Replace this file with `prentcsmacro.sty` for your meeting,
or with `entcsmacro.sty` for your meeting.
Both can be found at the ENTCS Macro Home Page.

A New Machine-checked Proof of Strong Normalisation for Display Logic

Jeremy E. Dawson [1,3] Rajeev Goré [2,4]

Department of Computer Science and Automated Reasoning Group
Australian National University
Canberra ACT 0200, Australia

Abstract

We use a deep embedding of the display calculus for relation algebras $\boldsymbol{\delta}\mathbf{RA}$ in the logical framework Isabelle/HOL to formalise a new, machine-checked, proof of strong normalisation and cut-elimination for $\boldsymbol{\delta}\mathbf{RA}$ which does not use measures on the size of derivations. Our formalisation generalises easily to other display calculi and can serve as a basis for formalised proofs of strong normalisation for the classical and intuitionistic versions of a vast range of substructural logics like the Lambek calculus, linear logic, relevant logic, BCK-logic, and their modal extensions. We believe this is the first full formalisation of a strong normalisation result for a sequent system using a logical framework.

1 Introduction

Sequent calculi provide a rigorous basis for meta-theoretic studies of logics. The central theorem is cut-elimination which states that detours through lemmata can be avoided, and it can be used to show many important logical properties like consistency, interpolation, and Beth definability. Cut-free sequent calculi are also useful for automated deduction [14], nonclassical extensions of logic programming [22], and studying deep connections between cut elimination, lambda calculi and functional programming. Sequent calculi, and their extensions, therefore play an important role in theoretical computer science.

Display Logic [5] is a generalised sequent framework for non-classical logics. Since it is not really a logic, we prefer the term display calculi. Display calculi extend Gentzen's language of sequents with extra, complex, n-ary

[1] Supported by an Australian Research Council Large Grant
[2] Supported by an Australian Research Council QEII Fellowship
[3] `jeremy@discus.anu.edu.au`
[4] `rpg@discus.anu.edu.au`

structural connectives, in addition to Gentzen's sole structural connective, the "comma". Properties such as associativity are explicitly stated as structural rules. Display calculi provide an extremely general sequent framework encompassing a vast range of substructural logics like the bi-intuitionistic and bi-classical versions of the Lambek calculus, linear logic, relevant logic, BCK-logic, and their modal and tense logical extensions, in a uniform way [24,13]. The most remarkable property of display calculi is a generic cut-elimination theorem, which applies whenever the display calculus rules satisfy certain, easily checked, conditions given by Belnap [5]. Display calculi therefore provide an important generalisation of sequent calculi.

Traditional proofs of cut-elimination in sequent calculi do not eliminate the cut rule directly, due to problems in eliminating applications of contraction above cut. Gentzen first replaced the cut rule with the mix rule, showed that cut was derivable from mix, and then eliminated the mix rule. Borisavljevic *et al* [6] have shown that direct cut-elimination is by no means trivial, requiring a further detour through a special normal form for derivations. The problems caused by contraction can also be avoided by first proving that contraction itself is an admissible rule. We cannot use this strategy as the ability to include or omit explicit structural rules like contraction give display calculi, and the associated cut-elimination theorems, their modularity.

In [5], Belnap gives a direct proof that the cut rule is *admissible* in display calculi. That is, he considers a derivation that contains only one application of cut, at the bottom, and shows how to derive the same end-sequent without using cut. In separate work, we have also described a fully formal proof of cut-admissibility for the display calculus $\boldsymbol{\delta}\mathbf{RA}$ [9]. Given a derivation with multiple cuts, such a procedure repeatedly eliminates the top-most cut, and this procedure is sometimes known as weak-normalisation. A much harder task is to give an effective procedure to eliminate an arbitrary cut rather than just the top-most one, and such procedures are sometimes known as strong-normalisation. In [24], Wansing gives a direct proof of strong-normalisation for a particular display calculus. Wansing's proof of termination uses a complex measure on the size of derivations. When we tried to mechanise Wansing's proof, we discovered that it contained a serious gap.

We therefore developed and mechanised a new direct proof of strong normalisation for the display calculus $\boldsymbol{\delta}\mathbf{RA}$ using four complex orderings, which we proved to be well-founded. Our orderings do not depend upon the size of derivations. As far as we are aware, this is the first full formalisation of a strong normalisation result for a sequent system using a logical framework. The work required approximately 9 man-months, approximately 1600 lines of Isabelle/HOL theory files and 8500 lines of ML proof files, and takes approximately 35 minutes to fully check on a 300MHz Pentium machine.

Various authors have fully formalised proofs of strong normalisation for λ-calculi: see references in [2], and also [3]. But λ-calculi are typically restricted to intuitionistic logics, and their corresponding natural deduction calculi are usually single conclusioned. Sequent calculi, and particularly display calculi, are a much more general framework, catering to both intuitionistic and classical variants of many substructural logics and their modal and tense extensions [13]. The strong-normalisation results for sequent calculi are therefore more widely applicable. We are aware of only four other attempts to formalise proof theory or cut elimination in sequent calculi.

The first is the work of Pfenning which formalises the admissibility (weak-normalisation) of cut for classical, intuitionistic and linear logic using the logical framework Elf [21]. Pfenning readily admits this is not a full formalisation [21], but Pfenning's proofs have now been fully formally verified in Twelf by Schürmann; see [23, Section 8.5]. Both formalisations make explicit use of properties of the underlying logical framework to obtain certain structural rules like exchange, weakening and contraction "for free". The associated *weak normalisation* results are therefore not as modular as in display calculi since omitting one or more of these rules is then not possible (trivially).

The second is the work of Matthews *et al* [16,4,17,15] which implements Feferman's FS_0 in Isabelle, and uses it to formalise and extend various meta-theoretical systems. It should be possible to formalise weak normalisation for cut-elimination in this implementation since Matthews outlines how it might be done using "pencil and paper", and suggests it as a further project [15, §6,§9.2]: but it does not appear to have been done.

The third is the work of Mikhajlova and von Wright [18], giving a formalisation of a sequent calculus for first-order logic in the system HOL. This work does not involve cut-elimination or strong normalisation, and as we point out later, actually contains an unfortunate bug.

The fourth is the work of Adams [1] in which he formalises three cut-free sequent-style calculi in Coq, and proves a weak-normalisation result for derivations with respect to certain permutations.

The paper is set out as follows. In Section 2 we describe the display calculus $\boldsymbol{\delta}$**RA** for the logic of relation algebras, the logical framework Isabelle and its theory HOL, and our encoding of $\boldsymbol{\delta}$**RA** into Isabelle/HOL. In Section 3 we describe functions essential for reasoning about derivations. In Section 4 we describe Isabelle/HOL functions for reasoning about cuts, and discuss the problems we had in formalising Wansing's proof of strong normalisation. In Section 5 we describe our new proof of strong normalisation for $\boldsymbol{\delta}$**RA**. Section 6 presents further work. A longer version of this paper and actual Isabelle code can be found at `http://discus.anu.edu.au/~rpg/CutElim/` and `http://discus.anu.edu.au/~jeremy/deep-files/` respectively.

2 A Deep Embedding of δRA in Isabelle/HOL

The following grammar defines the syntax of relation algebras:

$$A\ B ::= p_i \mid \top \mid \bot \mid \neg A \mid A \wedge B \mid A \vee B \mid \mathbf{1} \mid \mathbf{0} \mid \smile A \mid A + B \mid A \circ B$$

A display calculus for relation algebras called $\boldsymbol{\delta}\mathbf{RA}$ can be found in [11]. Sequents of $\boldsymbol{\delta}\mathbf{RA}$ are expressions of the form $X \vdash Y$ where X and Y are built from the nullary structural constants E and I and formulae, using the structural connectives $\bullet$ and $*$ and $\;;\;$ and $\;,\;$ according to the grammar below:

$$X\ Y ::= A \mid I \mid E \mid *X \mid \bullet X \mid X \;;\; Y \mid X \;,\; Y$$

The defining feature of display calculi is that in all logical introduction rules, the principal formula is always "displayed" as the whole of the right-hand or left-hand side. For example, the rule ($\mathbf{LK}$-$\,\vdash \vee$), shown below left, is typical of Gentzen's sequent calculi like $\mathbf{LK}$, while the rule ($\boldsymbol{\delta}\mathbf{RA}$-$\,\vdash \vee$) shown on the right is typical of display calculi; see [11] for details:

$$\frac{\Gamma \vdash \Delta, P \;,\; Q}{\Gamma \vdash \Delta, P \vee Q}(\mathbf{LK}\text{-}\vdash \vee) \qquad\qquad \frac{X \vdash P \;,\; Q}{X \vdash P \vee Q}(\boldsymbol{\delta}\mathbf{RA}\text{-}\vdash \vee)$$

Isabelle is an automated proof assistant [20]. Its meta-logic is an intuitionistic typed higher-order logic, sufficient to support the built-in inference steps of higher-order unification and term rewriting. Isabelle accepts inference rules of the form "from $\alpha_1, \alpha_2, \ldots, \alpha_n$, infer β" where the α_i and β are expressions of the Isabelle meta-logic, or are expressions using a new syntax, defined by the user, for some "object logic". Most users build on one of the comprehensive "object logics" already supplied, like Isabelle/HOL [19], which is an Isabelle theory based on the higher-order logic of Church and the HOL system of Gordon [10]. In [8], we describe our initial attempts to formalise display calculi in various logical frameworks and describe why we selected Isabelle/HOL. We assume the reader is familiar with ML and logical frameworks in general.

Formulae, structures and sequents of $\boldsymbol{\delta}\mathbf{RA}$ are represented by the datatypes shown in Figure 1. The constructors `FV` and `SV` represent a formula or structure variable appearing in the statement of a rule or theorem, and which is instantiated to an actual formula or structure of $\boldsymbol{\delta}\mathbf{RA}$ when constructing derivations. The constructor `PP` represents a primitive proposition p_i, for which we can substitute only another primitive proposition. The operator `Structform` causes a formula to be "cast" into a structure. A sequent `(Sequent X Y)` can also be represented as `$X |- $Y` in which the `$` indicates that an atom is a structure rather than a formula. Thus the term `Sequent (SV ''X'') (Structform (FV ''A''))` is printed, and may be entered, as `($''X'' |- ''A'')`.

The notation in parentheses in Figure 1 describes an alternative infix syntax which is closer to the actual syntax of $\boldsymbol{\delta}\mathbf{RA}$. A different method is used to specify the infix syntax for structures and sequents: details omitted.

The premisses of a rule are represented as a list of sequents while the conclusion is a single sequent. So `Rule prems concl` means a rule with premisses `prems` and conclusion `concl`. Many single-premiss rules of display calculi are defined to be usable from top to bottom as well as from bottom to top: the two constants `Bidi` and `InvBidi` cater for this. Functions `premsRule` and `conclRule` return the premisses and the conclusion of a rule.

In contrast to the formula and structure variables modelled explicitly by `FV`

```
datatype formula =
    Btimes formula formula ("_ && _" [68,68] 67)
  | Rtimes formula formula ("_ oo _" [68,68] 67)
  | Bplus formula formula ("_ v _" [64,64] 63)
  | Rplus formula formula ("_ ++ _" [64,64] 63)
  | Bneg formula ("--_" [70] 70)
  | Rconv formula ("_^" [75] 75)
  | Btrue ("T")
  | Bfalse("F")
  | Rtrue ("r1")
  | Rfalse("r0")
  | FV string
  | PP string
```

```
datatype structr =
    Comma structr structr
  | SemiC structr structr
  | Star structr
  | Blob structr
  | I
  | E
  | Structform formula
  | SV string
```

```
datatype sequent =
    Sequent structr structr
```

```
datatype rule =
    Rule (sequent list) sequent
  | Bidi sequent sequent
  | InvBidi sequent sequent
```

Fig. 1. Formulae, Structures, Sequents and Rules of $\boldsymbol{\delta}$**RA** in Isabelle/HOL.

and `SV`, Isabelle has "scheme variables", identified by a preceding '`?`'. When a scheme variable appears in an Isabelle theorem, the theorem is true when anything, of the appropriate type, is substituted for that variable.

The constant `rls` represents the set of rules of $\boldsymbol{\delta}$**RA** and `cutr` is the name of the cut rule, encoded using the datatypes just described. For example, the rule ($\boldsymbol{\delta}$**RA**-$\vdash\vee$) shown above is represented and printed as the rule `ora`:

```
ora == Rule [(''A'' |- $''X''), (''B'' |- $''Y'')]
             (''A'' v ''B'' |- $''X'',, $''Y'')
```

The actual rules `rls` are not critical to this paper, since we believe our results will apply for any display calculus satisfying Belnap's conditions (see §6).

We represent rules explicitly using a datatype `rule` and use explicit constructors `SV` and `FV` for structure and formula variables because we need to express and check that the rules satisfy (some of) Belnaps' conditions. For example, Belnap's condition C3 amounts to "a structure variable cannot appear more than once in the conclusion of a rule". This condition could not be checked, in Isabelle itself, if we had installed the $\boldsymbol{\delta}$**RA** rules using Isabelle's rule, variable and substitution features. We therefore also had to handle substitutions for our SV and FV variables explicitly, via various functions. Although this approach made our task more difficult, it was necessary so that we could structure the proofs of both weak [9] and strong normalisation in terms of Belnap's conditions. Such aspects of "deep" versus "shallow" embeddings are also discussed in [8].

We use the term "derivation" to refer to a proof *within* the sequent calculus, reserving the term "proof" for a meta-theoretic proof of a theorem *about* the sequent calculus. We model a derivation tree using the following datatype:

```
datatype dertree = Der sequent rule (dertree list)
                 | Unf sequent
```

In a term `Der seq rule dts` the subterm `seq` is the sequent at the root (bottom) of the tree, and `rule` is the rule used in the last (bottom) inference. Thus, if the tree represents a real derivation, then sequent `seq` will be an instantiation of the conclusion of `rule`, and the corresponding instantiations of the premisses of `rule` will be the roots of the trees in the list `dts`. The trees in `dts` are called the *immediate* subtrees of `Der seq rule dts`. The leaves of a derivation tree are either initial sequents with no premisses, or "`Unf`inished" sequents whose derivations are currently unfinished.

Display calculi use the initial sequent $p \vdash p$, using primitive propositions only. It is then proved that the sequent $A \vdash A$ is derivable for all formulae A by induction on the size of A, where A *stands for* a formula composed of primitive propositions and logical connectives. We proved this as the theorem `idfpp` and therefore added a new (derived) rule called `idf` with shape $A \vdash A$. We also need to reason about the derivation trees of derived rules which may use the (derived) rule $A \vdash A$, for arbitrary formula A. Therefore the derivation tree `Der (''A'' |- ''A'') idf []` stands for a complete derivation which uses the lemma that $A \vdash A$ is derivable, while `Unf (''A'' |- ''A'')` stands for an incomplete derivation with unfinished premiss $A \vdash A$.

For example, the incomplete derivation tree shown below at left is represented as the Isabelle/HOL term shown below at right where `''A'' |- PP ''p'' && ''A''` stands for $A \vdash p \wedge A$ and `cA` and `ands` are the contraction and $(\vdash \wedge)$ rules, and `idf` is the derived rule $A \vdash A$:

$$\dfrac{\dfrac{A \vdash p \quad A \vdash A}{A, A \vdash p \wedge A}(\vdash \wedge)}{A \vdash p \wedge A}(ctr)$$

```
Der (''A'' |- PP ''p'' && ''A'') cA
 [Der (''A'', ''A'' |- PP ''p'' && ''A'') ands
 [Unf (''A'' |- PP ''p''),
  Der (''A'' |- ''A'') idf []]]
```

3 Reasoning About Derivations and Derivability

We now describe various functions which allow us to reason about derivations in $\boldsymbol{\delta}\mathbf{RA}$ and theorems we have proved about derivability in $\boldsymbol{\delta}\mathbf{RA}$.

`allDT f dt` holds if property `f` holds for every sub-tree in the tree `dt`.

`allNextDTs f dt` holds if property `f` holds for every proper sub-tree of `dt`.

`wfb (Der concl rule dts)` holds if sequent rule `rule` has an instantiation with conclusion instance `concl` and premiss instances which are the conclusions of the derivation trees in list `dts`. Such a node is *well-formed*.

`allDT wfb dt` holds if every sub-tree of the tree `dt` is *well-formed*. That is, if every node in `dt` is well-formed. Such a derivation is said to be *well-formed*.

`frb rules (Der concl rule dts)` holds when the lowest rule `rule` used in a derivation tree `Der concl rule dts` belongs to the set `rules`.

`allDT (frb rules) dt` holds when every rule used in a derivation tree `dt` belongs to the set `rules`.

`premsDT dt` returns a list of all "premisses" (unfinished leaves) of the derivation tree `dt` (ie, the sequents found in nodes of `dt` of the form `Unf seq`).

`conclDT dt` returns the end-sequent of the derivation tree `dt`. That is, the conclusion of the bottom-most rule instance.

Definition 3.1 A derivation tree `dt` is *valid* if it is well-formed, uses rules in the set of rules `rules` of the calculus, and has no unfinished leaves.

```
valid_def = "valid ?rules ?dt ==
   allDT wfb ?dt & allDT (frb ?rules) ?dt & premsDT ?dt = []"
```

The question marks in front of `rules` and `dt` flag that they are Isabelle scheme variables, even though the question marks would be absent in the Isabelle/HOL theory file itself: we do this throughout this paper.

Definition 3.2 [IsDerivableR] `IsDerivableR rules prems' concl` holds iff there exists a derivation tree `dt` which uses only rules from set `rules`, is well-formed, has conclusion `concl`, and has premisses from set `prems'`.

```
"IsDerivableR ?rules ?prems' ?concl == (EX dt.
      allDT (frb ?rules) dt & allDT wfb dt &
      conclDT dt = ?concl & set (premsDT dt) <= ?prems')"
```

Here, `set` is a function that allows us to treat its argument as a set rather than a list, and `<=` is the subset relation $\subseteq$. Our main result about derivability is a recursive characterisation of derivability.

Theorem 3.3 (IsDerivableR_recur) *A conclusion* `concl` *is derivable from premisses* `prems` *using rules* `rules` *iff either* `concl` *is one of* `prems`*, or there exists an instantiated* `rule` *obtained from* `rules` *and the conclusion of* `rule` *is* `concl` *and the premisses of* `rule` *are derivable from* `prems` *using* `rules`*.*

```
IsDerivableR_recur = "IsDerivableR ?rules ?prems ?concl = (
 ?concl : ?prems | (EX rule. ruleFromSet rule ?rules &
 conclRule rule = ?concl & (ALL p : set (premsRule rule).
 IsDerivableR ?rules ?prems p)) )" : thm
```

The appellation "`: thm`" indicates a statement that has been proved in Isabelle/HOL using previous Isabelle/HOL definitions and theorems: we follow this practice for theorems and lemmata throughout this paper.

In the third clause for `IsDerivable` on [18, page 302], which deals with the case of a result being provable using derived rules, an incorrect use of an existential quantifier gives that $P \to Q$ could be used as a derived rule on the grounds that one instance of it, say $\mathit{True} \to \mathit{True}$, is provable.

4 Reasoning About Cuts and Wansing's Proof

We assume familiarity with notions like "parametric ancestors" of a cut formula from [5], "principal moves" and "parametric moves" from the literature on strong normalisation: details can be found in the Appendix (Section 7).

Definition 4.1 A cut rule application is *left-* [*right-*] principal if the cut-formula is the principal formula of the left [right] premiss of the cut rule.

Each of the following functions requires the bottom node of the derivation tree to be of the form `Der seq rule dts`, and that if `rule` is (*cut*), then for: `cutOnFmls s` the cut is on a formula in the set `s`; `cutIsLP A` the cut is on formula `A` and is left-principal; `cutIsLRP A` the cut is on formula `A` and is (left- and right-)principal. It follows from the definitions that a derivation tree satisfying any of `allDT (cutOnFmls s)`, `allDT (cutIsLP A)` and `allDT (cutIsLRP A)` has no unfinished leaves. We omit details.

The ideas inherent in principal and parametric moves are well-known, but they were first applied to prove strong normalisation for a display calculus by Wansing to show that any (sufficiently long) sequence of steps in the process of cut-elimination terminates. This work is reproduced in [24, §4.3].

When we tried to machine-check Wansing's proof, we found a serious gap. Basically, Wansing assumes that the subtree $\Pi'[Y]$ and its conclusion $Z[Y] \vdash A$ in the right (transformed) tree of Figure 4 in the Appendix is the same as the tree $\Pi[A]$ and its conclusion $Z[A] \vdash A$ in the left (original) tree: that is, the transformation process does not alter Π but leaves it intact. In fact, this is true only if Z contains no parametric ancestors of A, and there are examples which use contraction where this fails: see the long version of our paper.

5 A New Proof of Strong Normalisation

We now give a new direct proof of strong normalisation which does not rely on the size of derivation trees. Given a derivation tree with (*cut*) at its root, the tree can be changed to deal with that particular cut; we call these "cut-reductions". Following Wansing [24, §4.2], we classify these cut-reductions as principal or parametric. More generally, we can change a tree by performing a cut-reduction on any subtree; we call such changes *reductions*.

5.1 Defining Strongly Normalisable Derivations

Definition 5.1 [reduces] Assuming an irreflexive relation cutReduces (defined later), derivation tree Π_0 reduces to derivation tree Π_1 if either (a) Π_0 cutReduces to Π_1, or (b) Π_0 and Π_1 are identical except that exactly one immediate subtree Π_0 reduces to the corresponding immediate subtree of Π_1.

```
reduces_Unf = "reduces (Unf ?seq) ?dtn = False"
reduces_Der  = "reduces (Der ?seq ?rule ?dtl) ?dtn =
 ( (EX dtl'. onereduces ?dtl dtl' & ?dtn = Der ?seq ?rule dtl')
   | cutReduces (Der ?seq ?rule ?dtl) ?dtn )"

onereduces_Nil  = "onereduces [] ?dtl' = False"
onereduces_Cons = "onereduces (?h # ?t) ?dtl' = ( ?dtl' ~= []
  & (reduces ?h (hd ?dtl') & ?t = (tl ?dtl')
     | ?h = (hd ?dtl') & onereduces ?t (tl ?dtl')) )"
```

Wansing [24, p. 52] defines a strongly normalisable derivation tree as a tree from which every sequence of reductions is finite. We define inductively the set of strongly normalisable derivation trees using Isabelle's inductive definition feature [19, § 2.10], which defines the minimal set closed under the given rules. For example, the rules $\{0 \in \mathcal{S}, n \in \mathcal{S} \Longrightarrow n+2 \in \mathcal{S}\}$ define $\mathcal{S}$ to be the set of even naturals, although the set of all naturals also satisfies the rules.

Definition 5.2 [sn_set, strongly normalisable] The set sn_set is the smallest set of derivation trees satisfying: if every tree Π_1 to which Π_0 reduces is in sn_set then $\Pi_0 \in$ sn_set. A derivation tree is *strongly normalisable* iff it is a member of sn_set.

```
inductive "sn_set"
  intrs
    snI "(ALL dtn.
      reduces ?dt dtn --> dtn : sn_set) ==> ?dt : sn_set"
```

Note that cutReduces is irreflexive, so if Π_0 cannot be reduced at all, then it follows that $\Pi_0 \in$ sn_set.

5.2 Various Well-Founded Orderings

To prove that every derivation tree is strongly normalisable, we use a binary relation dtorder on derivation trees, and show that it is well-founded. First we need an auxiliary definition.

Definition 5.3 [sn1red] For two lists dtl1 and dtl2 of derivation trees, sn1red dtl1 dtl2 holds iff the lists dtl1 and dtl2 are non-empty and differ at only one position where dtl1 contains tree dt1 and dtl2 contains tree dt2, and dt1 reduces to dt2, and dt1 is strongly normalisable.

```
sn1red_Nil  = "sn1red [] ?dtl2 = False"
```

```
sn1red_Cons = "sn1red (?h # ?t) ?dtl2 = (?dtl2 ~= [] &
       (reduces ?h (hd ?dtl2) & ?h : sn_set & ?t = (tl ?dtl2)
       | ?h = (hd ?dtl2) & sn1red ?t (tl ?dtl2)) )"
```

Definition 5.4 [`LRPorder`, `cutorder`, `sn1order`, `dtorder`] We define four binary relations, `LRPorder`, `cutorder`, `sn1order` and `dtorder` on derivation trees as sets of ordered pairs (but we omit the corresponding Isabelle code):

(a) $\Pi_1 <_{\texttt{LRP}} \Pi_0$ if the bottom inferences of derivations Π_1 and Π_0 are both (*cut*), and either
  (i) the cut in Π_1 is left-principal or right-principal, and the cut in Π_0 is neither, or
  (ii) the cut in Π_1 is (left- and right-)principal, and the cut in Π_0 is not (left- and right-)principal.

(b) $\Pi_1 <_{\texttt{cut}} \Pi_0$ if the bottom inferences of derivations Π_1 and Π_0 are both (*cut*), and if either
  (i) the size of the cut-formula of Π_1 is smaller than that of Π_0, or
  (ii) each derivation has the same cut-formula, and $\Pi_1 <_{\texttt{LRP}} \Pi_0$.

(c) $\Pi_1 <_{\texttt{sn1}} \Pi_0$ if Π_0 and Π_1 are the same except that one of the immediate subtrees of Π_0 is strongly normalisable and reduces to the corresponding immediate subtree of Π_1.

(d) $\Pi_1 <_{\texttt{dt}} \Pi_0$ iff any of the following hold:
  (i) the bottom inference of Π_1 is not (*cut*), but that of Π_0 is
  (ii) $\Pi_1 <_{\texttt{cut}} \Pi_0$
  (iii) $\Pi_1 <_{\texttt{sn1}} \Pi_0$.

Theorem 5.5 (`wf_LRPorder, wf_cutorder, wf_sn1order`) *The relations `LRPorder`, `cutorder` and `sn1order` are well-founded.*

Despite the notation, these relations are *not* reflexive, and some are *not* transitive. Intuitively, $(\Pi_1, \Pi_0) \in$ `dtorder` means that Π_1 is closer to being cut-free (in some sense) than is Π_0. To prove that `dtorder` is well-founded, we first need a result on the interaction between `cutorder` and `sn1order`.

Lemma 5.6 (`sntr'`) *If* $\Pi_2 <_{cut} \Pi_1$ *and* $\Pi_1 <_{sn1} \Pi_0$ *then* $\Pi_2 <_{cut} \Pi_0$.

```
sntr' = "[| (?dty, ?dtza) : cutorder; (?dtza, ?dtz) : sn1order |]
          ==> (?dty, ?dtz) : cutorder" : thm
```

Theorem 5.7 (`wf_dtorder`) *`dtorder` is well-founded.*

5.3 Inheriting Strong Normalisation

We next define `snHered`, a property of derivation trees which indicates that a tree inherits strong normalisation from its immediate subtrees.

Definition 5.8 [`snHered`] Π satisfies `snHered` iff: if all the immediate subtrees of Π are strongly normalisable then Π is strongly normalisable.

```
snHered_def = "snHered ?dt ==
  set (nextUp ?dt) <= sn_set --> ?dt : sn_set"
```

Lemma 5.9 (hereds_sn) *A derivation tree* Π *is strongly normalisable iff every subtree of* Π *has the property* `snHered`.

```
hereds_sn = "(ALL dts.
  isSubt ?dt dts --> snHered dts) = (?dt : sn_set)" : thm
```

Proof. The "only if" part uses the theorem `sn_set.induct` below which is generated automatically by Isabelle from the inductive definition of `sn_set`.

```
sn_set.induct = "[| ?xa : sn_set ; (!!dt. (ALL dtn.
    reduces dt dtn --> dtn : sn_set & ?P dtn) ==> ?P dt ) |]
 ==> ?P ?xa" : thm
```

The "if" part uses induction on the height of Π using the assumption that every subtree of Π, including itself, has the property `snHered`. ☐

5.4 Reasoning About Cut-Reductions

We intend to define cut-reduction in a way that enables us to make some assertions about cut-reductions. We first define properties `nparRedP` and `c8redP` of reductions (which in fact apply to parametric and principal reductions respectively). The definitions do not require that the bottom inference of the derivation be (*cut*), but they are used only where this is so.

Definition 5.10 [`nparRedP`] The relation `nparRedP` holds between two derivation trees Π_0 and Π_1 if every subtree Π_1^s of Π_1 with a bottom inference (*cut*) satisfies either (a) Π_1^s is a proper subtree of Π_0, or (b) Π_1^s and Π_0 have the same cut-formula and $\Pi_1^s <_{\texttt{LRP}} \Pi_0$.

```
nparRedP_Unf = nparRedP "(Unf ?seq) ?dtn = False"

nparRedP_Der  = "nparRedP (Der ?seq ?rule ?dtl) ?dtn = (ALL dts.
 isSubt ?dtn dts & isCut dts -->
    isSubt (Der ?seq ?rule ?dtl) dts & Der ?seq ?rule ?dtl ~= dts
  | cutForm (Der ?seq ?rule ?dtl) = cutForm dts &
    (dts, Der ?seq ?rule ?dtl) : LRPorder)"
```

Definition 5.11 [`c8redP, c8redsfP`] A reduction from Π_0 to Π_1 satisfies `c8redP` if the lowest rule of Π_0 is (*cut*), and for each subtree Π_1^s of Π_1 whose lowest rule is (*cut*), either (a) Π_1^s is a proper subtree of Π_0, or (b) the cut-formula of Π_1^s is smaller than the cut-formula of the lowest rule of Π_0.

`c8redsfP` is defined similarly, but (b) reading instead "the cut-formula of Π_1^s is a proper subformula of the cut-formula of the lowest rule of Π_0"

```
c8redP = "c8redP ?dt ?dtn == ALL dts.
  isSubt ?dtn dts & botRule dts = cutr -->
```

```
isSubt ?dt dts & ?dt ~= dts |
size (cutForm dts) < size (cutForm ?dt)"
```

Note that both Definition 5.11(b) and Definition 5.10(b) imply $(\Pi_1^s, \Pi_0) \in$ `dtorder`, and that `c8redsfP` implies `c8redP`. We now define a *cut-reduction* (being either parametric or principal) as satisfying one of the properties `nparRedP` and `c8redP`, as well as some further simple conditions which help the proof.

Definition 5.12 [`cutReduces`] The derivation tree Π_0 *cut-reduces* to Π_1 if the following hold simultaneously: Π_0 and Π_1 satisfy either `nparRedP` (for a parametric reduction) or `c8redP` (for a principal reduction); Π_0 and Π_1 have the same conclusion; the bottom rule of Π_0 is (*cut*); Π_0 and Π_1 are not identical; Π_1 does not consist solely of an unfinished leaf; and Π_0 has at least one immediate subtree.

```
cutReduces_Der = "cutReduces (Der ?seq ?rule ?dtl) ?dtn = (
 (nparRedP (Der ?seq ?rule ?dtl) ?dtn
   | c8redP (Der ?seq ?rule ?dtl) ?dtn)
  & conclDT ?dtn = ?seq & ?rule = cutr
  & (Der ?seq ?rule ?dtl) ~= ?dtn & ~ isUnf ?dtn & ?dtl ~= [] )"
```

Note that for the purposes of the proof of strong normalisation, we have defined cut-reduction weakly in that, for example, we do not require that the new derivation tree `dtn` be well-formed (via `allDT wfb dtn`) or require that it use rules which belong to the calculus (via `allDT (frb rls) dtn`). However the definition is also strong in that it requires that either `nparRedP` or `c8redP` holds. Later we will show that the reductions in which we are interested do satisfy `nparRedP` or `c8redP`. The result of this is that we prove strong normalisation for a class of reductions which is larger than is really necessary. The requirement that Π_0 and Π_1 be distinct excludes null reductions.

5.5 Strong-Normalisation

Lemma 5.13 (`dth`) *For a given derivation Π_0, if all derivation trees $\Pi' <_{dt} \Pi_0$ have the property* ***`snHered`****, then so does Π_0.*

```
dth = "ALL dt'. (dt', ?dt) : dtorder --> snHered dt'
       ==> snHered ?dt" : thm
```

Proof. The machine-proof is quite complicated, and a careful examination of it highlights why the definition of *dtorder* needs to be so complex. □

Theorem 5.14 (`all_sn`) *Every derivation tree is strongly normalisable.*

```
all_snH = "snHered ?dt"    : thm
all_sn  = "strongNorm ?dt" : thm
```

Proof. By well-founded induction, it follows from Lemma 5.13 that every derivation tree satisfies `snHered`; the result follows from Lemma 5.9. □

At this point we have actually shown using Isabelle that a class of reductions which we have defined is strongly normalising. We need to show that this class of reductions contains the ones in which we are interested.

5.6 Making A Cut (Left) Principal

The following are the main theorems used to develop the mechanised proof based on making cuts (left and right) principal.

We use a functional definition `mLP dtAY seqY dtA` of the transformation used to make a cut left-principal. The arguments to `mLP` are: a derivation tree `dtAY` with root $A \vdash Y$ such as the right subtree Π_R of the left tree given in Figure 4 in the Appendix; a sequent `seqY`; and a derivation tree `dtA` with root `seqA` such as the tree with root $X \vdash A$ which forms the left subtree of the left tree given in Figure 4. Sequent `seqA` will usually contain occurrences of A in one or more succedent position(s), and sequent `seqY` is `seqA` with zero, one or more of these occurrences of A changed to Y. The result of `mLP` is a new derivation tree, with root `seqY` whose new cuts (on A) are left-principal, such as the right tree given in Figure 4. The definition of `mRP` is similar.

We also defined a function `ncLP seqY dtA` which returns true iff calculating the tree `mLP dtAY seqY dtA` does not involve traversing another cut; `ncRP` has analogous meaning. The result stated by Wansing requires that the transformation used to make a cut left-principal be allowed only when this condition holds. We need to show that the transformation performed by `mLP` (subject of this condition) are in the class of reductions that are strongly normalising. We assume an initial cut as in Figure 4: that is, with conclusion $X \vdash Y$ and cut-formula A.

Theorem 5.15 (pRedLP2) *Consider a parametric reduction of a cut which proceeds by transforming the left subtree (to change its conclusion from $X[A] \vdash A$ to $X[Y] \vdash Y$), using the function* `mLP`*. Assume that the subtree satisfies the condition* `ncLP` *(in effect, that the transformation can be performed without traversing another cut). Also assume the cut is not already left-principal. Then the reduction satisfies the condition* `nparRedP`*.*

```
pRedLP2 = "[| ?dtY = mLP ?dtAY ?seqY ?dtA;
      ?dt = Der ?seqY cutr [?dtA, ?dtAY];
      ~ rootIsSucP ?dtA; valid rls ?dt;
      ncLP ?seqY ?dtA |]
   ==> nparRedP ?dt ?dtY & valid rls ?dtY &
       conclDT ?dtY = conclDT ?dt" : thm
```

The condition ∼`rootIsSucP ?dtA` means that the cut is not already left-principal. It is required to avoid null reductions: if the cut is already left-principal, then `?dt` and `mLP ?dtAY ?seqY ?dtA` are equal. The following result is similar, but says that the parametric reduction is a cut-reduction.

Theorem 5.16 (pRedLP3) *The same as Theorem 5.15, but with conclusion* cutReduces ?dt ?dtY & valid rls ?dtY"

Analogous theorems pRedRP2, pRedRP3 guarantee that we can make a cut right-principal, and consequently another theorem which guarantees that we can make a cut (left and right) principal. The associated functions also produce derivations that satisfy cutReduces. Thus every parametric move corresponds to a reduction in the class of strongly normalising reductions.

5.7 Dealing With Principal Cuts

For each logical connective and constant in the calculus, we prove a result like the following one for $\vee$.

Theorem 5.17 (orvC8W) *Given a valid derivation tree* **dt***, assume that if the bottom rule of* **dt** *is (*cut*), then the cut is principal and its cut-formula is* $A \vee B$*. Then then there is a valid derivation tree* **dtn** *with the same conclusion as* **dt** *such that* **dt** *and* **dtn** *satisfy* **c8resfP** *(since the original cut on* $A \vee B$ *is replaced with new cuts on its proper subformulae* A *and* B*).*

```
orvC8W = "[| cutIsLRP (?A v ?B) ?dt; valid rls ?dt |]
  ==> EX dtn. conclDT dtn = conclDT ?dt &
    c8redsfP ?dt dtn & valid rls dtn"
```

These principal reductions satisfy c8redsfP and hence c8redP, and are in fact cut-reductions. These results (one for each logical connective) were combined to give Theorem 5.18.

Theorem 5.18 (vformC8W') *If the bottom-most rule of a valid derivation tree* Π_0 *is a (left and right) principal cut, then there exists a valid derivation tree* Π_1 *with the same conclusion as* Π_0*, such that* Π_0 **cutReduces** *to* Π_1*.*

```
vformC8W' = "[| ?dt = Der ?seq cutr ?dtl;  valid rls ?dt ;
       cutIsLRP ?form ?dt |]
   ==> EX dtn. cutReduces ?dt dtn & valid rls dtn" : thm
```

Thus every principal move gives a reduction which is in the class of strongly normalising reductions.

5.8 Cut-Elimination Via Strong Normalisation

Recall that a *valid* derivation tree is one which is well-formed, uses rules from rls, and has no unfinished leaves.

Definition 5.19 [validRed] A *valid reduction* is a reduction of tree Π_0 to tree Π_1, where Π_1 is a valid tree which has the same conclusion as Π_0.

```
validRed_def "validRed == {(dtn, dt).
    conclDT dtn = conclDT dt & valid rls dtn & reduces dt dtn}"
```

Lemma 5.20 *A valid reduction of a subtree is a valid reduction of the whole tree.*

Theorem 5.21 *Parametric and principal moves give valid reductions.*

Proof. Using `pRedLP3`, `pRedRP3` and `vformC8W'`. □

Theorem 5.22 (`cutExRed`) *For any valid tree with a single cut at the bottom there exists a cut-reduction to another valid tree with the same conclusion (using only parametric and principal moves).*

```
cutExRed = "[| isCut ?dt; valid rls ?dt;
       allNextDTs (cutOnFmls {}) ?dt |]
   ==> EX dtn. cutReduces ?dt dtn & valid rls dtn" : thm
```

Proof. We use only `pRedLP3` and `pRedRP3` and `vformC8W'`, so that the reductions are restricted to parametric and principal moves. Note that `conclDT dtn = conclDT dt` is implied by `cutReduces dt dtn`. □

Theorem 5.23 (`ExRed`) *For any valid tree which contains a cut, there is available at least one valid reduction (apply Theorem 5.22 to a top-most cut).*

```
ExRed = "[| valid rls ?dt ; ~ allDT (Not o isCut) ?dt |]
   ==> EX dtn. (dtn, ?dt) : validRed" : thm
```

From here, we repeatedly perform valid reductions. The notation `^*` denotes transitive closure, and `allDT (Not o isCut) dtn` means that `dtn` is cut-free since `o` denotes composition of functions in Isabelle/HOL. Finally, Theorem 5.27 states our result in terms of cut-admissibility.

Theorem 5.24 (`validRed_min`) *For any derivation tree* Π *there exists a tree* Π^r*, obtained from* Π *by repeated valid reductions, such that* Π^r *cannot be further validly reduced.*

```
validRed_min = "EX dtn. (dtn, ?dt1) : validRed^* &
                ~ (EX dts. (dts, dtn) : validRed)" : thm
```

Proof. Every tree with a cut admits at least one valid reduction, and there is no infinite sequence of valid reductions. So repeatedly perform any sequence of (principal and parametric) reductions until no reduction is possible. □

Theorem 5.25 (`redNoCut`) *For any valid tree* Π*, there exists a cut-free valid tree* Π^r*, obtained from* Π *by repeated valid reductions, such that* Π^r *has the same conclusion as* Π*.*

```
redNoCut = "valid rls ?dt ==> EX dtn.
   allDT (Not o isCut) dtn & valid rls dtn &
   (dtn, ?dt) : validRed^* & conclDT dtn = conclDT ?dt" : thm
```

Corollary 5.26 *Any sequence of principal and parametric moves starting from some derivation* Π *containing cuts, eventually terminates with a cut-free* *derivation* Π^r *that has the same conclusion as* Π.

Proof. In Theorem 5.25, starting from Π, choose a sequence of parametric and principal reductions only. Since valid reductions are reductions, they are strongly normalising by Theorem 5.14, so this sequence terminates with some valid derivation Π^r from which there is no reduction: that is, Π^r is cut-free.□

Theorem 5.27 (`cutElim_SN`) *If a sequent can be derived using rules* `rls`*, then it can be derived from those rules omitting (*cut*).*

```
cutElim_SN = "IsDerivableR rls {} ?concl ==>
              IsDerivableR (rls - {cutr}) {} ?concl" : thm
```

6 Further Work

Belnap's theorem applies to any Display Calculus satisfying his conditions. To prove his theorem in that form would require modelling an arbitrary Display Calculus, with generalised rules for arbitrary sets of structural and logical connectives. A theoretical framework for such generalised rules can be found in [12]. This would require a "deeper" embedding still. For, in our first implementation [7], we set up the specific connectives and rules of $\boldsymbol{\delta}$**RA** in Isabelle, and used Isabelle proofs as the $\boldsymbol{\delta}$**RA**-derivations. In the present implementation, we again set up the specific connectives and rules of $\boldsymbol{\delta}$**RA**, although we set up data structures to model arbitrary derivations. We believe our proofs used only Belnap's conditions on the rules of $\boldsymbol{\delta}$**RA**, and that it would be straight forward to adapt them to any display calculus satisfying those conditions, but this was not proved formally. To prove the generic Belnap theorem, we would need to set up the necessary structures to model arbitrary sets of connectives and rules from [12].

References

[1] Adams, A. A Formalisation of Weak Normalisation (with respect to Permutations) of Sequent Calculus Proofs. Journal of Computation and Mathematics 3 (2000), 1–26.

[2] Altenkirch, T. A formalization of the strong normalization proof for System F in LEGO. In TLCA-93, LNCS 664:13–28. Springer, 1993.

[3] Barras, B and B Werner. Coq in Coq. `http://pauillac.inria.fr/~barras/download/coqincoq.ps.gz`

[4] Basin, D A and S Matthews. *Structuring metatheory on inductive definitions.* Information and Computation, **162** (2000), 80–95.

[5] Belnap, N D. Display logic. *J. of Philosophical Logic*, 11:375–417, 1982.

[6] Borisavljevic, M, K Dosen, and Z Petric. *On permuting cut with contraction.* Mathematical Structures in Computer Science, **10** (2000), 99–136.

[7] Dawson, J E and R Goré. A mechanised proof system for relation algebra using display logic. In *Proc. JELIA98*, LNAI 1489:264-278. Springer, 1998.

[8] Dawson, J E and R Goré. *Embedding display calculi into logical frameworks: Comparing Twelf and Isabelle.* ENTCS **42** (2001), 89–103. 2001. Elsevier.

[9] Dawson, J E and R Goré. Formalised Cut Admissibility for Display Logic. In Proc. TPHOLS'02, LNCS 2410, 131–147, Springer, 2002.

[10] Gordon, M J C, and T F Melham. *Introduction to HOL: a Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[11] Goré, R. Cut-free display calculi for relation algebras. In Proc. CSL96, LNCS 1258, 198–210, Springer, 1997.

[12] Goré, R. Gaggles, Gentzen and Galois: How to display your favourite substructural logic. Logic Journal of the IGPL **6** (1998), 669–694.

[13] Goré, R. Substructural logics on display. LJIGPL **6** (1998), 451–504.

[14] Heuerding, A. *Sequent Calculi for Proof Search in some Modal Logics.* PhD thesis, Inst. for Applied Mathematics and Computer Science, Berne, 1998.

[15] Matthews, S. A Theory and its Metatheory in FS_0. In Dov M Gabbay (Ed), *What is a logic system?*, 329–354. Oxford University Press, 1994.

[16] Matthews, S, A Smaill, and D Basin. Experience with FS_0 as a framework theory. In G Huet and G Plotkin (Eds), *Logical Environments*, 61–82. CUP, 1993.

[17] Matthews, S. Implementing FS_0 in Isabelle: adding structure at the metalevel. In J Calmet and C Limongelli (Eds), *Proc. Disco'96.* Springer, 1996.

[18] Mikhajlova, A and J von Wright. Proving isomorphism of first-order proof systems in HOL. In TPHOLs98, LNCS 1479:295–314. Springer, 1998.

[19] Nipkow, T, L C Paulson, and M Wenzel. Isabelle's logics: HOL. Technical report. 15 February 2001, `doc/logics-HOL.dvi` in the Isabelle distribution.

[20] Paulson, L C. The Isabelle reference manual. Technical report. 15 February 2001, `doc/ref.dvi` in the Isabelle distribution.

[21] Pfenning, F. Structural cut elimination. In *Proc. LICS 94*, 1994.

[22] Pym, D and J Harland. *A Uniform Proof-theoretic Investigation of Linear Logic Programming.* J. of Logic and Computation 4 (1994), 175–207.

[23] Schürmann, C. *Automating the Meta Theory of Deductive Systems.* PhD thesis, Dept. of Comp. Sci. , Carnegie Mellon University, USA, CMU-CS-00-146, 2000.

[24] Wansing, H. "Displaying Modal Logic", volume 3 of *Trends in Logic.* Kluwer Academic Publishers, Dordrecht, August 1998.

$$\dfrac{\dfrac{\dfrac{\Pi_{ZAB}}{Z \vdash A, B}}{Z \vdash A \vee B}\,(\vdash \vee) \qquad \dfrac{\dfrac{\Pi_{AX}}{A \vdash X} \quad \dfrac{\Pi_{BY}}{B \vdash Y}}{A \vee B \vdash X, Y}\,(\vee \vdash)}{Z \vdash X, Y}\,(cut)$$

Fig. 2. Principal cut on formula $A \vee B$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_{ZAB}}{Z \vdash A, B}}{*A, Z \vdash B}\,(cs1) \qquad \dfrac{\Pi_{BY}}{B \vdash Y}}{*A, Z \vdash Y}\,(cut)}{Z \vdash A, Y}\,(\overline{cs1})}{Z, *Y \vdash A}\,(cs2) \qquad \dfrac{\Pi_{AX}}{A \vdash X}}{\dfrac{Z, *Y \vdash X}{Z \vdash X, Y}\,(\overline{cs2})}\,(cut)$$

Fig. 3. Transformed principal cut on formula $A \vee B$

7 Appendix: An Operational View of Cut-Elimination

In this section we give an operational view of cut-elimination, to give some intuition of what is involved in the overall cut-elimination procedure a là Belnap [5]. We assume familiarity with notions like "parametric ancestors" of a cut formula from [5].

Definition 7.1 An application of the cut rule is *left-principal* [*right-principal*] if the cut-formula is the principal formula of the left [right] premiss of the cut rule.

Belnap's condition C8 guarantees that all principal cuts can be replaced by new cuts on strictly smaller formulae. For example, the principal cut on $A \vee B$ shown in Figure 2 can be replaced by the derivation shown in Figure 3. The replacement derivation contains new cuts only on A and B, which are smaller formulae than $A \vee B$. Where the cut-formula is a single formula variable, it is necessarily introduced by the identity axiom: such cuts can be removed trivially.

The transformation of a pricipal cut on A into one or more cuts on proper subformulae of A is known as a "principal move". We now need a way to turn arbitrary cuts into principal ones.

In the case of a cut that is not left-principal, say we have a tree like the one on the left in Figure 4. Then we transform the subtree rooted at $X \vdash A$ by simply changing its root sequent to $X \vdash Y$, and proceeding upwards, changing all ancestor occurrences of A to Y. In doing this we run into difficulty at each

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Pi[A]}{Z[A] \vdash A}\,(\text{intro-}A)}{\Pi_L[A]}\,(\pi)}{X \vdash A}\,(\rho) \qquad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}\,(cut)
\qquad\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\Pi'[Y]}{Z[Y] \vdash A}\,(\text{intro-}A) \qquad \dfrac{\Pi_R}{A \vdash Y}}{Z[Y] \vdash Y}\,(cut)}{\Pi_L[Y]}\,(\pi)}{X \vdash Y}\,(\rho)
$$

Fig. 4. Making a cut left-principal

point where A is introduced: at such points we insert an instance of the cut rule. The diagram on the right hand side of Figure 4 shows this in the case where A is introduced at just one point.

In Figure 4, the notation $\Pi_L[A]$ and $Z[A]$ means that the sub-derivation Π_L and structure Z may contain occurrences of A which are parametric ancestors of the cut-formula A: thus (intro-A) is the lowest rule where A is the principal formula on the right of $\vdash$. The notation $\Pi_L[Y]$ and $Z[Y]$ means that all such "traced" instances of A are changed to Y: that is, instances of A which can be traced to the instance displayed on the right in $X \vdash A$. The rules contained in the new sub-derivation $\Pi_L[Y]$ are the same as the rules used in Π_L; thus it remains to be proved that $\Pi_L[Y]$ is well-formed. The resulting cut in the diagram on the right of Figure 4 is left-principal. Notice that the original sub-derivation Π may be transformed into a *different* sub-derivation Π' during this process since the parametric ancestors of A occurring in $\Pi[A]$ will in turn need to be "cut away" below where they are introduced, and replaced by Y.

There is one condition: for a parametric move, the portion of the derivation tree that is changed must not contain a cut. That is, there must be no cuts in the parts of the trees in Figure 4 shown below:

$$
\dfrac{\dfrac{Z[A] \vdash A}{\Pi_L[A]}\,(\pi)}{X \vdash A}\,(\rho)
\qquad\qquad
\dfrac{\dfrac{Z[Y] \vdash Y}{\Pi_L[Y]}\,(\pi)}{X \vdash Y}\,(\rho)
$$

Subsequently, the "mirror-image" procedure is followed, to convert a left-principal cut into one or more (left- and right-)principal cuts.

The process of making a cut left-principal, or of making a left-principal cut (left and right) principal is called a "parametric move".