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 ⊥, 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:

\[
\text{datatype } \text{′a form} = \text{At } \text{nat} \\
\quad | \text{Compound } \text{′a } \text{′a form list} \\
\quad | \text{ff}
\]

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

\[
\text{datatype } Gp = \text{con } | \text{dis } | \text{imp} \\
\text{type-synonym } Gp\text{-form } = \text{Gp form}
\]
abbreviation **con-form** \((\text{infixl } \land \ast 80)\) where
\(p \land \ast q \equiv \text{Compound con } [p,q]\)

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

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

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

### 2.2 Rules and Rule Sets

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

**inductive-set** \(Ax\) where
\[ \text{id: } ([\ ], \{ \text{At i } \}) \Rightarrow \ast [\ ] ) \in Ax \]
| \(Lbot: ([\ ], \{ \text{ff } \}) \Rightarrow \ast [\ ] ) \in Ax \)

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

**inductive-set** \(upRules\) where
\[ I: [ [mset c \equiv \{ \text{Compound R Fs } \} ] ; ps \neq [ ] ] \Rightarrow (ps,c) \in upRules \]

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
\[ \text{conL: } ([\ ], \{ A \} + \{ B \} \Rightarrow \ast [\ ]); \{ A \land \ast B \} \Rightarrow \ast [\ ] ) \in g3cp \]
| \(\text{conR: } ([\ ] \Rightarrow \ast \{ A \}; \emptyset \Rightarrow \ast \{ B \} ); \emptyset \Rightarrow \ast \{ A \land \ast B \} ) \in g3cp \]
| \(\text{disL: } ([\ ]; \{ A \} \Rightarrow \ast [\ ]); \{ B \} \Rightarrow \ast [\ ]); \{ A \lor \ast B \} \Rightarrow \ast [\ ] ) \in g3cp \]
| \(\text{disR: } ([\ ]; \{ A \} \Rightarrow \ast [\ ]); \{ B \} \Rightarrow \ast [\ ]); \emptyset \Rightarrow \ast \{ A \lor \ast B \} ) \in g3cp \]
| \(\text{impL: } ([\ ] \Rightarrow \ast \{ A \}; \{ B \} \Rightarrow \ast [\ ]); \{ A \lor B \} \Rightarrow \ast [\ ] ) \in g3cp \]
| \(\text{impR: } ([\ ]; \{ A \} \Rightarrow \ast \{ B \} ); \emptyset \Rightarrow \ast \{ A \lor B \} ) \in g3cp \)

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

Given a rule set 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 \( R \) if and only if 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:

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

where

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

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:

\[
\text{lemma } \text{principal-means-premiss:}
\text{assumes } a: \text{rightPrincipal } r (A \land* B)
\text{and } b: r \in G3cp
\text{shows } (\emptyset \Rightarrow \{ A \}) \in \text{set } (\text{fst } r)
\]

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

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

\[
\text{inductive-set } \text{derivable} :: \text{a rule set} \Rightarrow \text{a deriv set}
\text{for } R :: \text{a rule set}
\text{where}
\text{base: } \{([],C) \in R \} \implies (C,0) \in \text{derivable } R
\mid \text{step: } \{ r \in R ; (\text{fst } r) \neq []; \forall p \in \text{set } (\text{fst } r). \exists n \leq m. (p,n) \in \text{derivable } R \} \\
\implies (\text{snd } r, m + 1) \in \text{derivable } R
\]

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:

\[
\text{inductive-set } \text{derivable} :: \text{a rule set} \Rightarrow \text{a deriv set}
\text{for } R :: \text{a rule set}
\text{where}
\text{base: } \{([],C) \in R \} \implies C \in \text{derivable}' R
\mid \text{step: } \{ r \in R ; (\text{fst } r) \neq []; \forall p \in \text{set } (\text{fst } r). p \in \text{derivable}' R \} \\
\implies (\text{snd } r) \in \text{derivable}' R
\]

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 **asms** by (induct) auto

**Lemma deriv-to-deriv2:**
assumes \(C \in \text{derivable}' R\)
shows \(\exists \ n. (C,n) \in \text{derivable } R\)
using **asms**

**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 \(\forall \ p \in \text{set } (a \# as). \exists \ n. (p,n) \in \text{derivable } R\) 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 \(\forall \ p \in \text{set } (a \# as). \exists \ n \leq (\max m m') (p,n) \in \text{derivable } R\)
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 \(r = (ps,c): \text{and } r \in R:: \exists \ n \leq m. (p,n) \in \text{derivable } R\) using \(ps \neq []\) and
\(\text{derivable.step[where } r=(ps,c) \text{ and } R=R \text{ and } m=m]\) 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)
than 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 premises are empty
moreover
{assume r ∈ Ax
then obtain i where extendRule i r = ([], [At i] ⇒* [At i]) = r ∨
  r = ([], [ff i] ⇒* [ff i]) using characteriseAx[where r=r] by auto
moreover — Case split on the kind of axiom used
{assume r = ([], [At i] ⇒* [At i])
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 = ([], [ff i] ⇒* [ff i])
then have ff ∈ # Γ by auto
then have ff ∈ # 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
}
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
  - \( \forall p \in \text{set } Ps, \exists n \leq n'. (p,n) \in \text{derivable } R* \) by auto
- then obtain \( r S \) where \( r \in Ax \lor r \in R' \)
  - and \text{extendRule } \( S r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F Fs) \) by auto
- moreover
  - \{assume \( r \in Ax \) — Gives a contradiction
  - then have \( \text{fst } r = [] \) apply \((\text{cases } r)\) by \((\text{rule } Ax.\text{cases})\) auto
  - and \( \text{extendRule } S r = (Ps, \Gamma \Rightarrow \Delta \oplus \text{Compound } F Fs) \) by auto
  - ultimately have \( \exists m \leq n. (\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{derivable } R* \) by auto
  \}
- moreover
  - \{assume \( r \in R' \)
  - obtain \( ps c \) where \( r = (ps,c) \) by \((\text{cases } r)\) auto
  - have \((\text{rightPrincipal } r \text{ (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 \( r \):

- \{assume \( \text{rightPrincipal } r \text{ (Compound } F Fs) \)
- then have \((\Gamma' \Rightarrow \Delta') \in \text{set } ps \) using \( b' \) by auto
- then have \( \text{extend } S \) \((\Gamma' \Rightarrow \Delta') \in \text{set } Ps \)
  - using \((\text{extendRule } S r = (Ps,\Gamma \Rightarrow \Delta \oplus \text{Compound } F Fs))\)
  - by \((\text{simp})\)
- moreover have \( S = (\Gamma \Rightarrow \Delta) \) by \((\text{cases } S)\) auto
- ultimately have \((\Gamma + \Gamma' \Rightarrow \Delta + \Delta', m) \in \text{set } Ps \) by \((\text{simp add:extend-def})\)
- 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 \((\text{auto})\)
\}

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 \( \neg(\text{rightPrincipal } r \text{ (Compound } F Fs) \)
- obtain \( \Phi \Psi \) where \( S = (\Phi \Rightarrow \Psi) \) by \((\text{cases } S)\) (auto)
- then obtain \( G H \) where \( c = (G \Rightarrow H) \) by \((\text{cases } c)\) (auto)
- then have \( \exists \text{ Compound } F Fs \) \( \not\exists H \) — Proof omitted
- have \( \Psi + H = \Delta \oplus \text{Compound } F Fs \)
using \( S = (\Phi \Rightarrow \Psi) \) and \( r = (p,s,c) \) and \( c = (G \Rightarrow H) \) by auto

moreover from \( r = (p,s,c) \) and \( c = (G \Rightarrow H) \):

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

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

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

then obtain \( \Psi_{1} \) where \( S = (\Phi \Rightarrow \Psi_{1} \uplus \text{Compound F Fs}) \) by auto

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

by (auto)

then have \( \forall \ p \in \text{set Ps}. \ \exists \Phi' \Psi', m. \ m \leq n' \land \)

\( (\Phi' + \Gamma' \Rightarrow \Psi' + \Delta', m) \in \text{derivable R*} \land \)

\( p = (\Phi' \Rightarrow \Psi' \uplus \text{Compound F Fs}) \) using \( IH \) by (arith)

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

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

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

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

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

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

All of the cases are now complete.

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

qed

As an example, we show the left premiss of \( R \wedge \) 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 conRInvert:**

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

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

proof—

have \( \forall r \in \text{g3cp}. \ \text{rightPrincipal} r (A \wedge B) \rightarrow (\emptyset \Rightarrow \Delta \uplus A) \in \text{set (fst r)} \)

using **principal-means-premiss** by auto

with **assms show ?thesis using rightInvertible by (auto)**

qed

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

**Lemma leftInvertible:**

fixes \( \Gamma \Delta :: 'a form multiset \)

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

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

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

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

A rule is invertible if every premiss is derivable at a height lower than that of the conclusion. A set of rules is invertible if 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 Ex, which will stand for the empty formula:
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**

\[ \text{extend} \equiv \text{extend} \]

\[ \text{extendRule} \equiv \text{extendRule} \]

*begin*

**definition** `extend`

where

\[ \text{extend forms seq} \equiv \]

\[ \text{if} \ (\text{succ seq} = \text{Em}) \]

\[ \text{then} \ (\text{antec forms} \oplus \text{antec seq}) \Rightarrow^* \text{succ forms} \]

\[ \text{else} \ (\text{antec forms} \oplus \text{antec seq} \Rightarrow^* \text{succ seq}) \]

**definition** `extendRule`

where

\[ \text{extendRule forms R} \equiv \]

\[ (\text{map} \ (\text{extend forms}) \ (\text{fst R}), \text{extend forms} \ (\text{snd R})) \]

*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 $(\Gamma \Rightarrow^* \text{Em}, n) \in \text{derivable R}$

and $R' \subseteq \text{upRules}$

and $R = \text{Ax} \cup R'$

shows $(\Gamma \Rightarrow^* C, n) \in \text{derivable R}$  — Proof omitted

Of course, if $C = \text{Em}$, 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 $R' \subseteq \text{upRules} \land R = \text{Ax} \cup R'$

and $(\Gamma \Rightarrow^* \text{Compound Fs}, n) \in \text{derivable R}$

and $\forall \ r' \in R. \ rightPrincipal r' (\text{Compound Fs}) \rightarrow (\Gamma' \Rightarrow^* E) \in \text{set} (\text{fst r'})$

and $E \neq \text{Em}$

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

**lemma** `leftInvertible`:

assumes $R' \subseteq \text{upRules} \land R = \text{Ax} \cup R'$

and $(\Gamma \oplus \text{Compound Fs} \Rightarrow^* \delta, n) \in \text{derivable R}$

and $\forall \ r' \in R. \ leftPrincipal r' (\text{Compound Fs}) \rightarrow (\Gamma' \Rightarrow^* \text{Em}) \in \text{set} (\text{fst r'})$

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

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

**Lemma impRInvert:**

assumes $\Gamma \Rightarrow (A \supset B)$, $n \in \text{derivable}(Ax \cup g3ip)$

shows $\exists m \leq n. \Gamma \Rightarrow (A \supset B)$

**Proof**

- have $\forall r \in (Ax \cup g3ip). \text{rightPrincipal} r (A \supset B) \longrightarrow (\Gamma \Rightarrow (A \supset B)) \in \text{set}(\text{fst } r)$

**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 (rightPrincipal $r (A \supset B)$) have snd $r = (\emptyset \Rightarrow (A \supset B))$ by auto

  with $r \in g3ip$ and (rightPrincipal $r (A \supset B)$)

  have $r = (\Gamma (A) \Rightarrow B), \emptyset \Rightarrow (A \supset B))$ by (rule g3ip.cases) auto

  then have $(\Gamma (A) \Rightarrow B) \in \text{set}(\text{fst } r)$ by auto

  thus $\text{thesis}$ by auto

  qed

  with assms show $\text{thesis}$ 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 (¬(∇ [x].))
| ff

and form-list = FNil
| FCons form form-list

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

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

inductive-set provRules where

[ mset c = ∫ F ∇ [x].A ; ps ≠ [] ; x ♯ set-of-prem (ps − A) ]
⇒ (ps,c) ∈ provRules

inductive-set nprovRules where

[ mset c = ∫ F ∇ [x].A ; ps ≠ [] ]
⇒ (ps,c) ∈ nprovRules

lemma nprovContain:
shows provRules ⊆ nprovRules
proof−
{fix ps c
assume (ps,c) ∈ provRules
then have (ps,c) ∈ nprovRules by (cases) auto
}
then show ?thesis by auto
qed

Substitution is defined in the usual way:

nominal-primrec

subst-form :: var ⇒ var ⇒ form ⇒ form ([·,·])
and subst-forms :: var ⇒ var ⇒ form-list ⇒ form-list ([·,·])
where

[z,y](At P xs) = At P ([z;y]xs)
| x‡(z,y) Γ ⇒ [z,y](F ∇ [x].A) = F ∇ [x].([z,y]A)
| [z,y](Cpd0 F Fs) = Cpd0 F ([z,y]Fs)
| [z,y]ff = ff
| [z,y]FNil = FNil
| [z,y](FCons f Fs) = FCons ([z,y]f) ([z,y]Fs)

Substitution is extended to multisets in the obvious way.

To formalise the condition “no specific substitutions”, an inductive predicate is introduced. If some formula in the multiset Γ is a non-trivial substitution, then multSubst Γ:
definition multSubst :: form multiset ⇒ bool where
multSubst-def: multSubst Γ = (∃ A ∈ (set-mset Γ). ∃ x y. [y,x]B = A ∧ y≠x)

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

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

inductive-set extRules :: rule set ⇒ rule set ( -∗ )
  for R :: rule set
  where
    id: \[ r ∈ R ; r ∈ Ax \] ⇒ extendRule S r ∈ R∗
    sc: \[ r ∈ R ; r ∈ upRules \] ⇒ extendRule S r ∈ R∗
    np: \[ (ps,c) ∈ R ; (ps,c) ∈ provRules ; mset c = \{ F \sqcup [x].A \} ; x ∅ set-of-seq S \] ⇒ extendRule S (ps,c) ∈ 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 formSubst:
  shows y ∅ x ∧ y ∅ A ⇒ F \sqcup [x].A = F \sqcup [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 extend-for-any-seq:
  fixes S :: sequent
  assumes rules: R1 ⊆ upRules ∧ R2 ⊆ nprovRules ∧ R3 ⊆ provRules
  and rules2: R = Ax ∪ R1 ∪ R2 ∪ R3
  and rin: r ∈ R
  shows extendRule S r ∈ R∗

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

assume r ∈ R3
then have r ∈ provRules using rules by auto
obtain ps c where r = (ps,c) by (cases r) auto
then have r1: (ps,c) ∈ R
  and r2: (ps,c) ∈ provRules using \( r ∈ provRules \) and rin by auto
with \( r = (ps,c) \) obtain F x A
  where c = ( \{ F \sqcup [x].A \}) ∨
    c = ( \{ F \sqcup [x].A \} ⇒ ∅) ∧ x ∅ set-of-prem ( ps = A )
  using provRuleCharacterise and \( r ∈ provRules \) by auto
then have mset c = \{ F \nabla [x].A \} \land x \not\in \text{set-of-prem} (ps - A) \text{ by auto}
moreover obtain y where fr: y \not\in x \land
y \not\in A \land
y \not\in \text{set-of-seq} S \land
(y :: \var) \not\in \text{set-of-prem} (ps - A)
using getFresh by auto
then have fr2: y \not\in \text{set-of-seq} S \text{ by auto}
ultimately have mset c = \{ F \nabla [y].[y,x]A \} \land y \not\in \text{set-of-prem} (ps - A)
using formSubst and fr by auto
then have mset c = \{ F \nabla [y].[y,x]A \} \text{ by auto}
then have extendRule S (ps,c) \in R* using r1 and r2 and fr2 and extRules,p by auto
then have extendRule S r \in R* using (r = (ps,c)) 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:

lemma rightInvert:
fixes \Gamma \cdot \Delta :: form multiset
assumes rules: R1 \subseteq \text{upRules} \land R2 \subseteq \text{nproveRules} \land R3 \subseteq \text{provRules} \land R = \text{Ax} \cup R1 \cup R2 \cup R3
and a: (\Gamma \Rightarrow \Delta \cup F \nabla [x].A,n) \in \text{derivable} R*
and b: \forall r' \in R. rightPrincipal r' (F \nabla [x].A) \rightarrow (\Gamma' \Rightarrow \Delta') \in \text{set} (\text{fst} r')
and c: \neg \text{multiSubst} \Gamma' \land \neg \text{multiSubst} \Delta'
shows \exists m \leq n. (\Gamma + \Gamma' \Rightarrow \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 provRules using rules and (r \in R3) by auto
have rightPrincipal r (F \nabla [x].A) \lor \neg rightPrincipal r (F \nabla [x].A) by blast
moreover
{assume rightPrincipal r (F \nabla [x].A)
then have (\Gamma' \Rightarrow \Delta') \in \text{set} ps using (r = (ps,c)) and (r \in R3) and rules by auto
then have extend S (\Gamma' \Rightarrow \Delta') \in \text{set} Ps using
\langle\text{extendRule} S r = (Ps,\Gamma \Rightarrow \Delta \cup F \nabla [x].A)\rangle
and (r = (ps,c)) by (simp add:extendContain)
moreover from (rightPrincipal r (F \nabla [x].A)) have
\begin{align*}
c &= (\emptyset \Rightarrow (F \nabla [x].A)) \\
\text{using (r = (ps,c)) by (cases r) auto}
\end{align*}
with (extendRule S r = (Ps,\Gamma \Rightarrow \Delta \cup F \nabla [x].A)) have S = (\Gamma \Rightarrow \Delta) \\
using (r = (ps,c)) 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*
using (n = Suc n') by (simp)
}

lemma leftInvert:
fixes $\Gamma$, $\Delta :: \text{form multiset}$
assumes rules: $R1 \subseteq \text{upRules} \land R2 \subseteq \text{nprovRules} \land R3 \subseteq \text{provRules} \land R = Ax \cup R1 \cup R2 \cup R3$
  
  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 $c: \neg \text{multSubst } \Gamma' \wedge \neg \text{multSubst } \Delta'$
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:

```plaintext
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 \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:

```plaintext
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:

\[
\text{inductive-set } \text{modRules2 with }
\begin{array}{l}
\text{\{ } ps \neq \text{\{ } Modal M Ms \text{\} } \implies (ps,c) \in \text{modRules2}
\end{array}
\]

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 \( \bullet \)), and returns the set of active parts extended with \( ! \cdot \Gamma \Rightarrow \bullet \cdot \Delta \):

\[
\text{inductive-set } p-e : (\langle a,b \rangle \text{ rule set } \Rightarrow 'b \Rightarrow (\langle a,b \rangle \text{ rule set for } R : \langle a,b \rangle \text{ rule set and } M N : 'b)
\]

\[
\text{where }
\begin{array}{l}
\text{\{ } (Ps, c) \in R ; R \subseteq \text{modRules2 } \} \implies \text{extendRule } (M \cdot \Gamma \Rightarrow N \cdot \Delta) (Ps, c) \in p-e \text{ R M N}
\end{array}
\]

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

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

\[
\text{begin}
\]

\[
\text{definition } \text{extendConc } : (\langle a,b \rangle \text{ sequent } \Rightarrow (\langle a,b \rangle \text{ rule }) (\langle a,b \rangle \text{ rule for } R \text{ R' } : (\langle a,b \rangle \text{ rule set and } M N : 'b)
\]

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

\[
\text{end}
\]

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

\[
\text{inductive-set } \text{ext } : (\langle a,b \rangle \text{ rule set } \Rightarrow (\langle a,b \rangle \text{ rule set for } R \text{ R' } : (\langle a,b \rangle \text{ rule set and } M N : 'b)
\]

\[
\text{where }
\begin{array}{l}
\text{ax: } \text{\{ } r \in R ; r \in \text{Ax } \} \implies \text{extendRule } \text{seq } r \in \text{ext } R R' M N
\mid \text{up: } \text{\{ } r \in R ; r \in \text{upRules } \} \implies \text{extendRule } \text{seq } r \in \text{ext } R R' M N
\mid \text{mod1: } \text{\{ } r \in p-e \text{ R' } M N ; r \in R \} \implies \text{extendConc } \text{seq } r \in \text{ext } R R' M N
\mid \text{mod2: } \text{\{ } r \in R ; r \in \text{modRules2 } \} \implies \text{extendRule } \text{seq } r \in \text{ext } R R' M N
\end{array}
\]

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:

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

\[
\text{fixes } \Gamma \text{ } \Delta : (\langle a,b \rangle \text{ form multiset}
\]

\[
\text{assumes rules: } R1 \subseteq \text{upRules } \land R2 \subseteq \text{modRules2 } \land R3 \subseteq \text{modRules2 } \land
\]

\[
\]
\[ R = Ax \cup R1 \cup (p-e R2 M1 M2) \cup R3 \]
\[ R' = Ax \cup R1 \cup R2 \cup R3 \]

and a: \((\Gamma \Rightarrow \star \Delta \oplus \text{Modal } M M_s, n) \in \text{derivable} (\text{ext } R R2 M1 M2)\)

and b: \(\forall r' \in R', \text{rightPrincipal } r' (\text{Modal } M M_s) R' \rightarrow (\Gamma' \Rightarrow \star \Delta') \in \text{set} (\text{fst } r')\)

and \(\text{neq: } M2 \neq M\)

shows \(\exists m \leq n. (\Gamma + \Gamma' \Rightarrow \star \Delta + \Delta', m) \in \text{derivable} (\text{ext } R R2 M1 M2)\)

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

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

\{assume \( ba: r \in \text{p-e } R2 M1 M2 \land \)
extendConc \( S r = (Ps, \Gamma \Rightarrow \star \Delta \oplus \text{Modal } M M_s)\)
with \(\text{rules } obtain\) \( F Fs \) \( \Gamma'' \Delta'' ps r' \) where
ca: $r = \text{extendRule} \left( M_1 \cdot \Gamma'' \Rightarrow^* M_2 \cdot \Delta'' \right) \ r'$ and
cb: $r' \in R_2$ and
c: $r' = (ps, \emptyset \Rightarrow^* (\text{Modal F Fs} \ j)) \lor r' = \left( ps, \text{Modal F Fs} \ j \Rightarrow^* \emptyset \right)$

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 \( \text{fst} \left( \text{extendConc} \left( \Gamma_1 + \Gamma' \Rightarrow^* \Delta_2 + \Delta' \right) \ r \right) = \text{Ps} \)
by (auto simp add: extendConc-def)
ultimately have \( \left( \Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', n' + 1 \right) \in \text{derivable} \ (\text{ext R R_2 M_1 M_2}) \)
by auto
then have \( \exists \ m \leq n. \ (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', m) \in \text{derivable} \ (\text{ext R R_2 M_1 M_2}) \)
using \( \langle n = \text{Suc n'} \rangle \) by auto
ultimately have \( \exists \ m \leq n. \ (\Gamma + \Gamma' \Rightarrow^* \Delta + \Delta', m) \in \text{derivable} \ (\text{ext R R_2 M_1 M_2}) \)
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 succeedent 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:

\[
\begin{align*}
\text{!} \cdot \Gamma & \Rightarrow A, \bullet \cdot \Delta \\
\Gamma', \text{!} \cdot \Gamma & \Rightarrow \bullet A, \bullet \cdot \Delta, \Delta' \\
R_1 \\
\bullet \cdot \Gamma & \Rightarrow \text{!} \cdot \Delta \\
\Gamma', \bullet \cdot \Gamma & \Rightarrow \bullet A, \text{!} \cdot \Delta, \Delta' \\
R_2
\end{align*}
\]

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

\[
\begin{align*}
\text{!} \cdot \Gamma_1, \bullet \cdot \Gamma_2 & \Rightarrow A, \text{!} \cdot \Delta_1, \bullet \cdot \Delta_2 \\
\Gamma', \text{!} \cdot \Gamma_1, \bullet \cdot \Gamma_2 & \Rightarrow \bullet A, \text{!} \cdot \Delta_1, \bullet \cdot \Delta_2, \Delta'
\end{align*}
\]
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 \text{g3mod2}
where
\[
\text{diaR: } (\emptyset \Rightarrow^* i A \ j), \emptyset \Rightarrow^* i \Diamond A \ j) \in \text{g3mod2} \\
\boxL: (\text{[i] A} \ j \Rightarrow^* \emptyset), \text{[i] \Box A} \ j \Rightarrow^* \emptyset) \in \text{g3mod2}
\]

inductive-set \text{g3mod1}
where
\[
\text{boxR: } (\emptyset \Rightarrow^* i A \ j), \emptyset \Rightarrow^* i \Box A \ j) \in \text{g3mod1} \\
\text{diaL: } (\text{[i] A} \ j \Rightarrow^* \emptyset), \text{[i] \Diamond A} \ j \Rightarrow^* \emptyset) \in \text{g3mod1}
\]

We then show the strong admissibility of the rule:
\[\Gamma \Rightarrow \square A, \Delta\]
\[T \Rightarrow A, \Delta\]

**Lemma invertBoxR:**

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

and \((\Gamma \Rightarrow \ast \Bigoplus (\square A), \mathbf{n}) \in \text{derivable} \ (\text{ext} R g3mod1 \square \Diamond)\)

**Shows** \(\exists \ m \leq \mathbf{n}. \ (\Gamma \Rightarrow \ast \Bigoplus A, m) \in \text{derivable} \ (\text{ext} R g3mod1 \square \Diamond)\)

**Proof:**

- From Assms show ?thesis
  - Using principal and rightInvert and g3 by auto

where **principal** is the result which fulfils 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 \(P = Q\#P'\), or if \(P = P'\#Q\), and so on. The order of the premisses is not really important, since the formalisation operates on the finite set based upon the list. The more general “append” lemma could be proved from the lemma we give; we prove the inductive step case in the proof of such an append lemma. This is a height preserving transformation. Some of the proof is shown:

**Lemma removeRedundant:**

**Assumes** \(r1 = (p\#ps, c) \land r1 \in \text{upRules}\)

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

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

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

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

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

**Case** 0

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

then have \(([] , T) \in (R \cup \{r1\} \cup \{r2\})^\ast\) 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 premisses must be an axiom. Since \( \mathcal{R} \) contains the set of axioms, so will \( \mathcal{R} \cup r_2 \):

\[
\begin{align*}
\text{then have } r &\in (\mathcal{R} \cup \{r_2\}) \text{ using } c \text{ by auto} \\
\text{then have } (T,0) &\in \text{ derivable } (\mathcal{R} \cup \{r_2\})* \text{ by auto} \\
\text{then show } \exists m \leq n. (T,m) &\in \text{ derivable } (\mathcal{R} \cup \{r_2\})* \text{ using } (n = 0) \text{ by auto}
\end{align*}
\]

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

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:

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

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: }

\[
\begin{align*}
\text{assumes } r_1 &= ((\emptyset \Rightarrow \star \{A\}) \# ((\{A\} \Rightarrow \star \emptyset) \# ps),c) \land r_1 \in \text{ upRules} \\
\text{and } R1 &\subseteq \text{ upRules} \land R = Ax \cup R1 \\
\text{and } (T,n) &\in \text{ derivable } (\mathcal{R} \cup \{r_1\})* \\
\text{and } CA: \forall \Gamma \Delta A. ((\Gamma \Rightarrow \star \Delta \oplus A) \in \text{ derivable' } R*) \rightarrow
\end{align*}
\]
\[(Γ ⊕ A ⇒* Δ) ∈ \text{derivable}' R^* \rightarrow \]
\[(Γ ⇒* Δ) ∈ \text{derivable}' R^* \]

shows \[T ∈ \text{derivable}' R^* \]

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

**lemma** removeCombinable:

**assumes** \[a: r_1 = (p \ # \ ps,c) \land r_1 ∈ \text{upRules} \]

**and** \[b: r_2 = (q \ # \ ps,c) \land r_2 ∈ \text{upRules} \]

**and** \[c: r_3 = (\text{extend } p \ q \ # \ ps, c) \land r_3 ∈ \text{upRules} \]

**and** \[d: R_1 \subseteq \text{upRules} \land R = \text{Ax} \cup R_1 \]

**and** \[(T,n) ∈ \text{derivable} (R \cup \{r_1\} \cup \{r_2\})^* \]

**shows** \[(T,n) ∈ \text{derivable} (R \cup \{r_3\})^* \]

**8 Conclusions**

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

**theory** SequentInvertibility

**imports** MultiSequents SingleSuccedent NominalSequents ModalSequents SRCTransforms

begin

end

**References**