivable } R*$
using $\forall \, p \in \text{set } Ps. \exists \, n \leq n'. \, (p, n) \in \text{derivable } R*$ by auto
then have $\exists \, m \leq n. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{derivable } R*$ by (auto)

If the formula is not principal, then it must appear in the premisses. The first two lines give a characterisation of the extension and conclusion, respectively.
Then, we apply the induction hypothesis at the lower height of the premisses:

\[ \{ \text{assume } \neg \text{rightPrincipal } r \, (\text{Compound } F \, Fs) \} \]
obtain $\Phi \, \Psi$ where $S = (\Phi \Rightarrow \Psi)$ by (cases $S$) (auto)
then obtain $G \, H$ where $c = (G \Rightarrow H)$ by (cases $c$) (auto)
then have $\hat{\Psi} \, \text{Compound } F \, Fs \, \hat{\Psi} \neq H$ — Proof omitted
have $\Psi + H = \Delta \oplus \text{Compound } F \, Fs$
using \( \langle S = (\Phi \Rightarrow \Psi) \rangle \), and \( \langle r = (ps,c) \rangle \), and \( \langle c = (G \Rightarrow* H) \rangle \) by auto

moreover from \( \langle r = (ps,c) \rangle \) and \( \langle c = (G \Rightarrow* H) \rangle \).

have \( H = \emptyset \lor (\exists A. H = \{A\}) \) by auto

ultimately have \( \text{Compound F Fs} \in \# \Psi \) — Proof omitted

then have \( \exists \Psi 1. \Psi = \Psi 1 \oplus \text{Compound F Fs} \) by (auto)

then obtain \( \Psi 1 \) where \( S = (\Phi \Rightarrow* \Psi 1 \oplus \text{Compound F Fs}) \) by auto

have \( \forall p \in \text{set Ps}. (\text{Compound F Fs} \in \# \text{succ } p) \) — Appears in every premiss

by (auto)

then have \( \forall p \in \text{set Ps}. \exists \Phi^p \Psi^p' m. m \leq n' \land \Phi^p + \Gamma^p \Rightarrow* \Psi^p' + \Delta^p' m \in \text{derivable } R* \land \)

\( p = (\Phi^p \Rightarrow* \Psi^p \oplus \text{Compound F Fs}) \text{ using } IH \) by (arith)

To this set of new premisses, we apply a new instance of \( r \), with a different extension:

obtain \( \text{Ps'} \) where \( eq: \text{Ps'} = \text{map} (\text{extend} (\Phi^p + \Gamma^p \Rightarrow* \Psi^p 1 + \Delta^p')) \text{ ps by auto} \)

have \( (\text{Ps'}, \Gamma + \Gamma^p \Rightarrow* \Delta + \Delta') \in \text{R* by simp} \)

then have \( \forall p \in \text{set Ps'}, \exists n \leq n'. (p,n) \in \text{derivable } R* \) by auto

then have \( \exists m \leq n. (\Gamma + \Gamma^p \Rightarrow* \Delta + \Delta', m) \in \text{derivable } R* \)

using \( (\text{Ps'}, \Gamma + \Gamma^p \Rightarrow* \Delta + \Delta') \in \text{R*} \) by (auto)

All of the cases are now complete.

ultimately show \( \exists m \leq n. (\Gamma + \Gamma^p \Rightarrow* \Delta + \Delta', m) \in \text{derivable } R* \) by blast

qed

As an example, we show the left premiss of \( R\land \) in \textbf{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 \textit{conRInvert}:

assumes \( (\Gamma \Rightarrow* \Delta \uplus (A \land* B), n) \in \text{derivable } (g3cp \cup Ax)* \)

shows \( \exists m \leq n. (\Gamma \Rightarrow* \Delta \uplus A, m) \in \text{derivable } (g3cp \cup Ax)* \)

proof—

have \( \forall r \in g3cp. \text{rightPrincipal } (A \land* B) \rightarrow (\emptyset \Rightarrow* \{A \, \{\}) \in \text{set } (\text{fst } r) \)

using principal-means-premiss by auto

with \textit{assms} show \( \textit{thesis} \) using \textit{rightInvertible} by (auto)

qed

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

lemma \textit{leftInvertible}:

fixes \( \Gamma \cdot \Delta :: \text{‘a form multiset} \)

assumes rules: \( R' \subseteq \upRules \land R = \text{Ax} \cup R' \)

and \( a: (\Gamma \oplus \text{ Compound F Fs} \Rightarrow* \Delta, n) \in \text{derivable } R* \)

and \( b: \forall r' \in R. \text{leftPrincipal } r' (\text{Compound F Fs}) \rightarrow (\Gamma' \Rightarrow* \Delta') \in \text{set } (\text{fst } r') \)

shows \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow* \Delta + \Delta', m) \in \text{derivable } R* \)

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 ≡
  ∀ n S. (r ∈ R ∧ (snd (extendRule S r), n) ∈ derivable R*) →
  (∀ p ∈ set (fst (extendRule S r)). ∃ m ≤ n. (p, m) ∈ derivable R*)
definition invertible-set
where invertible-set R ≡ ∀ (ps, c) ∈ 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' ⊆ upRules ∧ R = Ax ∪ R'
and uniqueConclusion R'
shows invertible-set R

lemma g3cp-invertible:
shows invertible-set (Ax ∪ g3cp)
using g3cp-uc and g3cp-upRules
and unique-to-invertible[where R'=g3cp and R=Ax ∪ g3cp]
by auto

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 ⊃ from G3ip? The left premiss will be A ⊃ B ⇒ A, and the extension will be Γ ⇒ 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 ≡ extend
extendRule ≡ 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

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 $(\Gamma \Rightarrow Em.n) \in \text{derivable } R*$
and $R' \subseteq \text{upRules}$
and $R = Ax \cup R'$
shows $(\Gamma \Rightarrow C.n) \in \text{derivable } R*$ — Proof omitted

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 $R' \subseteq \text{upRules} \land R = Ax \cup R'$
and $(\Gamma \Rightarrow* \text{Compound } F Fs.n) \in \text{derivable } R*$
and $\forall r' \in R. \text{rightPrincipal } r' (\text{Compound } F Fs) \rightarrow (\Gamma' \Rightarrow* E) \in \text{set (fst } r')$
and $E \neq Em$
shows $\exists m \leq n. (\Gamma + \Gamma' \Rightarrow* E,m) \in \text{derivable } R*$

lemma leftInvertible:
assumes $R' \subseteq \text{upRules} \land R = Ax \cup R'$
and $(\Gamma \oplus \text{Compound } F Fs \Rightarrow* \delta.n) \in \text{derivable } R*$
and $\forall r' \in R. \text{leftPrincipal } r' (\text{Compound } F Fs) \rightarrow (\Gamma' \Rightarrow* Em) \in \text{set (fst } r')$
shows $\exists m \leq n. (\Gamma + \Gamma' \Rightarrow* \delta.m) \in \text{derivable } R*$

G3ip can be expressed in this formalism:
inductive-set $g3ip$

where

\begin{align*}
\text{conL: } & ((\emptyset \Rightarrow \top, \emptyset \Rightarrow \bot, \emptyset \Rightarrow (A \land B) \Rightarrow \bot) \in g3ip) \\
\text{conR: } & ((\emptyset \Rightarrow \top, \emptyset \Rightarrow \bot, \emptyset \Rightarrow (A \land B) \Rightarrow \bot) \in g3ip) \\
\text{disL: } & ((\emptyset \Rightarrow \top, \emptyset \Rightarrow \bot, \emptyset \Rightarrow (A \lor B) \Rightarrow \bot) \in g3ip) \\
\text{disR1: } & ((\emptyset \Rightarrow \top, \emptyset \Rightarrow (A \lor B) \Rightarrow \bot) \in g3ip) \\
\text{disR2: } & ((\emptyset \Rightarrow (A \lor B), \emptyset \Rightarrow (A \lor B)) \in g3ip) \\
\text{impL: } & ((\emptyset \Rightarrow (A \Rightarrow B), \emptyset \Rightarrow (A \Rightarrow B)) \in g3ip) \\
\text{impR: } & ((\emptyset \Rightarrow (A \Rightarrow B), \emptyset \Rightarrow (A \Rightarrow B)) \in g3ip)
\end{align*}

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

\begin{verbatim}
lemma impRInvert: assumes (Γ ⊢ (A ⊃ B), n) ∈ derivable (Ax ∪ g3ip)* and B ≠ Em
shows ∃ m≤n. (∆ ⊢ B, m) ∈ derivable (Ax ∪ g3ip)*
proof
 have ∀ r ∈ (Ax ∪ g3ip). rightPrincipal r (A ⊃ B) ---
   ⟹ (⌜A⌝ ⊢∗ B) ∈ set (fst r)
 proof --- Showing that A ⊢ B is a premiss of every rule with A ⊃ B principal
{fix r
  assume r ∈ (Ax ∪ g3ip)
  moreover assume rightPrincipal r (A ⊃ B)
  ultimately have r ∈ g3ip by auto   --- If A ⊃ B was principal, then r ∉ Ax
  from rightPrincipal r (A ⊃ B) have snd r = (∅ ⊢∗ (A ⊃ B)) by auto
  with r ∈ g3ip and (rightPrincipal r (A ⊃ B))
    have r = (⌜A⌝ ⊢∗ B, ∅ ⊢∗ (A ⊃ B)) by (rule g3ip.cases) auto
    then have (⌜A⌝ ⊢∗ B) ∈ set (fst r) by auto
}
 thus ?thesis by auto
qed
\end{verbatim}

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
definition Cpdl string 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 premises, and returns all the formulae in that list:

$$\text{inductive-set provRules where}$$

\[
\begin{align*}
\text{mset } c = \{ F \triangledown [x].A \} & ; ps \neq [] ; x \notin \text{set-of-prem (ps } - A) ] \\
\implies (ps,c) \in \text{provRules}
\end{align*}
\]

$$\text{inductive-set nprovRules where}$$

\[
\begin{align*}
\text{mset } c = \{ F \triangledown [x].A \} & ; ps \neq [] ] \\
\implies (ps,c) \in \text{nprovRules}
\end{align*}
\]

**lemma** nprovContain: shows provRules $\subseteq$ nprovRules

**proof**

\{
\text{fix } ps \text{ c}
\text{assume } (ps,c) \in \text{provRules}
then have (ps,c) \in \text{nprovRules by } \{\text{cases} \text{ auto}\}
\}
then show ?thesis by auto
\}

qed

Substitution is defined in the usual way:

**nominal-primrec**

\[
\begin{align*}
\text{subst-form } :: \text{ var } \Rightarrow \text{ var } \Rightarrow \text{ form } (\text{[\text{-\text{-}}\text{-}])}
\text{and subst-forms } :: \text{ var } \Rightarrow \text{ var } \Rightarrow \text{ form-list } \Rightarrow \text{ form-list (\text{[\text{-\text{-}}\text{-}])}
\end{align*}
\]

**where**

\[
\begin{align*}
[z,y](\text{At } P \text{ xs}) &= \text{At } P (\text{[z;y]xs}) \\
[x,y]F &\Rightarrow [z,y](F \triangledown [x].A) = F \triangledown [x].([z,y]A)
\end{align*}
\]

\[
\begin{align*}
[z,y](\text{Cpd0 } F \text{ Fs}) &= \text{Cpd0 } F ([z,y]Fs)
\end{align*}
\]

\[
\begin{align*}
[z,y]\text{ff} &= \text{ff}
\end{align*}
\]

\[
\begin{align*}
[z,y]\text{FNil} &= \text{FNil}
\end{align*}
\]

\[
\begin{align*}
[z,y]\text{(FCons } f \text{ Fs}) &= \text{FCons ([z,y]f) ([z,y]Fs)}
\end{align*}
\]

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 $\Gamma$ is a non-trivial substitution, then $\text{multSubst } \Gamma$:
definition \textit{multSub} :: \textit{form multiset} $\Rightarrow$ \textit{bool} where
\textit{multSub-def}: \textit{multSub} $\Gamma$ $\equiv$ ($\exists$ \textit{A} $\in$ (\textit{set-mset} $\Gamma$). \textit{x y B}. [\textit{y},\textit{x}]B = \textit{A}$ \land \textit{y} \neq \textit{x})

The notation $[\textit{z};\textit{y}]\textit{xs}$ 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:

\begin{itemize}
  \item \textbf{inductive-set} \textit{extRules} :: \textit{rule set} $\Rightarrow$ \textit{rule set} \{ $\ast$ \}
  \begin{itemize}
    \item \textbf{id}: $[\textit{r} \in \textit{R} ; \textit{r} \in \textit{Ax}] \Rightarrow \text{extendRule} \textit{S} \textit{r} \in \textit{R}\ast$
    \item \textbf{sc}: $[\textit{r} \in \textit{R} ; \textit{r} \in \textit{upRules}] \Rightarrow \text{extendRule} \textit{S} \textit{r} \in \textit{R}\ast$
    \item \textbf{np}: $[\textit{r} \in \textit{R} ; \textit{r} \in \textit{nprovRules}] \Rightarrow \text{extendRule} \textit{S} \textit{r} \in \textit{R}\ast$
    \item \textbf{p}: $[\textit{(ps,c)} \in \textit{R} ; \textit{(ps,c)} \in \textit{provRules} ; \textit{mset} \textit{c} = \text{\textit{F} } \text{\textit{∇} } [\textit{x}]\textit{A} \, \, \, ; \, \, \, \textit{x} \neq \text{\textit{set-of-seq} S}] \Rightarrow \text{extendRule} \textit{S} \textit{(ps,c)} \in \textit{R}\ast$
  \end{itemize}
\end{itemize}

The final clause says we can only use an \textit{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:

\textbf{lemma} \textit{formSubst}:
s\textbf{shows} $\textit{y} \neq \textit{x} \land \textit{y} \neq \textit{A} \Rightarrow \text{\textit{F} } \text{\textit{∇} } [\textit{x}]\textit{A} = \text{\textit{F} } \text{\textit{∇} } [\textit{y}]\text{\textit{.} }\textit{A}$

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:

\textbf{lemma} \textit{extend-for-any-seq}:
\textbf{fixes} \textit{S} :: \textit{sequent}
\textbf{assumes} \textit{rules}: \textit{R1} $\subseteq$ \textit{upRules} $\land$ \textit{R2} $\subseteq$ \textit{nprovRules} $\land$ \textit{R3} $\subseteq$ \textit{provRules}
\begin{itemize}
  \item and \textit{rules2}: \textit{R} = \textit{Ax} $\cup$ \textit{R1} $\cup$ \textit{R2} $\cup$ \textit{R3}
  \item and \textit{rin}: \textit{r} $\in$ \textit{R}
\end{itemize}
\textbf{shows} \text{\textit{extendRule} \textit{S} \textit{r} $\in$ \textit{R}\ast}

We only show the interesting case: where the last inference had a freshness proviso:

\textbf{assume} \textit{r} $\in$ \textit{R3}
\begin{itemize}
  \item then \textbf{have} \textit{r} $\in$ \textit{provRules} using \textit{rules} by \textit{auto}
  \item obtain \textit{ps c} where \textit{r} = \textit{(ps,c)} by \textit{(cases r)} auto
  \item then \textbf{have} \textit{r1}: \textit{(ps,c)} $\in$ \textit{R}
    \begin{itemize}
      \item and \textit{r2}: \textit{(ps,c)} $\in$ \textit{provRules} using \textit{(r $\in$ \textit{provRules}) and \textit{rin} by \textit{auto}}
    \end{itemize}
    \begin{itemize}
      \item with \textit{(r = \textit{(ps,c)}) obtain} \textit{F x A}
        \begin{itemize}
          \item where \textit{c} = ( (\textit{∅} $\Rightarrow$ \textit{F} $\text{\textit{∇} } [\textit{x}]\textit{A}) \lor$
            \begin{itemize}
              \item \textit{c} = ( (\textit{F} $\text{\textit{∇} } [\textit{x}]\textit{A} \Rightarrow \textit{∅}) \land \textit{x} \neq \text{\textit{set-of-prem} ( \textit{ps} \, - \, \textit{A})}$
            \end{itemize}
          \end{itemize}
        \end{itemize}
    \end{itemize}
    \begin{itemize}
      \item using \textit{provRuleCharacterise} and \textit{(r $\in$ \textit{provRules}) by \textit{auto}}
    \end{itemize}
\end{itemize}
then have \( \text{mset } c = \{ F \triangledown [x].A \} \setminus x \notin \text{set-of-prem (ps - A)} \) by \text{auto}

moreover obtain \( y \) where \( fr: y \notin x \land \)

\( y \notin A \land \)

\( y \notin \text{set-of-seq S} \land \)

\( (y :: \text{var}) \notin \text{set-of-prem (ps-A)} \)

using \text{getFresh by auto}

then have \( fr2: y \notin \text{set-of-seq S} \) by \text{auto}

ultimately have \( \text{mset } c = \{ F \triangledown [y].((y,x)A) \} \setminus y \notin \text{set-of-prem (ps - A)} \)

using \text{formSubst and fr by auto}

then have \( \text{mset } c = \{ F \triangledown [y].((y,x)A) \} \) by \text{auto}

then have \( \text{extendRule S (ps,c)} \in R* \) using \( r1 \) and \( r2 \) and \( fr2 \) and \text{extRules.p by auto}

then have \( \text{extendRule S r } \in R* \) using \( \langle r = (ps,c) \rangle \) by simp

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:

\text{lemma rightInvert:}

\text{fixes } \Gamma \cdot \Delta :: \text{form multiset}

\text{assumes rules: } R1 \subseteq \text{upRules} \land R2 \subseteq \text{nprovRules} \land R3 \subseteq \text{provRules} \land R = Ax \cup R1 \cup R2 \cup R3

\text{and a: } (\Gamma \Rightarrow^* \Delta \oplus F \triangledown [x],A,n) \in \text{derivable R*}

\text{and b: } \forall r' \in R. \text{rightPrincipal r'} (F \triangledown [x],A) \Rightarrow^* (\Gamma' \Rightarrow^* \Delta') \in \text{set (fst r')}

\text{and c: } \sim \text{multSubst } \Gamma' \land \sim \text{multSubst } \Delta'

\text{shows } \exists m \leq n. (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta',m) \in \text{derivable R*}

\text{assume } r \in R3

\text{obtain } ps c \text{ where } r = (ps,c) \text{ by (cases r) auto}

\text{then have } r \in \text{provRules using rules and } \langle r \in R3 \rangle \text{ by auto}

\text{have } \text{rightPrincipal r } (F \triangledown [x],A) \land \sim \text{rightPrincipal r } (F \triangledown [x],A) \text{ by blast}

moreover

\text{assume } \text{rightPrincipal r } (F \triangledown [x],A)

\text{then have } (\Gamma' \Rightarrow^* \Delta') \in \text{set ps using } \langle r = (ps,c) \rangle \text{ and } \langle r \in R3 \rangle \text{ and rules by auto}

\text{then have } \text{extend S } (\Gamma' \Rightarrow^* \Delta') \in \text{set Ps using}

\langle \text{extendRule S r } = (Ps,\Gamma' \Rightarrow^* \Delta \oplus F \triangledown [x],A) \rangle

\text{and } \langle r = (ps,c) \rangle \text{ by (simp add:extendContain)}

moreover from (\text{rightPrincipal r } (F \triangledown [x],A)) \text{ have}

\text{c = } (\emptyset \Rightarrow^* (F \triangledown [x],A))

\text{using } \langle r = (ps,c) \rangle \text{ by (cases r) auto}

with \langle \text{extendRule S r } = (Ps,\Gamma' \Rightarrow^* \Delta \oplus F \triangledown [x],A) \rangle
\text{ have S } = (\Gamma \Rightarrow^* \Delta)

\text{using } \langle r = (ps,c) \rangle \text{ by (cases S) auto}

ultimately have \( (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta') \in \text{set Ps by (simp add:extend-def)}

then have \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta',m) \in \text{derivable R*}

\text{using } \forall p \in \text{set Ps. } \exists n \leq n'. (p,n) \in \text{derivable R*} \text{ by auto}

then have \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta',m) \in \text{derivable R*}

\text{using } \langle n = \text{Suc } n' \rangle \text{ by (simp)}

\}

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

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.

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 :: ('a,'b) rule set ⇒ bool
  modaliseMultiset :: ('a,'b) form multiset ⇒ ('a,'b) form multiset

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 \( ! \cdot \Gamma \), where \( ! \) is a modal operator and \( \Gamma \) 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:

\[
\text{inductive-set } \text{modRules2} \\text{where} \\
\qquad \left[p \neq \emptyset ; \text{mset } c = \emptyset \text{ Modal } M \text{ Ms } \right] \implies (ps, c) \in \text{modRules2}
\]

The second are the \textit{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 \texttt{p-e}, for “prime extend”, which takes a set of modal active parts and a pair of modal operators (say \(!\) and \(\bullet\)), and returns the set of active parts extended with \(\Gamma \Rightarrow \bullet \cdot \Delta\):

\[
\text{inductive-set } \text{p-e} :: \left(\langle a, b \rangle \right) \text{rule set} \Rightarrow \langle b \rangle \Rightarrow \langle a, b \rangle \text{ rule set}
\]

\text{for } R :: \langle a, b \rangle \text{ rule set and } M N :: \langle b \rangle \\
\text{where} \\
\quad \left[ (Ps, c) \in R ; R \subseteq \text{modRules2} \right] \implies \text{extendRule seq } (M \cdot \Gamma \Rightarrow \bullet \cdot \Delta) (Ps, c) \in \text{p-e R M N}

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

\textbf{overloading } \text{extendConc} \equiv \text{extendConc} \\
\textbf{begin} \\
definition \text{extendConc} :: \left(\langle a, b \rangle \right) \text{sequent} \Rightarrow \langle a, b \rangle \text{ rule} \Rightarrow \langle a, b \rangle \text{ rule} \\
\text{where} \text{extendConc S r} \equiv (\text{fst r, extend S (snd r)}) \\
\textbf{end}

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

\[
\text{inductive-set } \text{ext} :: \left(\langle a, b \rangle \right) \text{ rule set} \Rightarrow \langle a, b \rangle \text{ rule set} \Rightarrow \langle b \rangle \Rightarrow \langle a, b \rangle \text{ rule set}
\]

\text{for } R :: \langle a, b \rangle \text{ rule set and } M N :: \langle b \rangle \\
\text{where} \\
\quad \text{ax: } \left[ r \in R ; r \in \text{Ax} \right] \implies \text{extendRule seq } r \in \text{ext R R' M N}
\quad \text{| up: } \left[ r \in R ; r \in \text{upRules} \right] \implies \text{extendRule seq } r \in \text{ext R R' M N}
\quad \text{| mod1: } \left[ r \in \text{p-e R' M N} ; r \in R \right] \implies \text{extendConc seq } r \in \text{ext R R' M N}
\quad \text{| mod2: } \left[ r \in R ; r \in \text{modRules2} \right] \implies \text{extendRule seq } r \in \text{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.

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:

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

This is the case where the last inference was a normal modal inference:

{assume \(r \in \text{ modRules2}\)
obtain \(ps \ c \ \text{ where } r = (ps,c) \) by (cases \(r\) ) auto
with \(\langle r \in \text{ modRules2}\rangle\) obtain \(T Ts \ \text{ where } c = (\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset) \lor \langle c = (\emptyset \text{ Modal } T Ts \emptyset \Rightarrow^\ast \emptyset) \rangle\)
using modRule2Characterise[where \(Ps = ps\) and \(C = c\)] by auto
moreover
{assume \(c = (\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset)\)
then have \(bb: \text{rightPrincipal } r (\text{Modal } T Ts) R' \) using \(\langle r = (ps,c)\rangle\) and \(\langle r \in R\rangle\)
proof—
We need to know \(r \in R\) so that we can extend the active part
from \(\langle c = (\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset)\rangle\) and
\(\langle r = (ps,c)\rangle\) and
\(\langle r \in R\rangle\) and
\(\langle r \in \text{ modRules2}\rangle\)
have \(\langle ps,\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in R\) by auto
with rules have \(\langle ps, \emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in \text{ e-R } R2 \text{ } M1 \text{ } M2 \lor \langle ps, \emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in R3\) by auto
moreover
{assume \(\langle ps,\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in R3\)
then have \(\langle ps, \emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in R'\) using rules by auto
}
moreover
{assume \(\langle ps,\emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \in \text{ e-R } R2 \text{ } M1 \text{ } M2\)
In this case, we show that \(\Delta'\) and \(\Gamma'\) must be empty. The details are generally suppressed:
then obtain \(\Gamma' \Delta' r'\)
where aa: \(\langle ps, \emptyset \Rightarrow^\ast \emptyset \text{ Modal } T Ts \emptyset \rangle \) = extendRule \((M1 \cdot \Gamma' \Rightarrow^\ast M2 \cdot \Delta') r'\)
\(\land r' \in R2\) by auto
then have \(M1 \cdot \Gamma' = \emptyset\) and \(M2 \cdot \Delta' = \emptyset\)
by (auto simp add:modaliseMultiset-def)

The other interesting case is where the last inference was a modalised context inference:

{assume ba: \(r \in \text{ e-R } R2 \text{ } M1 \text{ } M2 \land \text{ extendConc } S r = (Ps, \Gamma \Rightarrow^\ast \emptyset \oplus \text{Modal } M \text{ Ms})\)
with rules obtain \(F Fs \Gamma'' \Delta'' ps r'\) where
ca: \( r = \text{extendRule} (M1 \cdot \Gamma' \Rightarrow * M2 \cdot \Delta' \) r' and
cb: \( r' \in R2 \) and
cc: \( r' = (ps, \emptyset \Rightarrow * (\text{Modal F Fs})) \) or \( r' = (ps, \text{Modal F Fs} \Rightarrow * \emptyset) \)

from \( ba \) and rules
have \( \text{extendConc}(\Gamma' + \Gamma' \Rightarrow * \Delta' + \Delta') \) \( r' \in (\text{ext R R2 M1 M2}) \) by auto
moreover from \( ba \) and \( ca \) have \( \text{fst}(\text{extendConc}(\Gamma' + \Gamma' \Rightarrow * \Delta' + \Delta')) \) = \( Ps \) by (auto simp add: extendConc-def)
ultimately have \( (\Gamma + \Gamma' \Rightarrow * \Delta + \Delta', n'+1) \in \text{derivable}(\text{ext R R2 M1 M2}) \) by auto
then have \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow * \Delta + \Delta', m) \in \text{derivable}(\text{ext R R2 M1 M2}) \) using \( \langle n = \text{Suc } n' \rangle \) by auto
ultimately have \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow * \Delta + \Delta', m) \in \text{derivable}(\text{ext R R2 M1 M2}) \) by blast

The other case, where the last inference was a left inference, is more straightforward, and so is omitted.

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:

\[
\begin{array}{ll}
\Gamma' \Rightarrow A, \Gamma \Rightarrow \Delta, \\
\Gamma \Rightarrow \Delta, \Delta' \Rightarrow A
\end{array}
\]

Since, if \( ([\emptyset \Rightarrow A], \emptyset \Rightarrow \bullet A) \in \mathcal{R} \), then \( R_1 \in \text{p-e } \mathcal{R} \) \( \bullet \), whereas \( R_2 \in \text{p-e } \mathcal{R} \) \( \bullet \). 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:

\[
\begin{array}{ll}
\Gamma' \Rightarrow A, \Gamma \Rightarrow \Delta, \\
\Gamma \Rightarrow \Delta, \Delta' \Rightarrow A
\end{array}
\]

cannot belong to any \( \text{p-e} \) set. It would be a simple matter to extend the definition of \( \text{p-e} \) 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 \( \Box \cdot \Gamma \Rightarrow \Diamond \cdot \Delta \):

\text{inductive-set } g3mod2
\text{where }
\begin{align*}
diaR: ([\emptyset \Rightarrow \Box A], \emptyset \Rightarrow * \Box A) & \in g3mod2 \\
boxL: ([\Box A \Rightarrow * \emptyset], \emptyset \Rightarrow \Box A) & \in g3mod2
\end{align*}

\text{inductive-set } g3mod1
\text{where }
\begin{align*}
boxR: ([\emptyset \Rightarrow \Box A], \emptyset \Rightarrow * \Box A) & \in g3mod1 \\
diaL: ([\Box A \Rightarrow * \emptyset], \emptyset \Rightarrow \Box A) & \in g3mod1
\end{align*}

We then show the strong admissibility of the rule:
\[
\Gamma \Rightarrow \Box A, \Delta
\]
\[
T' \Rightarrow A, \Delta
\]

**Lemma invertBoxR:**

- **Assumes** \( R = Ax \cup g_{3up} \cup (p-e g_{3mod1} \Box \Diamond) \cup g_{3mod2} \)
- **And** \( (\Gamma \Rightarrow \ast \Delta \oplus \Box A), n) \in \text{derivable} \ (\text{ext } R g_{3mod1} \Box \Diamond) \)
- **Shows** \( \exists m \leq n. (\Gamma \Rightarrow \ast \Delta \oplus A, m) \in \text{derivable} \ (\text{ext } R g_{3mod1} \Box \Diamond) \)

**Proof:**

- From assms show ?thesis
- Using principal and rightInvert and \( g_{3} \) by auto
- Qed

Where **principal** is the result which fulfils the principal formula conditions given in the inversion lemma, and \( g_{3} \) is a result about rule sets.

### 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 premises, a rule \( R \) with premises \( P \) will be redundant given a rule \( R' \) with premises \( P' \) 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 premises 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** \( r1 = (p\#ps, c) \land r1 \in \text{upRules} \)
- \( r2 = (ps, c) \land r2 \in \text{upRules} \)
- \( R1 \subseteq \text{upRules} \land R = Ax \cup R1 \)
- \( (T, n) \in \text{derivable} \ (R \cup \{r1\} \cup \{r2\})^* \)
- **Shows** \( \exists m \leq n. (T, m) \in \text{derivable} \ (R \cup \{r2\})^* \)

**Proof (induct n rule: nat-less-induct)**

**Case 0**

- \( (T, 0) \in \text{derivable} \ (R \cup \{r1\} \cup \{r2\})^* \) by simp
- then have \( ([], T) \in (R \cup \{r1\} \cup \{r2\})^* \) by (cases) auto
- then obtain \( S \ r \) where ext: extendRule \( S \ r = ([], T) \) and \( r \in (R \cup \{r1\} \cup \{r2\}) \) by (rule extRules_cases) auto
- then have \( r \in R \lor r = r1 \lor r = r2 \) using \( c \) by auto
It cannot be the case that \( r = r_1 \) or \( r = r_2 \), since those are uniprincipal rules, whereas anything with an empty set of premises must be an axiom. Since \( \mathcal{R} \) contains the set of axioms, so will \( \mathcal{R} \cup r_2 \):

\[
\begin{align*}
\text{then have } r & \in (\mathcal{R} \cup \{r_2\}) \text{ using } c \text{ by } auto \\
\text{then have } (T, 0) & \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \text{ by } auto \\
\text{then show } \exists m \leq n. (T, m) & \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \text{ using } \langle n = 0 \rangle \text{ by } auto \\
\text{next case } (\text{Suc } n') \\
\text{have } (T, n' + 1) & \in \text{derivable } (\mathcal{R} \cup \{r_1\} \cup \{r_2\})^* \text{ by } simp \\
\text{then obtain } Ps \text{ where } c: Ps \neq [] \\
\quad \text{and } f: (Ps, T) \in (\mathcal{R} \cup \{r_1\} \cup \{r_2\})^* \\
\quad \text{and } g: \forall P \in \text{set } Ps. \exists m \leq n'. (P, m) \in \text{derivable } (\mathcal{R} \cup \{r_1\} \cup \{r_2\})^* \\
\quad \text{by } auto \\
\text{have } g': \forall P \in \text{set } Ps. \exists m \leq n'. (P, m) \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \\
\text{from } f \text{ obtain } S r \text{ where } ext: \text{extendRule } S r = (Ps, T) \\
\quad \text{and } r \in (\mathcal{R} \cup \{r_1\} \cup \{r_2\}) \text{ by (rule } extRules\text{.cases } auto \\
\text{then have } r \in (\mathcal{R} \cup \{r_2\}) \lor r = r_1 \text{ by } auto \\
\end{align*}
\]

Either \( r \) is in the new rule set or \( r \) is the redundant rule. In the former case, there is nothing to do:

\[
\begin{align*}
\text{assume } r & \in (\mathcal{R} \cup \{r_2\}) \\
\text{then have } (Ps, T) & \in (\mathcal{R} \cup \{r_2\})^* \text{ by } auto \\
\text{with } g' \text{ have } (T, n) & \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \text{ using } n = \text{Suc } n' \text{ by } auto \\
\end{align*}
\]

In the latter case, the last inference was redundant. Therefore the premisses, which are derivable at a lower height than the conclusion, contain the premisses of \( r_2 \) (these premisses are \text{extend } S \text{ ps}). This completes the proof:

\[
\begin{align*}
\text{assume } r & = r_1 \\
\text{with } ext \text{ have } \text{map } (\text{extend } S) (p \neq ps) = Ps \text{ using } a \text{ by } (auto) \\
\text{then have } \forall P \in \text{set } (\text{map } (\text{extend } S) (p \neq ps)). \\
\exists m \leq n'. (P, m) \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \\
\quad \text{using } g' \text{ by } simp \\
\text{then have } h: \forall P \in \text{set } (\text{map } (\text{extend } S) ps). \\
\exists m \leq n'. (P, m) \in \text{derivable } (\mathcal{R} \cup \{r_2\})^* \text{ by } auto \\
\end{align*}
\]

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:

\textbf{lemma removeSuperfluous:}

\textbf{assumes} \( r_1 = ((\emptyset \Rightarrow \{A\}) \neq ((\{A\} \Rightarrow \emptyset) \neq ps), c) \land r_1 \in \text{upRules} \)
\textbf{and} \( R_1 \subseteq \text{upRules} \land R = Ax \cup R_1 \)
\textbf{and} \( (T, n) \in \text{derivable } (\mathcal{R} \cup \{r_1\})^* \)
\textbf{and} \( CA: \forall \Gamma \Delta A. ((\Gamma \Rightarrow \Delta \oplus A) \in \text{derivable'} R^* \rightarrow \to \)
\[(\Gamma \oplus A \Rightarrow \Delta) \in \text{derivable}' R^* \Rightarrow (\Gamma \Rightarrow \Delta) \in \text{derivable}' R^*\]

shows \(T \in \text{derivable}' R^*\)

Combining 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 \textit{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: \(r1 = (p \# ps, c) \land r1 \in \text{upRules}\)
and b: \(r2 = (q \# ps, c) \land r2 \in \text{upRules}\)
and c: \(r3 = (\text{extend } p q \# ps, c) \land r3 \in \text{upRules}\)
and d: \(R1 \subseteq \text{upRules} \land R = Ax \cup R1\)
and \((\text{T}, n) \in \text{derivable} (R \cup \{r1\} \cup \{r2\})^*\)

**shows** \((\text{T}, n) \in \text{derivable} (R \cup \{r3\})^*\)

### 8 Conclusions

Only a portion of the formalisation was shown; a variety of intermediate lemmata were not made explicit. This was necessary, for the \textit{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 SRCTrans- forms

**begin**

**end**

### References