Generic Methods for Formalising Sequent Calculi Applied to Provability Logic

Jeremy E. Dawson and Rajeev Goré

Logic and Computation Group, School of Computer Science
The Australian National University, Canberra ACT 0200, Australia
http://users.rsise.anu.edu.au/~jeremy/
http://users.rsise.anu.edu.au/~rpg/

Abstract. We describe generic methods for reasoning about multiset-based sequent calculi which allow us to combine shallow and deep embeddings as desired. Our methods are modular, permit explicit structural rules, and are widely applicable to many sequent systems, even to other styles of calculi like natural deduction and term rewriting systems. We describe new axiomatic type classes which enable simplification of multiset or sequent expressions using existing algebraic manipulation facilities. We demonstrate the benefits of our combined approach by formalising in Isabelle/HOL a variant of a recent, non-trivial, pen-and-paper proof of cut-admissibility for the provability logic **GL**, where we abstract a large part of the proof in a way which is immediately applicable to other calculi. Our work also provides a machine-checked proof to settle the controversy surrounding the proof of cut-admissibility for **GL**.

Keywords: provability logic, cut admissibility, interactive theorem proving, proof theory

1 Introduction

Sequent calculi provide a rigorous basis for meta-theoretic studies of various logics. The central theorem is cut-elimination/admissibility, which states that detours through lemmata can be avoided, since it can help to show many important logical properties like consistency, interpolation, and Beth definability. Cut-free sequent calculi are also used for automated deduction, for nonclassical extensions of logic programming, and for studying the connection between normalising lambda calculi and functional programming. Sequent calculi, and their extensions, therefore play an important role in logic and computation.

Meta-theoretic reasoning about sequent calculi is error-prone because it involves checking many combinatorial cases, some being very difficult, but many being similar. Invariably, authors resort to expressions like "the other cases are similar", or "we omit details". The literature contains many examples of meta-theoretic proofs containing serious and subtle errors in the original pencil-and-paper proofs. For example, the cut-elimination theorem for the modal "provability logic" **GL**, where $\Box\varphi$ can be read as "φ is provable in Peano Arithmetic", has a long and chequered history which has only recently been resolved [5].

When reasoning about sequent calculi using proof assistants, we have to represent proof-theoretical concepts like “sequents”, “derivations” and “derivability” in the meta-logic of the given proof assistant. The granularity of the representation plays a critical role in determining the versatility of the representation. At one extreme, we have “deep” embeddings in which a sequent and a derivation are represented explicitly as terms of the meta-logic, as espoused in our previous work on display calculi [2]. The advantage of this approach is that it allows us to encode fine-grained notions like “each structure variable appears at most once in the conclusion of a rule”. The disadvantage of this approach is that it requires a lot of ancillary low-level work to capture essential notions like “rule instance”. At the other extreme, we have “shallow” embeddings like the one in the Isabelle/Sequents package, which allows us to derive sequents, but does not allow us to reason about the derivations or derivability. Most practitioners [8, 9, 3] choose an approach somewhere between these extremes, which then limits their ability to generalise their work to other calculi or to handle arbitrary structural rules for example.

Here, we describe general methods for reasoning in Isabelle/HOL about the proof-theory of traditional, propositional, multiset-based sequent calculi. Our methods are modular in allowing an arbitrary set of rules, permit explicit structural rules, and are widely applicable to many sequent systems, even to other styles of calculi like natural deduction and term rewriting systems. They advance the “state of the art” in that they allow us to “mix and match” shallow and deep embeddings where appropriate, which, as far as we know, has not been done before. We then show how we used them to formalise a highly non-trivial proof of cut-admissibility for **GL** based on, but different from, that of [5].

In Section 2, we briefly describe traditional sequent calculi, discuss the need for multisets, and describe the controversy surrounding the cut-elimination theorem for the set-based sequent system GLS for provability logic **GL** . In Section 3 we describe general deep and shallow techniques and functions we have defined for reasoning about derivations and derivability, independently of the rules of a particular sequent system. We give very general induction principles which are useful beyond the application to GLS. We show how we formalise formulae, sequents and rules, and then show some of the **GL** sequent rules as examples. In Section 4 we describe an Isabelle axiomatic type class which we developed to facilitate reasoning about multisets of formulae, and sequents based on them. This development explores the interaction between lattice ($\wedge$, $\vee$, $\leq$) and “arithmetic” ($+$, $-$, 0, $\leq$) operations. In Section 5 we discuss how we made our Isabelle proofs as general as possible, and how they are useful for proving cut-elimination and other results in arbitrary sequent calculi which meet the associated preconditions. In Section 6 we describe the cut-admissibility proof for GLS.

We assume the reader is familiar with basic proof-theory, ML and Isabelle/HOL. In the paper we show some Isabelle code, edited to use mathematical symbols. The Appendix gives the actual Isabelle text of many definitions and theorems. Our Isabelle code can be found at `http://users.rsise.anu.edu.au/~jeremy/isabelle/200x/seqms/`. Some of this work was reported informally in [4].

2 Sequents, Multisets, Sets and Provability Logic

Proof-theorists typically work with sequents $\Gamma \vdash \Delta$ where Γ and Δ are "collections" of formulae. The "collections" found in the literature increase in complexity from simple sets for classical logic, to multisets for linear logic, to ordered lists for substructural logics, to complex tree structures for display logics. A sequent rule typically has a rule name, a (finite) number of premises, a side-condition and a conclusion. Rules are read top-down as "if all the premises hold then the conclusion holds". A derivation of the judgement $\Gamma \vdash \Delta$ is typically a finite tree of judgements with root $\Gamma \vdash \Delta$ where parents are obtained from children by "applying a rule". We use "derivation" to refer to a proof *within* a calculus, reserving "proof" for a meta-theoretic proof of a theorem *about* the calculus.

Provability logic **GL** is obtained by adding Löb's axiom $\Box(\Box A \to A) \to \Box A$ to propositional normal modal logic K. Its Kripke semantics is based on rooted transitive Kripke frames without infinite forward chains. It rose to prominence when Solovay showed that $\Box A$ could be interpreted as "A is provable in Peano Arithmetic" [10]. An initial proof-theoretic account was given by Leivant in 1981 when he "proved" cut-elimination for a set-based sequent calculus for **GL** [6]. But Valentini in 1983 found a simple counter-example and gave a new cut-elimination proof [11]. In 2001, Moen [7] claimed that Valentini's transformations don't terminate if the sequents $\Gamma \vdash \Delta$ are based on multisets. There is of course no *a priori* reason why a proof based on sets should not carry over with some modification to a proof based on multisets. The issue was recently resolved by Goré and Ramanayake [5] who gave a pen-and-paper proof that Moen is incorrect, and that Valentini's proof using multisets is *mostly* okay.

The sequent system GLS for the logic **GL** as given by Goré and Ramanayake in [5], like Valentini's, contains explicit weakening and contraction rules and, modulo these, a typical (additive) set of rules for the classical logical connectives $\neg, \wedge, \vee, \to$. The axiom $A \vdash A$ does not require atomic A. Since GLS admits axiom extension, it could have been formulated using $p \vdash p$, for p atomic. In fact the general result Theorem 2 doesn't depend on the axiom or on axiom extension.

The one additional rule GLR which characterises **GL** is shown below:

$$\frac{\Box X, X, \Box B \vdash B}{\Box X \vdash \Box B}\ GLR \text{ or } \ GLR(B) \text{ or } \ GLR(X, B)$$

The rule is unusual since the principal formula $\Box B$ changes polarity from conclusion to premise. To identify the principal formula involved, or all the formulae, we refer to it as $GLR(B)$, or $GLR(X, B)$. The full set of rules of GLS is shown in [5], but note that our formalisation does not regard the cut or multicut rules shown below as being part of the system.

We show a context-sharing cut rule and a context-splitting multicut rule. Given the contraction and weakening rules, the context-sharing and context-splitting versions are equivalent; our formal proofs show the admissibility of a context-splitting multicut rule where A is not contained in Γ'' or Δ''.

$$(\text{cut})\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta} \qquad (\text{multicut})\ \frac{\Gamma' \vdash A^n, \Delta' \qquad \Gamma'', A^m \vdash \Delta''}{\Gamma', \Gamma'' \vdash \Delta', \Delta''}$$

Thus our results will be lemmata of the form: if $\Gamma \vdash A, \Delta$ and $\Gamma, A \vdash \Delta$ are derivable then $\Gamma \vdash \Delta$ is derivable, where "derivable" means without using (cut).

3 Reasoning About Derivations and Derivability

3.1 Deep and Shallow Embeddings

In [2] we proved cut admissibility for $\boldsymbol{\delta}$**RA**, the display calculus for relation algebras. The proof was based on a proof of Belnap, which applied generally to display calculi whose inference rules satisfied certain properties. In that paper we described the formalisation as a "deep embedding". We now believe that to describe a particular formalisation as either a "deep embedding" or a "shallow embedding" over-simplifies the issue as we now discuss.

In [2], we modelled a derivation tree in HOL as a recursive structure which consists of a root sequent (which should be the conclusion of some rule), and an associated list of (sub)trees (each of which should derive a premise of that rule). The type of explicit derivation trees is a recursive datatype, such as:

```
datatype 'a dertree = Der 'a ('a dertree list)
    | Unf 'a (* unfinished leaf which remains unproved *)
```

We then had to express the property of such a tree, that it is in fact correctly based on the given rules, and so represents a valid derivation. We modelled a sequent rule as an object in HOL consisting of a list of premises and a conclusion, each of which was a sequent in the language of the logic (of relation algebras). Notably, our language of formulae included "variables" which could be instantiated with formulae, so we defined functions for such substitutions. This was necessary because we had to express conditions on the rules such as that a variable appearing in the premises also appears in the conclusion. It is much more common to let the variables in a rule be the variables of the theorem prover, and for the theorem prover to do the substitution. Thus it is more accurate to say that in [2], we used a deep embedding for *derivations*, *rules* and *variables*, since we modelled each of these features of the proof-theory explicitly rather than identifying them with analogous features of Isabelle.

In alternative ("shallow derivations") approaches to work of this type, the set of derivable sequents can be modelled as an inductively defined set, where the derivation tree does not exist, but it "happens" in the course of proving that a specific sequent is derivable. That is, derivability in the sequent calculus studied equates to derivability in Isabelle, giving a shallow embedding of *derivations*.

We now briefly describe the functions we used to describe derivability. More details are in Appendix A.1, and a more expository account is given in [4].

```
types 'a psc = "'a list * 'a"       (* single step inference *)
consts
  derl, adm :: "'a psc set => 'a psc set"
  derrec    :: "'a psc set => 'a set => 'a set"
```

An inference rule of type `'a psc` is a list of premises `ps` and a conclusion `c`. Then `derl rls` is the set of rules derivable from the rule set `rls`, and `adm rls` is the set of admissible rules of the rule set `rls`, and `derrec rls prems` is the set of sequents derivable using rules `rls` from the set `prems` of premises. These were defined separately using Isabelle's package for inductively defined sets as below. Thus, using shallow derivations, the rules of the sequent calculus can either be encoded explicitly as in `derrec` or they can be encoded in the clauses for the inductive definition of the set of derivable sequents as in `ders`.

Shallow Embedding of Derivations and Deep Embedding of Rules:

$$(\{\Gamma \vdash P,\ \ \Gamma \vdash Q\},\ \ \Gamma \vdash P \wedge Q) \in \texttt{rules} \qquad \text{(etc for other rules)}$$

$$c \in \texttt{prems} \Longrightarrow c \in \texttt{derrec rules prems}$$
$$[|\ (ps, c) \in \texttt{rules}\ ;\ ps \subseteq \texttt{derrec rules prems}\ |] \Longrightarrow c \in \texttt{derrec rules prems}$$

Shallow Embedding of Derivations and Shallow Embedding of Rules:

$$c \in \texttt{prems} \Longrightarrow c \in \texttt{ders prems}$$
$$[|\ \Gamma \vdash P \in \texttt{ders prems}\ ;\ \Gamma \vdash Q \in \texttt{ders prems}\ |] \Longrightarrow \Gamma \vdash P \wedge Q \in \texttt{ders prems}$$

The first clause for `derrec` says that each premise c in `prems` is derivable from `prems`, and the second says that a sequent c "obtained" from a set of derivable sequents ps by a rule (ps, c) is itself derivable. The set of rules `rules` is a parameter. The first clause for `ders` also says that each premise c in `prems` is derivable from `prems`. The rules however are no longer a parameter but are hard-coded as clauses in the definition of `ders` itself.

Thus, with a shallow embedding of derivations, we have a choice of either a deep or a shallow embedding of *rules*. It would also be possible to combine our deep embedding of derivations (`dertree`) with a shallow embedding of rules by encoding the rules in the function which checks whether a derivation tree is valid. Note however, that when we use a deep embedding of derivations in Lemma 3, our definition of validity is parameterised over the set of rules.

Our framework is generic in that a rule merely consists of "premises" and a "conclusion", and is independent of whether the things derived are formulae, sequents, or other constructs, but we will refer to them as sequents.

Our experience is that shallow embeddings generally permit easier proofs, but sometimes they are inadequate to express a desired concept. For example, with a deep embedding of rules it is easy to express that one set of rules is a subset of another set, but with a shallow embedding this is not possible. With a deep embedding of derivation trees it is easy to express that one property of derivation trees is the conjunction of two other properties, or that a derivation tree has a particular height, since each such tree has an explicit representation as a term, whereas to express such things using `ders` or `derrec` (as above), one would have to redefine these predicates incorporating the particular properties of interest. Indeed, in this work (see §6) we discuss how we found a shallow embedding of derivability inadequate, and we describe there how we "mix and match" the various styles as required.

3.2 Properties of Our Generic Derivability Predicates

We obtained the expected results linking `derl` and `derrec`, and a number of results expressing transitivity of derivation and the results of derivation using derived rules, of which the most elegant are:

```
derl_deriv_eq : "derl (derl ?rls) = derl ?rls"
derrec_trans_eq : "derrec ?rls (derrec ?rls ?prems)
                    = derrec ?rls ?prems"
```

`derl_deriv_eq` states that derivability using derived rules implies derivability using the original rules

`derrec_trans_eq` states that derivability from derivable sequents implies derivability from the original premises.

A simplified version of the induction principle generated by the definition of the inductive set `derrec` is as follows:

$$\frac{x \in \mathit{derrec\ rls\ prems} \quad \forall c \in \mathit{prems}.P\ c \quad \forall(\mathit{ps}, c) \in \mathit{rls}.(\forall p\ \mathit{in}\ \mathit{ps}.P\ p) \Rightarrow P\ c}{P\ x}$$

The principle says that if each rule preserves the property P from its premises to its conclusion, then so does the whole derivation, even though there is no explicit derivation as such. We contrast this principle with induction on the height of derivations where it is possible, and sometimes necessary, to transform a subtree in some height-preserving (or height-reducing) way, and assume that the transformed tree has property P. Such transformations are not available when using the induction principle above.

Where we have a property of two derivations, such as cut-admissibility, we need a more complex induction principle **Ind** which we derived, though, again, there are no explicit representations of derivation trees:

$$\frac{\begin{array}{c} \mathit{cl} \in \mathit{derrec\ rlsl}\ \{\} \qquad \mathit{cr} \in \mathit{derrec\ rlsr}\ \{\} \\ \forall(\mathit{lps}, \mathit{lc}) \in \mathit{rlsl}.\forall(\mathit{rps}, \mathit{rc}) \in \mathit{rlsr}. \\ (\forall \mathit{lp} \in \mathit{lps}.P\ \mathit{lp}\ \mathit{rc}) \wedge (\forall \mathit{rp} \in \mathit{rps}.P\ \mathit{lc}\ \mathit{rp}) \Rightarrow P\ \mathit{lc}\ \mathit{rc} \end{array}}{P\ \mathit{cl}\ \mathit{cr}}$$

Ind: Suppose cl and cr are the conclusions of the left and right subderivations. Assume that for every rule pair $(\mathit{lps}, \mathit{lc})$ and $(\mathit{rps}, \mathit{rc})$, if each premise lp in lps satisfies $P\ lp\ rc$, and each premise rp in rps satisfies $P\ lc\ rp$, then $P\ lc\ rc$ holds. Then we can conclude that $P\ cl\ cr$ holds.

Finally, (ps, c) is an admissible rule iff: if all premises in ps are derivable, then c is too: $(ps, c) \in \texttt{adm rls} \iff (ps \subseteq \texttt{derrec rls}\ \{\} \Rightarrow c \in \texttt{derrec rls}\ \{\})$.

We obtained the following four results, which were surprisingly tricky because `adm` is not monotonic (since any rule with a premise is in `adm` $\{\}$). For example, the last of these says that a derived rule, derived from the admissible rules, is itself admissible.

```
"derl ?rls <= adm ?rls"              "adm (adm ?rls) = adm ?rls"
"adm (derl ?rls) = adm ?rls"         "derl (adm ?rls) = adm ?rls"
```

3.3 Sequents, Formulae and the GLS rules

We define a language of formula connectives, formula variables and primitive (atomic) propositions:

```
datatype formula = FC string (formula list) (* formula connective *)
                 | FV string                (* formula variable *)
                 | PP string              (* primitive proposition *)
```

Although the formula connectives are fixed for each logic, the datatype is more general, using a single constructor `FC` for all formula connectives. We then define (for example) $P \wedge Q$ as `FC ''Btimes'' [P, Q]`. A sequent is a pair of multisets of formulae, written $\Gamma \vdash \Delta$.

Given a rule such as ($\vdash \wedge$) in the two forms below,

$$\mathcal{C}_s = \frac{\vdash A \quad \vdash B}{\vdash A \wedge B} \qquad\qquad \mathcal{C}_e = \frac{X \vdash Y, A \quad X \vdash Y, B}{X \vdash Y, A \wedge B}$$

we call $\mathcal{C}_e$ an *extension* of $\mathcal{C}_s$, and we define functions `extend` and `pscmap`, so that, where $+$ means multiset union,

$$\texttt{extend}\ (X \vdash Y)\ (U \vdash V) = (X + U) \vdash (Y + V)$$
$$\mathcal{C}_e = \texttt{pscmap}\ (\texttt{extend}\ (X \vdash Y))\ \mathcal{C}_s$$

Then we define `glss`, the set of rules of GLS by defining:

`glil` and `glir`: the unextended left and right introduction rules, like $\mathcal{C}_s$ above;
`wkrls` and `ctrrls` A: the unextended weakening and contraction (on A) rules;
`glne`: all of the above;
`glr` B: the *GLR*(B) rule;
`glss`: the axiom $A \vdash A$ (not requiring A to be atomic), the *GLR*(B) rule for all B, and all extensions of all rules in `glne`.

The Isabelle definitions are given in Appendix A.2. Note that in the *GLR* rule, X is a multiset, and $\Box X$ is informal notation for applying $\Box$ to each member of X; this is formalised as `mset_map`, used in the formalised *GLR* rule. Using a similar notation we write $\Box B^k$ for $(\Box B)^k$, the multiset containing n copies of $\Box B$. Development of `mset_map` and relevant lemmas is in the source files `Multiset_no_le.{thy,ML}`. Our results there also show multisets form a monad, see Appendix A.3 for details.

Our treatment of sequents and formulae amounts to a deep embedding of sequents and formulae which is independent of the set of rules. The implementation in [8] is a shallow embedding of sequents, which automatically implies the admissiblity of certain structural rules like contraction and weakening.

4 An Axiomatic Type Class for Multisets and Sequents

Isabelle provides a theory of finite multisets with an ordering which we did not use; we defined a different ordering $\leq$ analogous to $\subseteq$ for sets: $N \leq M$ if, for all x, N contains no more occurrences of x than does M.

An axiomatic type class in Isabelle is characterised by a number of axioms, which hold for all members of a type in the type class. The multiset operators $\leq$, $+$, $-$ and 0 have several useful properties, which are described by the axiomatic type classes `pm0` and `pm_ge0`. For any type in class `pm0`, the operations $+$ and 0 form a commutative monoid and the following two properties hold.

$$A + B - A = B \qquad\qquad A - B - C = A - (B + C)$$

We then define a class `pm_ge0` which also has an $\leq$ operator and a smallest element 0, in which the axioms of `pm0` and the following hold.

$$0 \leq A \qquad\qquad B \leq A \Rightarrow B + (A - B) = A$$
$$m \leq n \Leftrightarrow m - n = 0 \qquad x < y \Leftrightarrow x \leq y \wedge x \neq y \qquad a \sqsubseteq b \Leftrightarrow a \leq b$$

For a new type, the last three axioms could be made definitions, but for natural numbers, $\leq$ and $<$ are already defined. The naturals obey all these axioms. We explain $\sqsubseteq$ below.

Lemma 1. *Multisets are in `pm0` and `pm_ge0` using our definition of $\leq$, and, if Γ and Δ are of any type in the classes `pm0` or `pm_ge0`, then so is sequent $\Gamma \vdash \Delta$.*

Isabelle has "simplification procedures", which will (for example) simplify $a-b+c+b$ to $a+c$ for integers, or $a+b+c-b$ to $a+c$ for integers *or* naturals. We have been able to apply the ones which apply to naturals, other than those involving the successor function `Suc`, to types of the classes `pm0` and `pm_ge0`. This was a very great help in doing the proofs discussed in §6, especially since $X \vdash Y$ can be derived from $X' \vdash Y'$ by weakening if and only if $X \vdash Y \leq X' \vdash Y'$.

It is easy to show that, in the obvious way, multisets form a lattice. In fact we found the interesting result that the axioms of `pm_ge0` are sufficient to give a lattice. Isabelle's lattice type class uses $\sqsubseteq$ as the order operator, so, defining $a \sqsubseteq b$ iff $a \leq b$, we have:

Lemma 2. *Any type of class `pm_ge0` forms a lattice, using the definitions*

$$c \wedge d = c - (c - d) \qquad\qquad c \vee d = c + (d - c)$$

From these definitions it is possible (at some length) to prove the axioms for a lattice and so any type in the class `pm_ge0` is also in Isabelle's class `lattice`. The source files for this development are `pmg*.{thy,ML}`.

5 Capturing the Core of Cut-Admissibility Proofs

Many cut-elimination proofs proceed via two main phases. The first phase transforms the given derivations using several "parametric" steps so that the cut-formula is the principal formula of the final rule in the resulting sub-derivations. The "principal cut" is then "reduced" into cuts which are "smaller" in some well-founded ordering such as size of cut-formula. We describe how we captured this two-phase structure of cut-elimination proofs, and present a widely applicable result that a parametric step is possible under certain conditions.

In §3.2 we mentioned the induction principle **Ind** used for deriving cut-admissibility, or indeed any property P of pairs of subtrees. In the diagram below, to prove $P(\mathcal{C}_l, \mathcal{C}_r)$, the induction hypothesis is that $P(\mathcal{P}_{li}, \mathcal{C}_r)$ and $P(\mathcal{C}_l, \mathcal{P}_{rj})$ hold for all i and j:

$$\dfrac{\dfrac{\mathcal{P}_{l1} \ldots \mathcal{P}_{ln}}{\mathcal{C}_l}\rho_l \qquad \dfrac{\mathcal{P}_{r1} \ldots \mathcal{P}_{rm}}{\mathcal{C}_r}\rho_r}{?}\ (cut\ ?)$$

A proof of $P(\mathcal{C}_l, \mathcal{C}_r)$ using this induction hypothesis inevitably proceeds by cases on the actual rules ρ_l and ρ_r, and for a cut-formula A, on whether it is principal in either or both of ρ_l and ρ_r. But we also use induction on the size of the cut-formula, or, more generally, on some well-founded relation on formulae. So we actually consider a property P of a (cut) formula A and (left and right subtree conclusion) sequents $(\mathcal{C}_l, \mathcal{C}_r)$. In proving $P\ A\ (\mathcal{C}_l, \mathcal{C}_r)$, in addition to the inductive hypothesis above, we assume that $P\ A'\ (\mathcal{D}_l, \mathcal{D}_r)$ holds generally for A' smaller than A and all "`rls`-derivable" sequents $\mathcal{D}_l$ and $\mathcal{D}_r$: *i.e.* derivable from the empty set of premises using rules from rls. These intuitions give the following definition `gen_step2ssr` of a condition which permits one step of the inductive proof. See Appendix A.5 for reference to related more complex predicates and theorems.

Definition 1 (`gen_step2ssr`). *For a formula A, a property P, a subformula relation* `sub`*, a set of rules* `rls`*, inference rule instances $\mathcal{R}_l = (\mathcal{P}_{l1} \ldots \mathcal{P}_{ln}, \mathcal{C}_l)$ and $\mathcal{R}_r = (\mathcal{P}_{r1} \ldots \mathcal{P}_{rm}, \mathcal{C}_r)$,* `gen_step2ssr` *$P\ A$* `sub rls` *$(\mathcal{R}_l, \mathcal{R}_r)$ means:*

if *forall A' such that $(A', A) \in$* `sub` *and all* `rls`*-derivable sequents $\mathcal{D}_l$ and $\mathcal{D}_r$,*
$P\ A'\ (\mathcal{D}_l, \mathcal{D}_r)$ holds
and for each $\mathcal{P}_{li}$ in $\mathcal{P}_{l1} \ldots \mathcal{P}_{ln}$, $P\ A\ (\mathcal{P}_{li}, \mathcal{C}_r)$ holds
and for each $\mathcal{P}_{rj}$ in $\mathcal{P}_{r1} \ldots \mathcal{P}_{rm}$, $P\ A\ (\mathcal{C}_l, \mathcal{P}_{rj})$ holds
then *$P\ A\ (\mathcal{C}_l, \mathcal{C}_r)$ holds.*

The main theorem `gen_step2ssr_lem` below for proving an arbitrary property P states that if the step of the inductive proof is possible for all cases of final rule instances $\mathcal{R}_l$ and $\mathcal{R}_r$ on each side, then P holds in all cases.

Theorem 1 (`gen_step2ssr_lem`). *If*

- `A` *is in the well-founded part of the subformula relation* `sub`*,*
- *sequents $\mathcal{S}_l$ and $\mathcal{S}_r$ are* `rls`*-derivable, and*
- *for all formulae A', and all rules $\mathcal{R}_l$ and $\mathcal{R}_r$, our induction step condition* `gen_step2ssr` *$P\ A'$* `sub rls` *$(\mathcal{R}_l, \mathcal{R}_r)$ holds*

then $P\ A\ (\mathcal{S}_l, \mathcal{S}_r)$ holds.

This enables us to split up an inductive proof, by showing, separately, that `gen_step2ssr` holds for particular cases of the final rules $(\mathcal{P}_l, \mathcal{C}_l)$ and $(\mathcal{P}_r, \mathcal{C}_r)$ on each side. For example, the inductive step for the case where the cut-formula A

$$\dfrac{\mu\left\{\dfrac{\Pi_l}{\Box X, X, \Box B \vdash B}\right.}{\Box X \vdash \Box B}\,GLR(B) \qquad \dfrac{\Pi_r}{\Box B^k, Y \vdash Z}\,\rho$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \; (multicut\ ?)$$
$$\Box X, Y \vdash Z$$

Fig. 1. A multicut on cut formula $\Box B$ where $\Box B$ is left-principal via *GLR*

is parametric, not principal, on the left is encapsulated in the following theorem where `prop2 mar erls` A $(\mathcal{C}_l, \mathcal{C}_r)$ means that the conclusion of a multicut on `A` whose premises are $\mathcal{C}_l$ and $\mathcal{C}_r$ is derivable using rules `erls`.

Theorem 2 (`lmg_gen_steps`). *For any relation `sub` and any rule set `rls`, given an instance of multicut with left and right subtrees ending with rules $\mathcal{R}_l$ and $\mathcal{R}_r$:*

if *weakening is admissible for the rule set `erls`,*
and all extensions of some rule $(\mathcal{P}, X \vdash Y)$ are in the rule set `erls`,
and $\mathcal{R}_l$ is an extension of $(\mathcal{P}, X \vdash Y)$,
and the cut-formula A is not in Y (meaning that A is parametric on the left)
then *`gen_step2ssr (prop2 mar erls)` A `sub rls` $(\mathcal{R}_l, \mathcal{R}_r)$ holds.*

Theorem 2 codifies multi-cut elimination for parametric cut-formulae, and applies generally to many different calculi since it holds independently of the values of `sub` and `rls`. Of course, for a system with explicit weakening rules, such as GLS, weakening is *a fortiori* admissible. As we note later, the proof for GLS involves one really difficult case and a lot of fairly routine cases. In dealing with the routine cases, automated theorem proving has the benefit of ensuring that no detail is overlooked. Moreover, as in this example, we often have more general theorems that apply directly to a set of rules such as GLS.

Notice that all of this section has used a shallow embedding of derivations since no explicit derivation trees were required.

6 The Proof of Cut-Admissibility for GLS

Valentini's proof of cut-admissibility for GLS uses a triple induction on the size of the cut-formula, the heights of the derivations of the left and right premises of cut, and a third parameter which he called the "width". Roughly speaking, the width of a cut-instance is the number of *GLR* rule instances above that cut which contain a parametric ancestor of the cut-formula in their conclusion. The Goré and Ramanayake [5] pen-and-paper proof follows Valentini but gives a constructive way to calculate the width of a cut by inspecting the branches of the left and right sub-derivations of a cut rule instance.

As usual, the proof of cut-admissibility for GLS proceeds by considering whether the cut-formula is principal in the left or right premise of the cut, or principal in both. The crux of the proof is a "reduction" when the cut-formula is of the form $\Box B$ and is principal in both the left and right premises of the cut. The solution is to replace this cut on $\Box B$ by cuts which are "smaller" either because their cut-formula is smaller, or because their width is smaller. In reality, most of the work involves a cut instance which is only left-principal as shown in Figure 1, and most of this section is devoted to show how we utilised our general methods to formalise these "reductions" as given by Goré and Ramanayake [5]. But there are some important differences which we now explain.

As explained in §3.2, our general methods do not model derivation trees explicitly, since we use a shallow embedding. So our proof uses induction on the size of the cut-formula and on the fact of derivation, rather than on the size of the cut-formula and the height of derivation trees. Also, we actually proved the admissibility of "multicut": that is, if $\Gamma' \vdash A^n, \Delta'$ and $\Gamma'', A^m \vdash \Delta''$ are both cut-free derivable, then so is $\Gamma', \Gamma'' \vdash \Delta', \Delta''$. This avoids problems in the cases where these sequents are derived using contraction on A.

In all other aspects, our proof of cut-admissibility for GLS is based on that given by Goré and Ramanayake [5]. In particular, although we do not actually require [5, Lemma 19], we use the construction in that lemma, which is fundamental to overcoming the difficulties of adapting standard proofs to GLS, as we explain shortly. Consequently, our proof uses the idea of induction on the *width* as defined in [5], although as we shall see, our proof is expressed in terms of `del0`, which approximates to the ∂^0 of [5], not width *per se*.

To cater for width/∂^0, we could have altered our shallow embedding, `derrec`, but that destroys the modularity of our approach. Instead, we defined our `del0` by using the datatype `dertree` from §3.1 to represent explicit derivation tree objects. These trees, and the general lemmas about them, are similar to the trees of [2]. Thus we "mix and match" a deep embedding of derivation trees with a shallow embedding of inductively defined sets of derivable sequents.

To ensure the correctness of our "mixing and matching" we needed the following relationship between our definitions of derivability according to the two embeddings. A *valid* tree is one whose inferences are in the set of rules and which as a whole has no premises.

Lemma 3. *Sequent $X \vdash Y$ is derivable, shallowly, from the empty set of premises using rules `rls` (ie, is in `derrec rls` $\{\}$) iff some explicit derivation tree `dt` is valid wrt. `rls` and has a conclusion $X \vdash Y$.*

```
"(?a : derrec ?rls {}) = (EX dt. valid ?rls dt & conclDT dt = ?a)"
```

We now define a function `del0` which is closely related to ∂^0 and the width of a cut of [5], although we can avoid using the annotated derivations of [5].

Definition 2 (`del0`)**.** *For derivation tree `dt` and formula B, define `del0` B `dt`:*

- *if the bottom rule of `dt` is GLR(Y, A) (for* any *Y, A), then `del0` B `dt` is 1 (0) if $\Box B$ is (is not) in the antecedent of the conclusion of `dt`*
- *if the bottom rule of `dt` is not GLR, then `del0` B `dt` is obtained by summing `del0` B `dt'` over all premise subtrees `dt'` of `dt`.*

Thus, the calculation of `del0` traces up each branch of an explicit derivation tree until an instance of the GLR rule is found: it then counts 1 if $\Box B$ is in the antecedent, meaning that B is in the X of the statement of GLR. Where the derivation tree branches below any GLR rule, the value is summed over the branches. (When we use `del0`, its argument will be the sub-tree μ of Figure 1.)

We now give a sequence of lemmata which eventually give the "reduction" for left-and-right GLR-principal occurrences of a cut-formula of the form $\Box B$.

Lemma 4 (`gr19e`). *If μ is a valid derivation tree with conclusion $\Box X, X, \Box B \vdash B$, and `del0` B $\mu = 0$, then $\Box X, X \vdash B$ is derivable.*

Proof. By applying the GLR rule to the conclusion, we can derive $\Box X \vdash \Box B$. Tracing $\Box B$ upwards in μ, it is parametric in each inference, except possibly weakening, contraction or the axiom $\Box B \vdash \Box B$. That is, since `del0` B $\mu = 0$, when we meet a GLR inference as we trace upwards, $\Box B$ must have already disappeared (through weakening). So, tracing upwards, we can change each instance of $\Box B$ to $\Box X$ in the usual way. The axiom $\Box B \vdash \Box B$ is changed to $\Box X \vdash \Box B$, which is derivable. Contraction is not problematic since we use, as the inductive hypothesis, that *all* occurrences of $\Box B$ can be replaced by $\Box X$. $\dashv$

To abbreviate the statement of several lemmas, we define a function `muxbn`, based on Figure 1, which is from [5]. It concerns a multicut on a cut-formula $\Box B$ which is left-principal because the bottom rule on the left is $GLR(B)$.

Definition 3 (`muxbn`). *`muxbn` B n holds iff: for all instances of Figure 1 (for fixed B) such that `del0` B $\mu \leq n$, the multicut in Figure 1 is admissible.*

The next lemma says that multicut admissibility on B implies `muxbn` B 0.

Lemma 5 (`del0_ca'`, `caB_muxbn'`). *If μ is a valid derivation tree with conclusion $\Box X, X, \Box B \vdash B$, and `del0` B $\mu = 0$, and multicut on B is admissible, and $\Box B^k, Y \vdash Z$ is derivable, then $\Box X, Y \vdash Z$ is derivable.*

That is, if multicut on B is admissible, then `muxbn` B 0 holds.

Proof. $\Box X \vdash \Box B$ is derivable from $\Box X, X, \Box B \vdash B$ via $GLR(X, B)$. By Lemma 4, $\Box X, X \vdash B$ is derivable. The rest of the proof is by induction on the derivation of $\Box B^k, Y \vdash Z$, in effect, by tracing relevant occurrences of $\Box B$ up that derivation. Weakening and contraction involving $\Box B$ are not problematic, and the axiom $\Box B \vdash \Box B$ is changed to $\Box X \vdash \Box B$, which is derivable. Suppose an inference $GLR(Y, C)$ is encountered, where B is in Y. This inference, allowing for multiple copies of B, is as shown below left where Z is Y with B deleted:

$$\dfrac{\Box B^k, B^k, \Box Z, Z, \Box C \vdash C}{\Box B^k, \Box Z \vdash \Box C}\,GLR(Y,C) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Lemma 4}}{\Box X, X \vdash B} \quad \Box X, B^k, \Box Z, Z, \Box C \vdash C}{\Box X, \Box X, X, \Box Z, Z, \Box C \vdash C}\,mcut(B)}{\Box X, X, \Box Z, Z, \Box C \vdash C}\,ctr}{\Box X, \Box Z \vdash \Box C}\,GLR(C)$$

By induction, we have $\Box X, B^k, \Box Z, Z, \Box C \vdash C$ is derivable. From there we have the derivation shown above right. As the machine-checking process showed us, additional weakening steps are necessary if $\Box B$ is in Z or if B is in $\Box Z$. $\dashv$

This construction is like that of case 2(a)(ii) in the proof of Theorem 20 of [5]. Having shown, in Lemma 5, that `muxbn` B 0 holds, we now use the construction of [5, Lemma 19] to show that `muxbn` B n holds for all n: except that our inductive assumptions and results involve admissibility of multicut, not cut. Again we use induction: we assume that `muxbn` B n, and show that `muxbn` B $(n+1)$ holds.

So suppose a derivation tree $\mu/\Box X \vdash \Box B$ has a bottom inference $GLR(X, B)$, as shown in Figure 1, and `del0` B $\mu = n+1$. We follow the construction of [5, Lemma 19] to obtain a derivation μ' of $\Box X, X, \Box B \vdash B$, where `del0` B $\mu' \leq n$.

Since `del0` B $\mu > 0$, the tree $\mu/\Box X \vdash \Box B$ is as shown below left (with other branches not shown). We first delete the topmost application of $GLR(A)$ leaving a tree μ^-. Then adjoin $\Box A$ to each antecedent of μ^- obtaining the tree on the right (call it $\mu^A/\Box A, \Box X \vdash \Box B$), whose topmost sequent is now a weakened axiom, and which requires us to weaken in an occurrence of A just above the final GLR-rule instance:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Box G, G, \Box B^k, B^k, \Box A \vdash A}{\Box G, \Box B^k \vdash \Box A}\,GLR(A)}{\vdots}}{\Box X, X, \Box B \vdash B}}{\Box X \vdash \Box B}\,GLR(X, B) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Box A, \Box G, \Box B^k \vdash \Box A}{\vdots}}{\Box A, \Box X, X, \Box B \vdash B}}{\Box A, A, \Box X, X, \Box B \vdash B}\,(weakening)}{\Box A, \Box X \vdash \Box B}\,GLR(B)$$

Now `del0` B $\mu >$ `del0` B μ^A, and so $\mu^A/\Box A, \Box X \vdash \Box B$ can be used as the left branch of an admissible (i.e. "smaller") multicut. We do this twice, the right branches being derivations of $\Box X, X, \Box B \vdash B$ and $\Box G, G, \Box B^k, B^k, \Box A \vdash A$ respectively; which after contractions, give derivations of $\Box A, \Box X, X \vdash B$ and $\Box G, G, \Box X, B^k, \Box A \vdash A$. These two are cuts 1 and 2 of the description in [5, Lemma 19], producing derivations which are cut-free equivalents of Λ_{11} and Λ_{12} from [5, Lemma 19]. The result `gr19a` in the Isabelle code gives the existence of these two derivations; it uses the result `add_Box_ant` which permits adding $\Box A$ to the antecedent of each sequent of a derivation tree.

We combine these derivations of $\Box A, \Box X, X \vdash B$ and $\Box G, G, \Box X, B^k, \Box A \vdash A$ using an admissible ("smaller") multicut on B, and then use contraction to obtain $\Box G, G, \Box A, \Box X, X \vdash A$. This is cut 3 of [5, Lemma 19]. Then the GLR rule gives $\Box G, \Box X \vdash \Box A$. This is derivation Λ_2 of [5, Lemma 19]. Because of this GLR rule at this point, we do not need to worry about the `del0` B values of any of the subtrees mentioned above, whose existence is given by the inductive assumptions of multicut-admissibility. We now weaken the conclusion of this tree to $\Box X, \Box G, \Box B^k \vdash \Box A$, giving (a counterpart of) the derivation Λ_3 of [5, Lemma 19].

Returning to μ^-, as below left, we this time adjoin $\Box X$ in the antecedent, giving the tree below right, but we can now use Λ_3 as a derivation for its leaf:

That is, we have replaced a given derivation μ of $\square X, X, \square B \vdash B$ where `del0` $B\ \mu = n + 1$, with a derivation μ' of $\square X, X, \square B \vdash B$ where `del0` $B\ \mu' = n$.

$$\dfrac{\dfrac{\dfrac{\square G, \square B^k \vdash \square A}{\vdots}}{\square X, X, \square B \vdash B}}{\square X \vdash \square B}\,GLR(\mathrm{X,B}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Lambda_3}{\square X, \square G, \square B^k \vdash \square A}}{\vdots}}{\square X, \square X, X, \square B \vdash B}\,(contraction)}{\square X, X, \square B \vdash B}}{\square X \vdash \square B}\,GLR$$

We record this as the result as Lemma 6(a) (`gr19b`) below, however, we do not use this result directly. Instead, we obtain Lemma 6(b) (`gr19c'`) below by referring to Figure 1 and noting that we have replaced μ by μ'.

Lemma 6 (`gr19b, gr19c'`)**.** *Assume that multicut-admissibility holds for cut-formula B, and that* `muxbn` *B n holds.*

(a) If μ is a derivation of $\square X, X, \square B \vdash B$, where `del0` *$B\ \mu = n + 1$, then there exists another derivation μ' of $\square X, X, \square B \vdash B$ with* `del0` *$B\ \mu' \leq n$.*
(b) `muxbn` *B $(n + 1)$ holds.*

Proof. The proof of the first part is the construction described above. For the second part, we are given an instance of Figure 1, where μ has `del0` $B\ \mu = n+1$. Using the first part, we can transform μ to μ', with `del0` $B\ \mu' = n$. Since `muxbn` B n holds, the multicut on $\square B$ where the left premise derivation is $\mu'/\square X \vdash \square B$ is admissible. Hence the conclusion $\square X, Y \vdash Z$ of Figure 1 is derivable. ⊣

The next result, which we do not use directly, approximates to [5, Lemma 19].

Lemma 7 (`gr19d`)**.** *If multicut-admissibility holds for cut-formula B,* `muxbn` *B 0 holds, and $\square X, X, \square B \vdash B$ is derivable, then $\square X, X \vdash B$ is derivable.*

Proof. By using Lemma 6(b) repeatedly, `muxbn` B n holds for any n. ⊣

Lemma 8 (`caB_muxbn, cut_glr`)**.** *If multicut-admissibility holds for cut-formula B, then it holds for cut-formula $\square B$ for the case where the left premise derivation ends with a GLR(B) rule. That is,* `muxbn` *B n holds for any n.*

Proof. By combining Lemma 5 with repeated use of Lemma 6(b). ⊣

We now have the admissibility of principal multicuts on $\square B$, which we indicated was the crux of the cut-admissibility for **GL**. For the other cases, the usual proofs hold. Although there is no particular difficulty in showing this, many details need to be checked: this takes almost 3 pages of [5], even with "omitting details", and citing "standard transformations". The value of automated theorem proving at this point is simply to ensure that all the necessary details have been checked. The proof uses techniques described in §5, which means that a relatively small amount of the proofs remaining for this theorem are peculiar to the GLS calculus. Consequently, we get:

Theorem 3 (`glss_ca`)**.** *Multicut is admissible in GLS.*

7 Conclusions

We have described a formalised proof in the Isabelle theorem prover of cut admissibility for the sequent calculus GLS for the provability logic **GL**. The proof is based on the one from [5], particularly the construction in [5, Lemma 19], though we use it in a slightly different way. The Isabelle proof moves to and fro between our deep embedding of explicit trees (`dertree`) and our shallow embedding using `derrec`, so our proof has demonstrated the technique of combining use of a shallow embedding where adequate with a deep embedding where necessary. The work in §5, while complex, succeeds in extracting those parts of the cut elimination proof which follow a common pattern, and expressing them in a way which will be useful beyond the particular sequent system GLS.

We have made considerable use of definitions and theorems which are useful beyond this particular sequent system, or, indeed, beyond proofs about sequent systems. We have developed axiomatic type classes based on properties of $+$, $-$, 0 and $\leq$ for multisets, and shown how a rather small set of axioms about these operators is sufficient for defining a lattice. Multiset sequents (pairs of multisets) also belong to this type class. We have applied relevant simplification procedures to this type class, which was useful in our proofs. We have described extensive definitions and theorems relating to abstract derivability, which we have used in several different metalogical theories and proofs, and we have discussed the issue of deep versus shallow embeddings in the light of this and previous work.

References

1. J E Dawson and R Goré. Embedding display calculi into logical frameworks: Comparing Twelf and Isabelle. *ENTCS* 42, 89–103.
2. J E Dawson and R Goré. Formalised Cut Admissibility for Display Logic. In Proc. TPHOLS'02, LNCS 2410, 131–147, Springer, 2002.
3. A Gacek. The Abella interactive theorem prover (system description) In Proc. IJCAR 2008 LNCS 5195:154-161, Springer, 2008.
4. R Goré. Machine Checking Proof Theory: An Application of Logic to Logic. Invited talk, Third Indian Conference on Logic and Applications, Chennai, January 2009.
5. R Goré and R Ramanayake. Valentini's Cut-elimination for Provability Logic Resolved. Proc. AiML 2008, pp 67-86, College Publications `http://users.rsise.anu.edu.au/~rpg/publications.html`
6. D Leivant. On the Proof Theory of the Modal Logic for Arithmetic Provability. Journal of Symbolic Logic, 46:531538, 1981.
7. A Moen. The proposed algorithms for eliminating cuts in the provability calculus GLS do not terminate. NWPT 2001, Norwegian Computing Center, 2001-12-10 `http://publ.nr.no/3411`
8. F Pfenning. Structural cut elimination. In *Proc. LICS 94*, 1994.
9. F Pfenning and C Schürmann. System description: Twelf a meta-logical framework for deductive systems. In Proc. CADE-16, LNAI 1632: 202–206, Springer, 1999.
10. R M Solovay. Provability Interpretations of Modal Logic. Israel Journal of Mathematics, 25:287304, 1976.
11. S Valentini. The Modal Logic of Provability: Cut-elimination. Journal of Philosophical Logic, 12:471476, 1983.

A Isabelle text of selected definitions and theorems

A.1 More details of our General Derivability Predicates

We now describe more fully the functions we used to describe derivability. This framework is general in that a rule merely consists of "premises" and a "conclusion", and is independent of whether the things derived are formulae, sequents, or other constructs, but we will refer to them as sequents.

```
types 'a psc = "'a list * 'a"       (* single step inference *)
consts
  derl, adm :: "'a psc set => 'a psc set"
  dersl     :: "'a psc set => ('a list * 'a list) set"
  dercsl    :: "'a psc set => ('a list list * 'a list) set"
  derrec    :: "'a psc set => 'a set => 'a set"
  dersrec   :: "'a psc set => 'a set => 'a list set"
```

An inference rule (of type `'a psc`) is a list of premises `ps` and a conclusion `c`. Then, `derl rls` is the set of rules derivable from the rule set `rls`, and `derrec rls prems` is the set of sequents derivable using rules `rls` from the set `prems` of premises. These were defined separately using Isabelle's package for inductively defined sets, using also the functions `dersl` and `dersrec`. So $(\texttt{ps}, \texttt{c}) \in$ `derl rls` reflects the shape of a derivation tree: `ps` is a list of exactly the premises used, in the correct order, whereas $\texttt{c} \in$ `derrec rls prems` holds even if the set of premises `prems` contains superfluous sequents.

The auxiliary function `dersrec` represents several sequents, all derivable from the premises. The auxiliary function `dersl` represents several derivation trees side by side; $(\texttt{ps}, \texttt{cs}) \in$ `dersl rls` when `ps` is the concatenation of their lists of premises, and `cs` is the list of their conclusions. The function `dercsl` is similar, but it shows which premises are part of which tree. That is, $(\texttt{pss}, \texttt{cs}) \in$ `dercsl rls` implies $(\texttt{concat pss}, \texttt{cs}) \in$ `dersl rls`. Curiously, we have used this framework for several years, and only recently we found the need to redefine `derl` using `dercsl` rather than `dersl` as the auxiliary function.

The key clauses for defining `derl` and `derrec` are shown below (the full definitions are in the source file `gen/dtrel.{thy,ML}`):

```
  dtderI : "[| (ps, concl) : pscrel ; (pss, ps) : dersl pscrel |]
              ==> (pss, concl) : derl pscrel"
  derI   : "[| (ps, concl) : pscrel ; ps : dersrec pscrel prems |]
              ==>  concl : derrec pscrel prems"
```

We obtained the expected results linking `derl` and `derrec`, and a number of results expressing transitivity of derivation and the results of derivation using derived rules, for example:

`derrec_trans_eq` derivability from derivable sequents is derivability from the original premises `derl_deriv_eq` derivability (in terms of `derrec`) using derived rules is derivability using the original rules

`derrec_derl_deriv_eq` derivability (in terms of `derl`) using derived rules is derivability using the original rules

`derl_trans` composition of derivation trees (joining conclusions to premises) gives a derivation tree

```
derrec_trans_eq : "derrec ?rls (derrec ?rls ?prems) = derrec ?rls ?prems"
derl_deriv_eq : "derl (derl ?rls) = derl ?rls"
derrec_derl_deriv_eq : "derrec (derl ?rls) ?prems = derrec ?rls ?prems"
derl_trans : "[| (?ps, ?c) : derl ?rls; (?x, ?ps) : dersl ?rls |]
              ==> (?x, ?c) : derl ?rls"
```

The expected link between `derl` and `derrec` was proved in the following form, by defining `derrc` from `derl`, and proving `derl_rr`.

```
inductive "derrc pr prems"
  intros
    rcI : "[| (ps, c) : derl pr ; set ps <= prems |] ==>
          c : derrc pr prems"

derl_rr : "derrec == derrc"
```

A.2 Definitions of the GLS formulae, sequents and rules

We define a language of formula connectives, formula variables and primitive (atomic) propositions:

```
datatype formula = FC string (formula list) (* formula connective *)
                 | FV string                (* formula variable *)
                 | PP string             (* primitive proposition *)
```

Although the formula connectives are fixed for each logic, we made the datatype as general as possible, and then defined (for example) $P \wedge Q$ as `FC ''Btimes'' [P, Q]`.

```
consts Btimes :: "formula => formula => formula" ("_ && _" [67,68] 67)
       Bplus  :: "formula => formula => formula" ("_ v _" [63,64] 63)
defs   Btimes "Btimes f g == FC ''Btimes'' [f, g]"
       Bplus  "Bplus f g == FC ''Bplus'' [f, g]"
```

This permits a single definition of the immediate (proper) subformula relation, `ipsubfml`, which will not need to be changed when new connectives are added:

```
consts ipsubfml :: "(formula * formula) set"
inductive "ipsubfml" (* proper immediate subformula relation *)
  intros  ipsI : "P : set Ps ==> (P, FC conn Ps) : ipsubfml"
```

If we call $\mathcal{C}_e$ (in the main text) an *extension* of $\mathcal{C}_s$, `extrs` S means the set of all extensions of all rules in the set S.

```
consts
  extend :: "'a sequent => 'a sequent => 'a sequent"
  extrs  :: "'a sequent psc set => 'a sequent psc set"

defs
  extend_def : "extend fmls seq == seq + fmls"
  pscmap_def : "pscmap f (ps, c) = (map f ps, f c)"

inductive "extrs rules"
  intros
    I : "psc : rules ==> pscmap (extend flr) psc = epsc ==>
        epsc : extrs rules"
```

Then we define `glss`, the set of rules of GLS. Sets `wkrls` and `ctrrls` A refer to weakening and contraction (on A) rules (before extension).

```
inductive "wkrls"
  intros
    L : "([0], {#A#} |- {#}) : wkrls"
    R : "([0], {#} |- {#A#}) : wkrls"

inductive "ctrrls A"
  intros
    L : "([{#A#} + {#A#} |- {#}], {#A#} |- {#}) : ctrrls A"
    R : "([{#} |- {#A#} + {#A#}], {#} |- {#A#}) : ctrrls A"

inductive "glir" (* right introduction rules *)
  intros
    andr : "([{#} |- {#A#}, {#} |- {#B#}], {#} |- {#Btimes A B#}) : glir"
    orr1 : "([{#} |- {#A#}], {#} |- {#A v B#}) : glir"
    orr2 : "([{#} |- {#B#}], {#} |- {#A v B#}) : glir"
    negr : "([{#A#} |- {#}], {#} |- {#--A#}) : glir"
    (* single premise right implication rule *)
    impr : "([{#A#} |- {#B#}], {#} |- {#A -> B#}) : glir"

inductive "glil" (* left introduction rules *)
  intros
    andl1 : "([{#A#} |- {#}], {#Btimes A B#} |- {#}) : glil"
    andl2 : "([{#B#} |- {#}], {#Btimes A B#} |- {#}) : glil"
    orl : "([{#A#} |- {#}, {#B#} |- {#}], {#A v B#} |- {#}) : glil"
    negl : "([{#} |- {#A#}], {#--A#} |- {#}) : glil"
    impl : "([{#B#} |- {#}, {#} |- {#A#}], {#A -> B#} |- {#}) : glil"

inductive "glne" (* rules before being extended *)
```

```
  intros
    wkI : "(ps, c) : wkrls ==> (ps, c) : glne"
    ctrI : "(ps, c) : ctrrls A ==> (ps, c) : glne"
    ilI : "(ps, c) : glil ==> (ps, c) : glne"
    irI : "(ps, c) : glir ==> (ps, c) : glne"

inductive "glr B"
intros I : "([mset_map Box X + X + {#Box B#} |- {#B#}],
              mset_map Box X |- {#Box B#}) : glr B"

inductive "glss"
intros axiom : "([], {#A#} |- {#A#}) : glss"
        extI : "psc : glne ==> pscmap (extend flr) psc : glss"
        glrI : "psc : glr B ==> psc : glss"
```

A.3 Multisets form a monad

These results use the characterisation of a monad which uses three axioms and the functions *unit* (here `single`) and *ext* (here `mset_ext`). (These axioms are often expressed with *bind* in place of *ext*). We also show the definition of *map* (here `mset_map`).

```
  single :: "'a => 'a multiset"    ("{#_#}")
  mset_ext :: "('a => 'b multiset) => 'a multiset => 'b multiset"
  mset_map :: "('a => 'b) => 'a multiset => 'b multiset"

mset_map_def : "mset_map ?f == mset_ext (single o ?f)"

mset_ext_of_single : "mset_ext ?f {#?x#} = ?f ?x"
mset_ext_o_single : "mset_ext ?f o single = ?f"
multiset_E3 : "mset_ext (mset_ext ?f o ?g) = mset_ext ?f o mset_ext ?g"
```

A.4 Definitions of an atom in a lattice

We produced a number of equivalent definitions characterising atoms of the lattice, and properties of atomic elements (our definition of `atomic` includes 0). Thus we showed that "x is atomic" is equivalent to each of the following:

$$\forall y < x.\ y = 0$$

$$\forall z.\ x \leq z \vee (pg\_meet\ x\ z = 0) \qquad \forall ab.\ x \leq a + b \Rightarrow x \leq a \vee x \leq b$$

$$\forall ab.\ 0 < a \wedge 0 < b \Rightarrow a + b \neq x \qquad \forall ab.\ a < x \wedge b < x \Rightarrow a + b \neq x$$

A.5 Definition 1 and Theorem 1

See also the definition `gen_step2sr_simp` and the result `gen_step2sr_lem`, which is stronger, but not (here) usefully so, and the theorem `gen_step2sr_s_imp` relating them.

```
gen_step2ssr_simp :
"gen_step2ssr ?P ?A ?sub ?rls ((?psl, ?cl), (?psr, ?cr)) =
( (ALL A'. (A', ?A) : ?sub -->
   (ALL da:derrec ?rls {}. ALL db:derrec ?rls {}. ?P A' (da, db))) -->
  (ALL pa:set ?psl. ?P ?A (pa, ?cr)) -->
  (ALL pb:set ?psr. ?P ?A (?cl, pb)) --> ?P ?A (?cl, ?cr) )"

gen_step2ssr_lem :
 "[| ?A : wfp ?sub ;
     ?seql : derrec ?rls {} ; ?seqr : derrec ?rls {} ;
     ALL A. ALL (psl, cl):?rls. ALL (psr, cr):?rls.
         gen_step2ssr ?P A ?sub rls ((psl, cl), (psr, cr)) |]
     ==> ?P ?A (?seql, ?seqr)"
```

A.6 Definitions of mar, mas and masdt

mar rls $A\,(X_l \vdash Y_l, X_r \vdash Y_r)$ means that the result of doing multicut elimination of A for these two sequents, ie $(X_l + X_r \setminus A \vdash Y_l \setminus A + Y_r)$, is derivable.

mas rls A $(X_l \vdash Y_l, X_r \vdash Y_r)$ means the same, conditional upon the given sequents, $X_l \vdash Y_l$ and $X_r \vdash Y_r$, being derivable.

masdt is similar, except that its arguments are derivation trees rather than sequents, and uses the condition that they are valid.

A derivation tree is valid according to a set of rules if each "rule" contained in the tree is in fact a rule in the given set.

```
consts
  mar :: "'a sequent psc set => 'a => ('a sequent * 'a sequent) set"
  mas :: "'a sequent psc set => 'a => ('a sequent * 'a sequent) set"
  masdt ::  "'a sequent psc set => 'a =>
    ('a sequent dertree * 'a sequent dertree) set"

inductive "mar pscrel A"
  intros
    I : "(Xl + ms_delete {A} Xr |- ms_delete {A} Yl + Yr) : derrec pscrel {}
            ==> (Xl |- Yl, Xr |- Yr) : mar pscrel A"

inductive "mas pscrel A"
  intros
    I : "seql : derrec pscrel {} & seqr : derrec pscrel {} -->
              (seql, seqr) : mar pscrel A ==>
        (seql, seqr) : mas pscrel A"

consts
  masdt ::  "'a sequent psc set => 'a =>
    ('a sequent dertree * 'a sequent dertree) set"
```

```
inductive "masdt pscrel A"
  intros
    I : "valid pscrel dtl & valid pscrel dtr -->
            (conclDT dtl, conclDT dtr) : mar pscrel A ==>
          (dtl, dtr) : masdt pscrel A"
```

A.7 Theorem 2

```
lmg_gen_steps :
  "[| wk_adm ?erls; extrs {(?ps, ?c)} <= ?erls; ~ ?A :# succ ?c;
        ?pscl = pscmap (extend ?flr) (?ps, ?c) |]
    ==> gen_step2ssr (prop2 mar ?erls) ?A ?sub ?rls (?pscl, ?pscr)"
```

A.8 Definition 2

Here `glra` is the set of *all* *GLR*-rule instances, `antec` is a function to return the antecedent of a sequent, and `:#` is multiset membership.

```
primrec Der  : "del0 B (Der seq dtl) =
               (if (map conclDT dtl, seq) : glra then
                   if Box B :# antec seq then 1 else 0
                else del0s B dtl)"
        Unf  : "del0 B (Unf dt) = 0"

        Nil  : "del0s B [] = 0"
        Cons : "del0s B (dt # dts) = del0 B dt + del0s B dts"
```

A.9 Definition 3

```
defs muxbn_def : "muxbn B n ==
     ALL mu dtb seq. ([conclDT mu], seq) : glr B -->
      del0 B mu <= n --> (Der seq [mu], dtb) : masdt glss (Box B)"
```

A.10 Lemma 5

```
del0_ca' :
  "[| valid glss ?dt; del0 ?B ?dt = 0; mas glss ?B = UNIV;
    ([conclDT ?dt], ?sq) : glrxb ?X ?B; ?x : derrec glss {} |] ==>
  (mset_map Box ?X |- 0) +
    seq_delete (mset_map Box {#?B#} |- 0) ?x : derrec glss {}"

caB_muxbn' : "mas glss ?B = UNIV ==> muxbn ?B 0"
```

A.11 Lemma 4

```
"[| valid glss ?mu ; del0 ?B ?mu = 0 ; (conclDT ?mu) =
      (mset_map Box ?X + ?X + {# Box ?B#} |- {#?B#}) |]
  ==> (mset_map Box ?X + ?X |- {#?B#}) : derrec glss {}"
```

A.12 Lemma 6(a) and (b)

```
gr19b : "[| ((conclDT ?mu) =
             (mset_map Box ?X + ?X + {# Box ?B#} |- {#?B#})) ;
            valid glss ?mu ; del0 ?B ?mu = Suc ?n ;
            mas glss ?B = UNIV; muxbn ?B ?n |]
      ==> EX mupr. valid glss mupr &
                  conclDT mupr = conclDT ?mu & del0 ?B mupr <= ?n"

gr19c' : "[| muxbn ?B ?n; mas glss ?B = UNIV |] ==> muxbn ?B (Suc ?n)"
```

A.13 Lemma 8

```
cut_glr :
  "[| mas glss ?B = UNIV; ([conclDT ?mu], ?seq) : glr ?B |] ==>
  (Der ?seq [?mu], ?dtb) : masdt glss (Box ?B)"

caB_muxbn : "mas glss ?B = UNIV ==> muxbn ?B ?n"
```

A.14 Theorem 3

```
mar_glss :
  "[| ?seqa : derrec glss {}; ?seqb : derrec glss {} |] ==>
  (?seqa, ?seqb) : mar glss ?A"

glss_ca :
"[| (?Xl |- ?Yl) : derrec glss {}; (?Xr |- ?Yr) : derrec glss {} |] ==>
(?Xl + ms_delete {?A} ?Xr |- ms_delete {?A} ?Yl + ?Yr) : derrec glss {}"
```