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:

```haskell
datatype 'a form = At nat | Compound 'a 'a form list | ff
```

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

```haskell
datatype Gp = con | dis | imp
type-synonym Gp-form = Gp form
```
Abbreviation **con-form** (infixl \(\land\ast\)) where
\(p \land\ast q \equiv \text{Compound con} [p,q]\)

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

Abbreviation **imp-form** (infixl \(\supset\)) 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 \(+\), 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
- \(id: ([], \{ At i \}) \Rightarrow^\ast [\{ At i \}] \in Ax\)
- \(Lbot: ([], \{ \text{ff} \}) \Rightarrow^\ast \emptyset \in Ax\)

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
- \(I: \{ \text{mset c} \equiv \{ \text{Compound R Fs} \} \mid ps \neq [] \} \Rightarrow (ps,c) \in upRules\)

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**
- \(conL: ([], A \uplus B \Rightarrow^\ast \emptyset), \{ A \land\ast B \} \Rightarrow^\ast \emptyset \in g3cp\)
- \(conR: ([\emptyset \Rightarrow^\ast \{ A \}), \{ B \} \Rightarrow^\ast \{ A \land\ast B \}) \in g3cp\)
- \(disL: ([\emptyset \Rightarrow^\ast \{ A \}), \{ B \} \Rightarrow^\ast \emptyset), \{ A \lor\ast B \} \Rightarrow^\ast \emptyset \in g3cp\)
- \(disR: ([\emptyset \Rightarrow^\ast \{ A \}), \{ B \} \Rightarrow^\ast \emptyset), \{ A \lor\ast B \} \Rightarrow^\ast \emptyset \in g3cp\)
- \(impL: ([\emptyset \Rightarrow^\ast \{ A \}), \{ B \} \Rightarrow^\ast \emptyset), \{ A \supset B \} \Rightarrow^\ast \emptyset \in g3cp\)
- \(impR: ([\emptyset \Rightarrow^\ast \{ A \}), \{ B \} \Rightarrow^\ast \emptyset), \{ A \supset B \} \Rightarrow^\ast \emptyset \in g3cp\)

**Lemma** \(g3cp-upRules\):

**show** \(g3cp \subseteq upRules\)

**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**
- \( \text{extend} \equiv \text{extend} \)
- \( \text{extendRule} \equiv \text{extendRule} \)

**begin**

**definition** \( \text{extend} \)
- where \( \text{extend forms seq} \equiv (\text{antec forms} + \text{antec seq}) \Rightarrow^* (\text{succ forms} + \text{succ seq}) \)

**definition** \( \text{extendRule} \)
- where \( \text{extendRule forms} R \equiv (\text{map (extend forms) (fst R), extend forms (snd R)}) \)

**end**

Given a rule set \( R \), the extension of \( R \), called \( R^* \), is then defined as another inductive set:

**inductive-set** \( \text{extRules} :: \text{a rule set} \Rightarrow \text{a rule set} (\Rightarrow) \)
- for \( R :: \text{a rule set} \)
- where \( I: r \in R \implies \text{extendRule seq} r \in R^* \)

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

**overloading** \( \text{uniqueConclusion} \equiv \text{uniqueConclusion} \)

**begin**

**definition** \( \text{uniqueConclusion} :: \text{a rule set} \Rightarrow \text{bool} \)
- where \( \text{uniqueConclusion} R \equiv \forall r1 \in R. \forall r2 \in R. (\text{snd r1} = \text{snd r2}) \rightarrow (r1 = r2) \)

**end**

**lemma** \( \text{g3cp-uc} \):
- shows \( \text{uniqueConclusion g3cp} \)
- 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:

**inductive** rightPrincipal $::$ 'a rule $\Rightarrow$ 'a form $\Rightarrow$ bool

where

$up :: C = (\emptyset \Rightarrow \ast (\text{Compound } F \text{ Fs})))$ $\implies$

rightPrincipal $(Ps,C)$ $(\text{Compound } F \text{ Fs})$

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 $a ::$ rightPrincipal $r$ $(A \land \ast B)$

and $b ::$ $r \in g3cp$

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

proof

- from $a$ and $b$ obtain $Ps$ where req: $r = (Ps, \emptyset \Rightarrow \ast (\text{Compound } A \land \ast B \text{ Fs})))$
  
  by $(\text{cases } r)$ auto

  with $b$ have $Ps = [\emptyset \Rightarrow \ast (\text{Compound } A \text{ Fs})), \emptyset \Rightarrow \ast (\text{Compound } B \text{ Fs}))))$
  
  apply $(\text{cases } r)$ by $(\text{rule } g3cp.cases)$ auto

  with req show $(\emptyset \Rightarrow \ast (\text{Compound } A \text{ Fs}))) \in \text{set (fst } r)$ 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 derivable $R$:

**inductive-set** derivable $::$ 'a rule set $\Rightarrow$ 'a deriv set

for $R ::$ 'a rule set

where

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

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

$\implies (snd r,m + 1) \in \text{derivable } R$

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 derivable', which is a set of sequents:

**inductive-set** derivable' $::$ 'a rule set $\Rightarrow$ 'a sequent set

for $R ::$ 'a rule set

where

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

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

$\implies (snd r) \in \text{derivable'} R$

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 \(\forall \; p \in \text{set } as. \exists \; n \leq m. \; (p,n) \in \text{derivable } R\) by auto

moreover from \(\forall \; p \in \text{set } (a \# as). \exists \; n. \; (p,n) \in \text{derivable } R\) 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 \(\forall r \in R. \; \text{have } (c,m+1) \in \text{derivable } R\) 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.
lemma rightInvertible:
fixes Γ Δ :: 'a form multiset
assumes rules: R' ⊆ upRules ∧ R = Ax ∪ R'
  and a: (Γ ⇒∗ Δ ⊕ Compound F Fs,n) ∈ derivable R*
  and b: ∀ r' ∈ R. rightPrincipal r' (Compound F Fs) →
        (Γ' ⇒∗ Δ') ∈ set (fst r')
shows ∃ m≤n. (Γ + Γ' ⇒∗ Δ + Δ',m) ∈ derivable R*
using assms

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

proof (induct n arbitrary; Γ Δ rule:nat-less-induct)
case (1 n Γ Δ)
  then have IH:∀ m<n. ∀ Γ Δ. ( Γ ⇒∗ Δ ⊕ Compound F Fs, m) ∈ derivable R* →
        (∀ r' ∈ R. rightPrincipal r' (Compound F Fs) →
        (Γ' ⇒∗ Δ') ∈ set (fst r')) →
        (∃ m≤m. (Γ + Γ' ⇒∗ Δ + Δ', m') ∈ derivable R*)
  and a': (Γ ⇒∗ Δ ⊕ Compound F Fs,n) ∈ derivable R*
  and b': ∀ r' ∈ R. rightPrincipal r' (Compound F Fs) →
        (Γ' ⇒∗ Δ') ∈ set (fst r')
  
by auto
show ?case
proof (cases n)  — Case analysis on n
  case 0
    then obtain r S where extendRule S r = ([],Γ ⇒∗ Δ ⊕ Compound F Fs)
    and r ∈ Ax ∨ r ∈ R' by auto  — At height 0, the premisses are empty
    moreover
      {assume r ∈ Ax
       then obtain i where ([], At i ⊢ Δ) ⇒∗ ([], At i ⊢ Δ) = r ∨
       r = ([], If i ⊢ Δ) ⇒∗ 0)
       using characteriseAx[where r=r] by auto
      }  — Case split on the kind of axiom used
      moreover
        {assume r = ([], At i ⊢ Δ)
         then have At i ∈ # Γ ∧ At i ∈ # Δ by auto
         then have At i ∈ # Γ + Γ' ∧ At i ∈ # Δ + Δ' by auto
         then have (Γ + Γ' ⇒∗ Δ + Δ',0) ∈ derivable R* using rules by auto
        }
      moreover
        {assume r = ([],If i ⊢ Δ) ⇒∗ 0)
         then have If ∈ # Γ by auto
         then have If ∈ # Γ + Γ' by auto
         then have (Γ + Γ' ⇒∗ Δ + Δ',0) ∈ derivable R* using rules by auto
        }
      ultimately have (Γ + Γ' ⇒∗ Δ + Δ',0) ∈ derivable R* by blast
    }
  moreover
    {assume r ∈ R'  — This leads to a contradiction
     then obtain Ps C where Ps ≠ [] and r = (Ps,C) by auto
     moreover obtain S where r = ([],S) by blast  — Contradiction
     ultimately have (Γ + Γ' ⇒∗ Δ + Δ',0) ∈ derivable R* using rules by simp
    }
}
Then, we apply the induction hypothesis at the lower height of the premisses:

\(2\) lines give a characterisation of the extension and conclusion, respectively.

If the formula is not principal, then it must appear in the premisses. The first

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

then have \((\Gamma \Rightarrow \Delta \oplus \text{Compound } F \text{Fs}, n'+1) \in \text{derivable } R^* \text{ using } a' \text{ by simp}

then obtain \(Ps\) where \((Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \text{Fs}) \in R^* \text{ and}

\(Ps \neq [] \text{ 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 A x \lor r \in R'

and extendRule \(S r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \text{Fs}) \text{ by auto}

moreover

\{assume \(r \in A x \quad \text{— Gives a contradiction}\)

then have \(\text{fst } r = [] \text{ apply } (cases r) \text{ by (rule A.x.cases)} \text{ auto}

moreover obtain \(x y \text{ where } r = (x,y) \text{ by (cases r)}

then have \(x \neq [] \text{ using } (Ps \neq [])

and (extendRule \(S r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F \text{Fs}) \text{ by auto)

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

moreover

\{assume \(r \in R'

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

have \((\text{rightPrincipal } r \text{ (Compound } F \text{Fs)})) \lor

\neg((\text{rightPrincipal } r \text{ (Compound } F \text{Fs)}))

by blast — The formula is principal, or not

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

\{assume \(\text{rightPrincipal } r \text{ (Compound } F \text{Fs)}

then have \((\Gamma' \Rightarrow \Delta') \in \text{set } ps \text{ using } b' \text{ by auto}

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

using (extendRule \(S r = (Ps,\Gamma \Rightarrow \Delta \oplus \text{Compound } F \text{Fs})\)

by (simp)

moreover have \(S = (\Gamma \Rightarrow \Delta) \text{ by (cases } S \text{ auto)

ultimately have \((\Gamma + \Gamma' \Rightarrow \Delta + \Delta') \in \text{set } Ps \text{ 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^* \text{ by auto)

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

\{assume \(\neg \text{rightPrincipal } r \text{ (Compound } F \text{Fs)}

obtain \(\Phi \Psi \text{ where } S = (\Phi \Rightarrow \Psi) \text{ by (cases } S \text{) (auto)

then obtain \(G H \text{ where } c = (G \Rightarrow H) \text{ by (cases } c \text{) (auto)

then have \(\notin \text{ Compound } F \text{Fs} \notin \text{ H} \quad \text{— Proof omitted

have \(\Psi + H = \Delta \oplus \text{Compound } F \text{Fs} \)}
using \( S = (\Phi \Rightarrow \Psi) \) and \( r = (ps,c) \) and \( c = (G \Rightarrow H) \) by auto
moreover from \( r = (ps,c) \) and \( c = (G \Rightarrow H) \)
have \( H = \emptyset \lor (\exists \ A. \ H = \{ A \}) \) by auto
ultimately have \( \text{Compound } Fs \in \# \Psi \) — Proof omitted
then have \( \exists \Psi1. \ \Psi = \Psi1 \oplus \text{Compound } Fs \) by (auto)
then obtain \( \Psi1 \) where \( S = (\Phi \Rightarrow \Psi1 \oplus \text{Compound } Fs) \) by auto
have \( \forall \ p \in \text{set } Ps. (\text{Compound } Fs \in \# p) \) — Appears in every premiss
by (auto)
then have \( \forall \ p \in \text{set } Ps. \exists \Phi' \Psi' \ m. \ m \leq n' \land \)
\((\Phi' + \Gamma' \Rightarrow \Psi' + \Delta',m) \in \text{derivable } \text{R*} \land \)
\( p = (\Phi' \Rightarrow \Psi' \oplus \text{Compound } Fs) \) using \text{IH} by (arith)

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

obtain \( Ps' \) where eq: \( Ps' = \text{map} (\text{extend} (\Phi + \Gamma' \Rightarrow \Psi1 + \Delta')) ps \) by auto
have \((Ps',\Gamma + \Gamma' \Rightarrow \Psi1 + \Delta') \in \text{R*} \) by simp
then have \( \forall \ p \in \text{set } Ps'. \exists \ n \leq n'. (p,n) \in \text{derivable } \text{R*} \) by auto
then have \( \exists \ m \leq n. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta',m) \in \text{derivable } \text{R*} \)
using \((Ps',\Gamma + \Gamma' \Rightarrow \Delta + \Delta') \in \text{R*} \) by (auto)

All of the cases are now complete.

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

As an example, we show the left premiss of \( \text{R*} \) in \text{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 \((\Gamma \Rightarrow \Delta \oplus (A \land \bot \cdot B)) \cdot n \) in \text{derivable} \((g3cp \cup \text{Ax})\)
shows \( \exists \ m \leq n. (\Gamma \Rightarrow \Delta \oplus A \cdot m) \in \text{derivable} \((g3cp \cup \text{Ax})\)
proof-
have \( \forall \ r \in g3cp. \rightPrincipal r (A \land \bot \cdot B) \rightarrow (\emptyset \Rightarrow \bot \land A \land B) \) in \( \text{set} (\text{fst } r) \)
using principal-means-premiss by auto
with assms show \( ?\text{thesis} \) using rightInvertible by (auto)

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

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

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. \textbf{G3cp} has the unique conclusion property (as shown in §2.2). Thus, \textbf{G3cp} is an invertible set of rules:

\textbf{lemma unique-to-invertible:}
\textbf{assumes} R′ ⊆ upRules ∧ R = Ax ∪ R′
\textbf{and} uniqueConclusion R′
\textbf{shows} invertible-set R

\textbf{lemma g3cp-invertible:}
\textbf{shows} invertible-set \textbf{(Ax ∪ g3cp)}
\textbf{using} g3cp-uc and g3cp-upRules
  \textbf{and} unique-to-invertible[where R′=g3cp and R=Ax ∪ g3cp]
\textbf{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 \textbf{G3ip}? The left premiss will be \( A ⊃ B \Rightarrow A \), and the extension will be \( \Gamma \Rightarrow C \). The \texttt{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 \( E_0 \), 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 $(\Gamma \Rightarrow∗ Em, n) \in$ derivable $R^*$
  and $R' \subseteq upRules$
  and $R = Ax \cup R'$
  shows $(\Gamma \Rightarrow∗ C, n) \in$ 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 $R' \subseteq upRules \land R = Ax \cup R'$
  and $(\Gamma \Rightarrow∗ Compound F Fs, n) \in$ derivable $R^*$
  and $\forall r' \in R. \rightPrincipal r' (Compound F Fs) \longrightarrow (\Gamma' \Rightarrow∗ E) \in \set (fst r')$
  and $E \neq Em$
  shows $\exists m \leq n. (\Gamma + \Gamma' \Rightarrow∗ E, m) \in$ derivable $R^*$

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

G3ip can be expressed in this formalism:
As expected, $R \supset$ can be shown invertible:

**lemma** `impRInvert`:
**assumes** $(\Gamma \Rightarrow B, n) \in \text{derivable}(Ax \cup g3ip)$
**shows** $\exists m \leq n. (\Gamma \oplus A \Rightarrow B, m) \in \text{derivable}(Ax \cup g3ip)$
**proof**—
have $\forall r \in (Ax \cup g3ip). \text{rightPrincipal}(r, (A \supset B) \longrightarrow (B \Rightarrow B) \in \text{set}(\text{fst } r)$
**proof**— — Showing that $A \Rightarrow B$ is a premiss of every rule with $A \supset B$ principal
{fix $r$
assume $r \in (Ax \cup g3ip)$
moreover assume $\text{rightPrincipal}(r, (A \supset B)$
ultimately have $r \in g3ip$ by auto — If $A \supset B$ was principal, then $r \notin Ax$
from $(\text{rightPrincipal}(r, (A \supset B))$ have $\text{snd } r = (\emptyset \Rightarrow (A \supset B))$ by auto
with $r \in g3ip$ and $(\text{rightPrincipal}(r, (A \supset B)$
then have $(\emptyset \Rightarrow B) \in \text{set}(\text{fst } r)$ by auto
}
thus `?thesis` by auto
{qed
with `assms` show `?thesis` using `rightInvertible` by auto
qed

## 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 (- (\[\].-)
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:

\[\text{lemma} \ \text{nprovContain}: \ \text{shows} \ \text{provRules} \subseteq \text{nprovRules} \ \text{proof} - \ 
\{ \text{fix } ps \ c \ \text{assume} (ps, c) \in \text{provRules} \ \text{then have} (ps, c) \in \text{nprovRules} \text{ by (cases) auto} \} \ 
\text{then show } \exists \text{thesis by auto} \ 
\text{qed} \]

Substitution is defined in the usual way:

\[\text{nominal-primrec} \ 
\text{subst-form} :: \text{var} \Rightarrow \text{var} \Rightarrow \text{form} \Rightarrow \text{form} ([\cdot,\cdot]) \ 
\text{and subst-forms} :: \text{var} \Rightarrow \text{var} \Rightarrow \text{form-list} \Rightarrow \text{form-list} ([\cdot,\cdot]) \ 
\text{where} \ 
\begin{align*}
[z, y](\text{At } P \ xs) &= \text{At } P ([z, y]xs) \\
[x \y](z, y) &\Rightarrow [z, y](F \ \nabla [x].A) = F \ \nabla [x].([z, y]A) \\
[z, y](\text{Cpd0 } F \ Fs) &= \text{Cpd0 } F ([z, y]Fs) \\
[z, y]ff &= ff \\
[z, y]FNil &= FNil \\
[z, y](\text{FCons } f \ Fs) &= \text{FCons } ([z, y]f) ([z, y]Fs)
\end{align*} \]

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**  \( \text{multSubst} :: \text{form mset} \Rightarrow \text{bool} \) where

\[ \text{multSubst-def} : \text{multSubst} \Gamma \equiv (\exists A \in (\text{set-mset} \Gamma). \exists x y B. [y,x]B = A \land y \neq x) \]

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**  \(\text{extRules} :: \text{rule set} \Rightarrow \text{rule set} \ (\to\to)\)

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

where

| id: \( \left[ \begin{array}{l} r \in R ; r \in Ax \end{array} \right] \implies \text{extendRule S} r \in R^* \) |
| sc: \( \left[ \begin{array}{l} r \in R ; r \in \text{upRules} \end{array} \right] \implies \text{extendRule S} r \in R^* \) |
| np: \( \left[ \begin{array}{l} r \in R ; r \in \text{nprovRules} \end{array} \right] \implies \text{extendRule S} r \in R^* \) |
| p: \( \left[ \begin{array}{l} (ps,c) \in R ; (ps,c) \in \text{provRules} ; \text{mset c} = \{ F \nabla [x].A \} ; x \n\n set-of-seq S \end{array} \right] \implies \text{extendRule S} (ps,c) \in R^* \) |

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**  \(\text{formSubst:}\)

shows \(y \n\n x \land y \n\n A \implies F \nabla [x].A = F \nabla [y].([y,x]A)\)

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:

**lemma**  \(\text{extend-for-any-seq:}\)

fixes \(S :: \text{sequent}\)

assumes rules: \(R_1 \subseteq \text{upRules} \land R_2 \subseteq \text{nprovRules} \land R_3 \subseteq \text{provRules}\)

and rules2: \(R = Ax \cup R_1 \cup R_2 \cup R_3\)

and rin: \(r \in R\)

shows \(\text{extendRule S} r \in R^*\)

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

assume \(r \in R_3\)

then have \(r \in \text{provRules}\) using rules by auto

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

then have r1: \((ps,c) \in R\)

and r2: \((ps,c) \in \text{provRules}\) using \(r \in \text{provRules}\) and rin by auto

with \((r = (ps,c))\) obtain \(F \ x\ A\)

where \(c = (\emptyset \Rightarrow \top) \Rightarrow F \nabla [x].A) \vee\)

\(c = (\{ F \nabla [x].A \Rightarrow \emptyset \}) \land x \n\n \text{set-of-prem} (ps - A)\)

using provRuleCharacterise and \(r \in \text{provRules}\) by auto
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: \( R1 \subseteq \upRules \land R2 \subseteq \nprovRules \land R3 \subseteq provRules \land R = Ax \cup R1 \cup R2 \cup R3 \)

and a: \( (\Gamma \Rightarrow \ast \Delta \cup F \setminus [x].A, n) \in \text{derivable } R \)

and b: \( \forall r' \in R. \left( \text{rightPrincipal } r' \left( F \setminus [x].A \right) \Rightarrow (\Gamma' \Rightarrow \ast \Delta') \in \text{set } (\text{fst } r') \right) \)

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

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

assume \( r \in R3 \)

**obtain** \( ps c \) where \( r = (ps, c) \) by (cases \( r \)) auto

then have \( r \in \text{provRules} \) using rules and \( (r \in R3) \) by auto

have rightPrincipal \( r \left( F \setminus [x].A \right) \lor \neg \text{rightPrincipal } r \left( F \setminus [x].A \right) \) by blast

moreover

\( \text{(assume rightPrincipal } r \left( F \setminus [x].A \right) \) then have \( (\Gamma' \Rightarrow \ast \Delta') \in \text{set } ps \) using \( (r = (ps, c)) \) and \( (r \in R3) \) and rules by auto

then have extend \( S \left( \Gamma' \Rightarrow \ast \Delta' \right) \) \( \in \text{set } Ps \) using extendRule \( S r = (Ps, \Gamma \Rightarrow \ast \Delta \cup F \setminus [x].A) \) have \( S = (\Gamma \Rightarrow \ast \Delta) \)

using \( (r = (ps, c)) \) by (cases \( S \)) auto

moreover from \( \text{rightPrincipal } r \left( F \setminus [x].A \right) \) have \( c = (\emptyset \Rightarrow \ast \left( F \setminus [x].A \right)) \)

using \( (r = (ps, c)) \) by (cases \( c \)) auto

with extendRule \( S r = (Ps, \Gamma \Rightarrow \ast \Delta \cup F \setminus [x].A) \) have \( S = (\Gamma \Rightarrow \ast \Delta) \)

using \( (r = (ps, c)) \) by (cases \( S \)) auto

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

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

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

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

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

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

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:

```isar
datatype ('a, 'b) form =  
  At nat 
  | Compound ('a, 'b) form list 
  | Modal 'b ('a, 'b) form list 
  | ff
```

```isar
data datatype-compat form

overloading
uniqueConclusion \equiv uniqueConclusion
modaliseMultiset \equiv modaliseMultiset
begin

definition uniqueConclusion :: ('a,'b) rule set \Rightarrow bool
where uniqueConclusion R \equiv \forall r1 \in R. \forall r2 \in R. (\text{snd } r1 = \text{snd } r2) \rightarrow (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:

```isar
definition modaliseMultiset :: 'b \Rightarrow ('a,'b) form multiset \Rightarrow ('a,'b) form multiset
where modaliseMultiset a \Gamma \equiv \{ # Modal a [p], p \in # \Gamma # \}
```

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

\[
\text{inductive-set } \text{modRules2 where }
\begin{align*}
\[] & \; ps \neq [] ; \text{mset} c = \{ \text{Modal } M \text{ Ms } \} \implies (ps, c) \in \text{modRules2}
\end{align*}
\]

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 ! ∙ \( \Gamma \) ⇒ • ∙ \( \Delta \):

\[
\text{inductive-set } p-e :: (′a, ′b) \text{ rule set } ⇒ ′b ⇒ ′b ⇒ (′a, ′b) \text{ rule set for } R :: (′a, ′b) \text{ rule set and } M N :: ′b
\]

where

\[
\[ (Ps, c) \in R \; R \subseteq \text{modRules2 } \implies \text{extendRule seq r } \in \text{ext } R R′ M N\]
\]

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

\[
\text{overloading } \text{extendConc } \equiv \text{extendConc}
\]

begin

\[
\text{definition } \text{extendConc } :: (′a, ′b) \text{ sequent } ⇒ (′a, ′b) \text{ rule for } R :: (′a, ′b) \text{ rule and } M N :: ′b
\]

where

\[
\text{extendConc S r } \equiv (\text{fst r, extend S (snd r))}
\]

end

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

\[
\text{inductive-set } \text{ext } :: (′a, ′b) \text{ rule set } ⇒ (′a, ′b) \text{ rule set } ⇒ ′b ⇒ ′b ⇒ (′a, ′b) \text{ rule set for } R R′ :: (′a, ′b) \text{ rule set and } M N :: ′b
\]

where

\[
\text{ax: } [ \; r \in R \; r \in A x ] \implies \text{extendRule seq r } \in \text{ext } R R′ M N
\]

\[
\text{up: } [ \; r \in R \; r \in upRules ] \implies \text{extendRule seq r } \in \text{ext } R R′ M N
\]

\[
\text{mod1: } [ \; r \in p-e R′ M N \; r \in R ] \implies \text{extendConc seq r } \in \text{ext } R R′ M N
\]

\[
\text{mod2: } [ \; r \in R \; r \in modRules2 ] \implies \text{extendRule seq r } \in \text{ext } R R′ M N
\]

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:

\[
\text{lemma } \text{rightInvert:}
\]

\[
\text{fixes } Γ ∆ :: (′a, ′b) \text{ form multiset}
\]

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

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

\{assume \( r \in \text{modRules2} \)
  obtain \( ps \ c \) where \( r = (ps,c) \) by (cases \( r \)) auto
with \( (r \in \text{modRules2}) \) obtain \( T Ts \) where \( c = (\emptyset \Rightarrow* \text{ Modal } T Ts ) \) \( \vee \)
  \( c = (\text{ Modal } T Ts \Rightarrow* \emptyset) \)
  using \( \text{modRule2Characterise} \) [where \( Ps = ps \) and \( C = c \)] by auto
moreover
\{assume \( c = (\emptyset \Rightarrow* \text{ Modal } T Ts ) \)
then have \( bb: \text{rightPrincipal } r \) (\text{Modal } T Ts) \( R' \) using \( (r = (ps,c)) \) and \( (r \in R) \)
proof--

We need to know \( r \in R \) so that we can extend the active part

\[
\langle c = (\emptyset \Rightarrow* \text{ Modal } T Ts )\rangle \quad \text{and} \quad
\langle r \in R \rangle \quad \text{and} \quad
\langle r \in \text{modRules2} \rangle \quad \text{and} \quad
\langle ps,\emptyset \Rightarrow* \text{ Modal } T Ts \rangle \in R \text{ by auto} \]
with rules have \( (ps, \emptyset \Rightarrow* \text{ Modal } T Ts ) \in p-e R2 M1 M2 \) \( \vee \)
(\( ps, \emptyset \Rightarrow* \text{ Modal } T Ts \) \in R3 ) by auto
moreover
\{assume \( (ps,\emptyset \Rightarrow* \text{ Modal } T Ts ) \in R3 \)
then have \( (ps,\emptyset \Rightarrow* \text{ Modal } T Ts ) \in R' \) using rules by auto \}
moreover
\{assume \( (ps,\emptyset \Rightarrow* \text{ Modal } T Ts ) \in p-e R2 M1 M2 \)
In this case, we show that \( \Delta' \) and \( \Gamma' \) must be empty. The details are generally suppressed:

then obtain \( \Gamma' \Delta' r' \)
where \( aa: (ps,\emptyset \Rightarrow* \text{ Modal } T Ts ) = \text{extendRule } (M1\Gamma' \Rightarrow* M2\Delta') \quad r' \)
\wedge \( r' \in R2 \text{ by auto} \)
then have \( M1:\Gamma' = \emptyset \quad \text{and} \quad M2:\Delta' = \emptyset \)
by (auto simp add:modaliseMultiset-def)

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

\{assume \( ba: r \in p-e R2 M1 M2 \wedge \)
\( \text{extendCone } S \ r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Modal } M Ms) \)
with rules obtain \( F Fs \Gamma'' \Delta'' ps \ r' \) where
\[ r = \text{extendRule} (M_1 \cdot \Gamma'' \Rightarrow^* M_2 \cdot \Delta'') \ r' \text{ and } \]
\[ r' \in R_2 \text{ and } \]
\[ r' = (ps, \emptyset \Rightarrow^* \text{Modal F Fs}) \text{ and } r' = (ps, \text{Modal F Fs} \Rightarrow^* \emptyset) \]

from ba and rules
have extendConc \( (\Gamma_1 + \Gamma' \Rightarrow^* \Delta_2 + \Delta') \ r \in (\text{ext } R R_2 M_1 M_2) \) by auto
moreover from ba and ca have fst (extendConc \( (\Gamma_1 + \Gamma' \Rightarrow^* \Delta_2 + \Delta') \ r \)) = Ps
by (auto simp add: extendConc-def)
ultimately have \( (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', n' + 1) \) ∈ derivable \( (ext \ R R_2 M_1 M_2) \)
by auto
then have \( \exists \ m \leq n. \ (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', m) \) ∈ derivable \( (ext \ R R_2 M_1 M_2) \)
using \( (n = \text{Suc } n') \) by auto
\}
ultimately have \( \exists \ m \leq n. \ (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', m) \) ∈ derivable \( (ext \ R R_2 M_1 M_2) \)
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:

\[
! \cdot \Gamma \Rightarrow A, \bullet \cdot \Delta \\
\Gamma', ! \cdot \Gamma \Rightarrow !A, \bullet \cdot \Delta, \Delta' \quad R_1 \\
\bullet \cdot \Gamma \Rightarrow !A, \bullet \cdot \Delta, \Delta' \\
\Gamma', \bullet \cdot \Gamma \Rightarrow !A, \bullet \cdot \Delta, \Delta' \quad R_2
\]

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

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:

\[
! \cdot \Gamma_1, \bullet \cdot \Gamma_2 \Rightarrow !A, \bullet \cdot \Delta_1, \bullet \cdot \Delta_2 \\
\Gamma', ! \cdot \Gamma_1, \bullet \cdot \Gamma_2 \Rightarrow !A, \bullet \cdot \Delta_1, \bullet \cdot \Delta_2, \Delta'
\]

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 \( \Box \cdot \Gamma \Rightarrow \Diamond \cdot \Delta \):

**inductive-set g3mod2**

where

\[
diaR: ([\emptyset \Rightarrow \Box A \emptyset], \emptyset \Rightarrow \Box A \emptyset) \in g3mod2 \\
boxL: ([\Box A \emptyset \Rightarrow \emptyset], \Box A \emptyset \Rightarrow \emptyset) \in g3mod2
\]

**inductive-set g3mod1**

where

\[
boxR: ([\emptyset \Rightarrow \Box A \emptyset], \emptyset \Rightarrow \Box A \emptyset) \in g3mod1 \\
diaL: ([\Box A \emptyset \Rightarrow \emptyset], \Box A \emptyset \Rightarrow \emptyset) \in g3mod1
\]

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

**Lemma invertBoxR:**

**Assumes** \( R = Ax \cup g3up \cup (p-e \ g3mod1 \ \Box \ \Diamond) \cup g3mod2 \)

and \( (\Gamma \Rightarrow^* \Delta \oplus (\Box A), n) \in \text{derivable} \ (\text{ext } R \ g3mod1 \ \Box \ \Diamond) \)

**Shows** \( \exists \ m \leq n. (\Gamma \Rightarrow^* \Delta \oplus A, m) \in \text{derivable} \ (\text{ext } R \ g3mod1 \ \Box \ \Diamond) \)

**Proof**

- from assms show ?thesis using principal and rightInvert and g3 by auto

**Qed**

where *principal* is the result which fulfills the principal formula conditions given in the inversion lemma, and *g3* 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 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 \( P = Q \oplus P' \), or if \( P = P' \oplus Q \), 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** \( r1 = (p\#ps,c) \wedge r1 \in \text{upRules} \)

and \( r2 = (ps,c) \wedge r2 \in \text{upRules} \)

and \( R1 \subseteq \text{upRules} \wedge R = Ax \cup R1 \)

and \( (T,n) \in \text{derivable} \ (R \cup \{r1\} \cup \{r2\})^* \)

**Shows** \( \exists \ m \leq n. (T,m) \in \text{derivable} \ (R \cup \{r2\})^* \)

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

**Case 0**

- have \( (T,0) \in \text{derivable} \ (R \cup \{r1\} \cup \{r2\})^* \) by simp

then have \( ([],T) \in (R \cup \{r1\} \cup \{r2\})^* \) by (cases) auto

then obtain \( S \) \( r \) where \( \text{ext: extendRule } S \ r = ([],T) \) and

\( r \in (R \cup \{r1\} \cup \{r2\}) \) by (rule extRules.cases) auto

then have \( r \in R \vee r = r1 \vee r = r2 \) 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 premisses must be an axiom. Since \( \mathcal{R} \) contains the set of axioms, so will \( \mathcal{R} \cup r_2 \):

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

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

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

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 \: ps). This completes the proof:

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

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:

\[
\text{lemma } \text{removeSuperfluous:} \\
\text{assumes } & \: r_1 = ((\emptyset \Rightarrow \star A) \# ((A \Rightarrow \emptyset) \# ps),c) \land r_1 \in \text{upRules} \\
\text{and } & \: R1 \subseteq \text{upRules} \land R = Ax \cup R1 \\
\text{and } & \: (T,n) \in \text{derivable } (\mathcal{R} \cup \{r_1\})^* \\
\text{and } & \: CA: \: \forall \: \Gamma \Delta A. \: ((\Gamma \Rightarrow \star \Delta \oplus A) \in \text{derivable} \rightarrow R^*)
\]

\[(\Gamma \oplus A \Rightarrow^* \Delta) \in \text{derivable } R^* \] 
\[(\Gamma \Rightarrow^* \Delta) \in \text{derivable } R^* \]

shows \( T \in \text{derivable } R^* \)

Combineable 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 \text{extend} function is commutative, the order of \(p\) and \(q\) in the new rule is not important. This transformation is height preserving:

\text{lemma removeCombinable:}
\begin{align*}
\text{assumes} & \ a: r1 = (p \# ps, c) \wedge r1 \in \text{upRules} \\
\text{and} & \ b: r2 = (q \# ps, c) \wedge r2 \in \text{upRules} \\
\text{and} & \ c: r3 = (\text{extend } p q \# ps, c) \wedge r3 \in \text{upRules} \\
\text{and} & \ d: R1 \subseteq \text{upRules} \wedge R = Ax \cup R1 \\
\text{and} & \ (T,n) \in \text{derivable } (R \cup \{r1\} \cup \{r2\})^* \\
\text{shows} & \ (T,n) \in \text{derivable } (R \cup \{r3\})^* 
\end{align*}

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.

\text{theory SequentInvertibility}
\text{imports MultiSequents SingleSuccedent NominalSequents ModalSequents SRCTransforms}
\text{begin}
\text{end}

References