Formalised Cut-elimination for the Display Calculus of Relation Algebras

Jeremy E. Dawson and Rajeev Goré

Automated Reasoning Group, Computer Sciences Laboratory, Res. Sch. of Inf. Sci. and Eng., Institute of Advanced Studies

Formal Methods Group, Dept. of Computer Science, Fac. of Eng. and Inf. Tech., The Faculties

Australian National University

jeremy@arp.anu.edu.au — arp.anu.edu/~jeremy

rpg@arp.anu.edu.au — arp.anu.edu.au/~rpg

Goré supported by an Australian Research Council Queen Elizabeth II Fellowship.
Dawson supported by an Australian Research Council Large Research Grant.

Semantics and Syntax of Logic of Relation Algebras

RA: Algebra of binary relations (sets of ordered pairs) over underlying set U

Atomic constants: $c ::= \bot \mid \top \mid \mathbf{1} \mid \mathbf{0}$ ($\emptyset \mid U \times U \mid$ identity $\mid$ diversity)

Atomic formulae: $p ::= p_0 \mid p_1 \mid p_2 \mid \cdots$ (relation symbols)

Formulae: $A ::= c \mid p \mid \neg A \mid A \wedge A \mid A \vee A \mid \smile A \mid A \circ A \mid A + A$

$\neg$ is complement $\wedge, \vee$ are $\cap, \cup$ $A + B$ is $\neg(\neg A \circ \neg B)$

converse $\smile A := \{(y,x) | (x,y) \in A\}$

composition $A \circ B := \{(x,z) | \exists y \in U.\ (x,y) \in A \ \&\ (y,z) \in B\}$

Display Calculus $\delta\mathrm{RA}$ for Relation Algebra

RA: Algebra of binary relations (sets of ordered pairs) over underlying set U

Atomic constants: $c ::= \bot \mid \top \mid \mathbf{1} \mid \mathbf{0}$ ($\emptyset \mid U \times U \mid$ identity $\mid$ diversity)

Atomic formulae: $p ::= p_0 \mid p_1 \mid p_2 \mid \cdots$ (relation symbols)

Formulae: $A ::= c \mid p \mid \neg A \mid A \wedge A \mid A \vee A \mid \smile A \mid A \circ A \mid A + A$

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

Sequent: $X \vdash Y$ where X is the *antecedent* and Y is the *succedent*

Sequent rule: (name) $\dfrac{X_1 \vdash Y_1 \cdots X_n \vdash X_n}{X \vdash Y}$ *premisses* / *conclusion*

Display Calculus $\delta\mathrm{RA}$: Display Postulates

$$\frac{*X \vdash Y}{*Y \vdash X} \quad \frac{X \vdash *Y}{Y \vdash *X} \quad \frac{**X \vdash Y}{X \vdash Y} \qquad \frac{\bullet X \vdash Y}{X \vdash \bullet Y} \quad \frac{\bullet\bullet X \vdash Y}{X \vdash Y}$$

$$\begin{array}{c} X \vdash Z ; *\bullet Y \\ \hline\hline X ; Y \vdash Z \\ \hline\hline Y \vdash *\bullet X ; Z \end{array} \quad \begin{array}{c} Z ; *\bullet Y \vdash X \\ \hline\hline Z \vdash X ; Y \\ \hline\hline *\bullet X ; Z \vdash Y \end{array} \qquad \begin{array}{c} X \vdash Z , *Y \\ \hline\hline X , Y \vdash Z \\ \hline\hline Y \vdash *X , Z \end{array} \quad \begin{array}{c} Z , *Y \vdash X \\ \hline\hline Z \vdash X , Y \\ \hline\hline *X , Z \vdash Y \end{array}$$

Intuition: Allow us to unravel complex structures: think of high school algebra where we solved $x^2 - 4 = 0$ by "making x the subject"

Invertible rules: Reversible, merely bookkeeping device

Display Calculus $\delta\mathrm{RA}$: Display Postulates

$$\frac{*X \vdash Y}{*Y \vdash X} \quad \frac{X \vdash *Y}{Y \vdash *X} \quad \frac{**X \vdash Y}{X \vdash Y} \qquad \frac{\bullet X \vdash Y}{X \vdash \bullet Y} \quad \frac{\bullet\bullet X \vdash Y}{X \vdash Y}$$

$$\begin{array}{c} X \vdash Z ; *\bullet Y \\ \hline\hline X ; Y \vdash Z \\ \hline\hline Y \vdash *\bullet X ; Z \end{array} \quad \begin{array}{c} Z ; *\bullet Y \vdash X \\ \hline\hline Z \vdash X ; Y \\ \hline\hline *\bullet X ; Z \vdash Y \end{array} \qquad \begin{array}{c} X \vdash Z , *Y \\ \hline\hline X , Y \vdash Z \\ \hline\hline Y \vdash *X , Z \end{array} \quad \begin{array}{c} Z , *Y \vdash X \\ \hline\hline Z \vdash X , Y \\ \hline\hline *X , Z \vdash Y \end{array}$$

Theorem (Belnap82): For every sequent $X \vdash Y$ and every substructure Z of $X \vdash Y$, there is a structurally equivalent sequent $Z \vdash Y'$ or $X' \vdash Z$ that has Z displayed as the whole of its antecedent or succedent.

Display Calculus $\delta\mathrm{RA}$: Display Postulates

$$\frac{*X \vdash Y}{*Y \vdash X} \quad \frac{X \vdash *Y}{Y \vdash *X} \quad \frac{**X \vdash Y}{X \vdash Y} \qquad \frac{\bullet X \vdash Y}{X \vdash \bullet Y} \quad \frac{\bullet\bullet X \vdash Y}{X \vdash Y}$$

$$\begin{array}{c} X \vdash Z ; *\bullet Y \\ \hline\hline X ; Y \vdash Z \\ \hline\hline Y \vdash *\bullet X ; Z \end{array} \quad \begin{array}{c} Z ; *\bullet Y \vdash X \\ \hline\hline Z \vdash X ; Y \\ \hline\hline *\bullet X ; Z \vdash Y \end{array} \qquad \begin{array}{c} X \vdash Z , *Y \\ \hline\hline X , Y \vdash Z \\ \hline\hline Y \vdash *X , Z \end{array} \quad \begin{array}{c} Z , *Y \vdash X \\ \hline\hline Z \vdash X , Y \\ \hline\hline *X , Z \vdash Y \end{array}$$

Theorem (Belnap82): For every sequent $X \vdash Y$ and every substructure Z of $X \vdash Y$, there is a structurally equivalent sequent $Z \vdash Y'$ or $X' \vdash Z$ that has Z displayed as the whole of its antecedent or succedent.

e.g. $\dfrac{*p_3 \vdash p_0 ; (p_0 \circ p_1)}{*(p_0 ; (p_0 \circ p_1)) \vdash p_3}$ $\dfrac{*p_3 ; *\bullet(p_0 \circ p_1) \vdash p_0}{*p_3 \vdash p_0 ; (p_0 \circ p_1)}$ $\dfrac{*\bullet p_0 ; p_3 \vdash p_0 \circ p_1}{*p_3 \vdash p_0 ; (p_0 \circ p_1)}$

Display Calculus $\delta\mathrm{RA}$: Logical Rules

- Left and right introduction rules for every logical connective
- (id) $p \vdash p$ (cut) $\dfrac{X \vdash A \quad A \vdash Y}{X \vdash Y}$
- $(+\vdash)$ $\dfrac{A \vdash X \quad B \vdash Y}{A + B \vdash X ; Y}$ decoding/rewrite $(\vdash +)$ $\dfrac{Z \vdash A ; B}{Z \vdash A + B}$
- $(\wedge\vdash)$ $\dfrac{A , B \vdash Z}{A \wedge B \vdash Z}$ rewrite/decoding $(\vdash \wedge)$ $\dfrac{X \vdash A \quad Y \vdash B}{X , Y \vdash A \wedge B}$
- Introduced formula is always displayed
- Gentzen overloading:

Antecedent Part	$\top$	$\neg$	$\wedge$	$\mathbf{1}$	$\smile$	$\circ$
Connective	I	$*$	,	E	$\bullet$	;
Succedent Part	$\bot$	$\neg$	$\vee$	$\mathbf{0}$	$\smile$	$+$

- Example: $*(p_0 ; p_1) \vdash *\bullet (p_1 , p_3)$ means $\neg(p_0 + p_1) \vdash \neg \smile (p_1 \wedge p_3)$

Display Logic $\delta\mathrm{RA}$: Structural Rules

A *structural* rule contains no logical connectives and no formula variables.

$(A ; \vdash)$ $\dfrac{X ; (Y ; Z) \vdash W}{(X ; Y) ; Z \vdash W}$ $(\vdash A)$ $\dfrac{W \vdash X , (Y , Z)}{W \vdash (X , Y) , Z}$ $(\vdash P)$ $\dfrac{Z \vdash Y , X}{Z \vdash X , Y}$

$(I^{+}_{-} \vdash)$ $\begin{array}{c} X , I \vdash Y \\ \hline\hline X \vdash Y \\ \hline\hline I , X \vdash Y \end{array}$ $(\vdash E^{+}_{-})$ $\begin{array}{c} X \vdash E ; Y \\ \hline\hline X \vdash Y \\ \hline\hline X \vdash Y ; E \end{array}$ $(\vdash \bullet E)$ $\dfrac{X \vdash E}{X \vdash \bullet E}$ $(\bullet I \vdash)$ $\dfrac{I \vdash X}{\bullet I \vdash X}$

$(M \vdash)$ $\dfrac{X \vdash Z}{X , Y \vdash Z}$ $(\vdash C)$ $\dfrac{Z \vdash X , X}{Z \vdash X}$ (tag) $\dfrac{X ; Y \vdash Z}{\bullet Y ; \bullet X \vdash \bullet Z}$

Note: no (M) (P) or (C) rules for ; since $\circ / +$ are not classical connectives

Logical Variation Via Structural Rules (modularity)

- An equation $A \leq B$ is **primitive** iff A and B are built from relational variables and the constants $\top$ and $\mathbf{1}$ with the help of $\smile$, $\wedge$, $\vee$ and $\circ$ only, and if no relational variable appears in A more than once (Kracht96).
- Purely syntactic class of formulae

Logical Variation Via Structural Rules (modularity)

- An equation $A \leq B$ is **primitive** iff A and B are built from relational variables and the constants $\top$ and $\mathbf{1}$ with the help of $\smile$, $\wedge$, $\vee$ and $\circ$ only, and if no relational variable appears in A more than once (Kracht96).
- Examples: from modal logic

reflexivity	seriality	symmetry	transitivity
$(T)\ \mathbf{1} \leq A$	$(D)\ A \circ \top = \top$	$(B)\ A = \smile A$	$(4)\ A \circ B \leq A \vee B$

Logical Variation Via Structural Rules (modularity)

- An equation $A \leq B$ is **primitive** iff A and B are built from relational variables and the constants $\top$ and $\mathbf{1}$ with the help of $\smile$, $\wedge$, $\vee$ and $\circ$ only, and if no relational variable appears in A more than once (Kracht96).

reflexivity	seriality	symmetry	transitivity
$(T)\ \mathbf{1} \leq A$	$(D)\ A \circ \top = \top$	$(B)\ A = \smile A$	$(4)\ A \circ B \leq A \vee B$
$(T)\ \dfrac{X \vdash Z}{E \vdash Z}$	$(D)\ \dfrac{I \vdash Z}{X\ ;\ I \vdash Z}$	$(B)\ \dfrac{\bullet X \vdash Z}{X \vdash Z}$	$(4)\ \dfrac{X_1 \vdash Z \quad X_2 \vdash Z}{X_1\ ;\ X_2 \vdash Z}$

- Kracht shows how to convert each primitive equation into a purely structural rule

Logical Variation Via Structural Rules (modularity)

- An equation $A \leq B$ is **primitive** iff A and B are built from relational variables and the constants $\top$ and $\mathbf{1}$ with the help of $\smile$, $\wedge$, $\vee$ and $\circ$ only, and if no relational variable appears in A more than once (Kracht96).

reflexivity	seriality	symmetry	transitivity
$(T)\ \mathbf{1} \leq A$	$(D)\ A \circ \top = \top$	$(B)\ A = \smile A$	$(4)\ A \circ B \leq A \vee B$
$(T)\ \dfrac{X \vdash Z}{E \vdash Z}$	$(D)\ \dfrac{I \vdash Z}{X\ ;\ I \vdash Z}$	$(B)\ \dfrac{\bullet X \vdash Z}{X \vdash Z}$	$(4)\ \dfrac{X_1 \vdash Z \quad X_2 \vdash Z}{X_1\ ;\ X_2 \vdash Z}$

- Theorem [Belnap 82]: Any display calculus whose rules obey eight conditions given by Belnap enjoys cut-elimination. If $X \vdash Y$ is derivable, then it is derivable without the cut rule.
- Theorem: Using results of Kracht 96 we can construct a *cut-free* display calculus which is sound and complete for any primitive extension of RA.

Formalising $\delta\mathrm{RA}$ in Isabelle/HOL

Formulae: $A \ ::= \ c \mid p \mid \neg A \mid A \wedge A \mid A \vee A \mid \smile A \mid A \circ A \mid A + A$

```
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)
                 | Rneg formula ("_^" [75] 75)
                 | Btrue ("T")
                 | Bfalse("F")
                 | Rtrue ("r1")
                 | Rfalse("r0")
                 | PP string
                 | FV string
```


Formalising $\delta\mathrm{RA}$ in Isabelle/HOL

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

```
datatype structr = Comma structr structr                ,
                 | SemiC structr structr                ;
                 | Star structr                         *
                 | Blob structr                         @
                 | I                                    $I
                 | E                                    $E
                 | Structform formula
                 | SV string                            $''X''

datatype sequent = Sequent structr structr              |-
```

`$''X'' |- ''A''` is syntactic sugar for

`Sequent (SV ''X'') (Structform (FV ''A''))`

A Deep Embedding of Rules

(name) $\dfrac{X_1 \vdash Y_1 \cdots X_n \vdash X_n}{X \vdash Y}$ $\qquad$ $\dfrac{X \vdash Y\ ;\ Z}{* \bullet Y\ ;\ X \vdash Z}$

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

ss1   "ss1 == Bidi ($''X'' |- $''Y''; $''Z'')
                   ( *@$''Y''; $''X'' |- $''Z'')"
```

Deep embedding because of the `''X''`. Shallow embedding would use `?X`.

This means we must handle substitutions explicitly.

`rls` is the set of rules of $\delta\mathbf{RA}$

A shallow embedding would provide, not model, the set of rules.

Formalising Derivations

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

True derivation has leaves `Der (PP ''p'' |- PP ''p'') id []`

A derived rule will have some `Unf`inished leaf sequents

Example:

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

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

`idf` is the *admissible* rule $A \vdash A$

Reasoning About Derivations

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

`wfb :: "dertree => bool"` returns true if the end-sequent of `dertree` and the end-sequents of its immediate children in `dertree list` are a well-formed instance of `rule`

`allDT :: "(dertree => bool) => dertree => bool"` returns true if every node of `dertree` satisfies the given boolean valued function

`allDT wfb` returns true if every rule application in `dertree` is well formed

`allDT (frb rules)` returns true if every rule application in `dertree` is a member of set `rules`

Formalising Derivability

`IsDerivableR ::"rule set => [sequent set,sequent] => bool"`

Definition 1 *`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)"
```

Lemma 1 *The sequent `PP ''p'' |- PP ''p''` is derivable for all atoms p*

Proof: Let `dt` be `Der (PP ''p'' |- PP ''p'') id [] in`

```
IsDerivableR rls {} (PP ''p'' |- PP ''p'')
```


Results About Derivability

Our main result about derivability is a recursive characterisation of derivability.

Theorem 1 *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 themselves 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
```


Formalising Derivable Rules

```
"IsDerivable rules (Rule prems concl) =
     IsDerivableR rules (set prems) concl"
```

Lemma 2 (`idf`) *The sequent `''A'' |- ''A''` is derivable for all formulae A that contain no formula variables*

Add the rule `idf` to `rls` :

```
idf == Rule [] (''A'' |- ''A'')
```

`Der (''A'' |- ''A'') idf []` stands for a complete derivation which uses the (proven) admissible rule `idf` that $A \vdash A$ is derivable

`Unf (''A'' |- ''A'')` stands for an incomplete derivation with unfinished premiss $A \vdash A$

Reducing a Left-and-Right Principal Cut

- A cut on A is *left- [right-] principal* if the left [right] inference immediately above the cut introduces A via a logical rule, written $(\vdash A)$ $[(A \vdash)]$

$$\dfrac{\dfrac{\Pi_L}{X \vdash A}(\vdash A) \qquad \dfrac{\Pi_R}{A \vdash Y}(A \vdash)}{X \vdash Y}$$

- C8: All left-and-right principal cuts can be replaced by cuts on strictly smaller subformulae of A
- Base Case: (cut) $\dfrac{p \vdash p \quad p \vdash p}{p \vdash p}$ reduced to just $p \vdash p$
- Similar cases for $\top$, **1**, $\perp$, **0**

Example of a Principal reduction (C8 condition)

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

Principal cut on formula $A \vee B$

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

Transformed derivation uses cuts only on A and B

Must exhibit and hard-wire such transformations for every logical connective

So if cut is *not* left-and-right principal then we try to make it so ...

Making a cut left-principal

- Given a derivation ending in a non left-and-right principal cut on formula A, we trace the ancestor occurrences of this A up Π_L:

$$\dfrac{\dfrac{\Pi_L}{X \vdash A} \quad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}$$

- an ancestor of A is *principal* if introduced by logical rule $(\vdash A)$
- all other ancestors of A are *parametric* ancestors (including those "introduced" by weakening
- there must be at least one introduction of A in every branch
- not all occurrences of A need be ancestors of this occurrence of A

Making a cut left-principal

- Given a derivation ending in a non left-and-right principal cut on formula A, we trace all ancestor occurrences of this A up Π_L:
- Let $\Pi[A := Y]$ mean "in Π, replace all parametric ancestors of A with Y"
- Suppose Π_L does not branch at all for simplicity

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi_L^1}{V \vdash A}(\vdash A)}{\Pi_L^2}(\pi)}{X \vdash A}(\rho) \qquad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}(cut) \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\Pi_L'^1}{V \vdash A}(\vdash A) \quad \dfrac{\Pi_R}{A \vdash Y}}{V \vdash Y}(cut)}{\Pi_L^2[A := Y]}(\pi)}{X \vdash Y}(\rho)$$

- New cut is left-principal (similarly make it also right principal)

Making a cut left-principal

- Base case of induction, Π^1_L is empty, A introduced by $A \vdash A$. No new cuts needed

$$\dfrac{\dfrac{\dfrac{A \vdash A}{\Pi^2_L}\,(\pi)}{X \vdash A}\,(\rho) \qquad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}\,(cut) \qquad\qquad \dfrac{\dfrac{\dfrac{\Pi_R}{A \vdash Y}}{\Pi^2_L[A := Y]}\,(\pi)}{X \vdash Y}\,(\rho)$$

- Case for "introduction" by weakening, no new cuts needed

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi^1_L}{V \vdash W \,,\, Z(A)}\,(\text{Wk } Z(A))}{\Pi^2_L}\,(\pi)}{X \vdash A}\,(\rho) \qquad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}\,(cut) \qquad\qquad \dfrac{\dfrac{\dfrac{\Pi'^1_L}{V \vdash W \,,\, Z(Y)}\,(\text{Wk } Z(A := Y))}{\Pi^2_L[A := Y]}\,(\pi)}{X \vdash Y}\,(\rho)$$

Making a cut (left-and-right-)principal

- Similarly to make a left-principal cut also right principal
- But note, one cut $\rightsquigarrow$ *several* cuts (in general)
- Summing up:

  single non-principal cut $\rightsquigarrow$ several left-principal cuts
  single left-principal cut $\rightsquigarrow$ several principal cuts
  single principal cut $\rightsquigarrow$ several cuts on sub-formulae

- To deal with several left-and-right-principal/left-principal/arbitrary cuts, deal with a top-most one first (sub-tree containing single cut)
- Procedurally, this is a deeply nested recursion.
- Is an equally deeply nested inductive proof necessary?

Cut-Admissibility (Weak Normalisation)

Show that the cut rule (cut) $\dfrac{X \vdash A \quad A \vdash Y}{X \vdash Y}$ is *admissible*

Assume: we have cut-free derivations of $X \vdash A$ and $A \vdash Y$

Show: that there is a cut-free derivation of $X \vdash Y$

In general, cut-elimination proceeds by repeatedly eliminating an arbitrary top-most cut

Must show that this procedure terminates

Usual argument is a double induction on the size (degree) of the cut formula and the heights of the given derivations (grade)

Cut-Admissibility By Structural Induction

`elim` eliminates a single arbitrary top-most cut, by turning it into several left-principal cuts and using `elimAllLP` to eliminate them . . .

Our initial derivation contains only one bottom-most cut which is not left and right principal and we know how to turn it into multiple left principal cuts

Cut-Admissibility By Structural Induction

`elim` eliminates a single arbitrary top-most cut, by turning it into several left-principal cuts and using `elimAllLP` to eliminate them . . .

all new cuts are left principal

`elimAllLP` eliminates several left-principal cuts, by repeatedly using `elimLP` to eliminate the top-most remaining one . . .

Cut-Admissibility By Structural Induction

`elim` eliminates a single arbitrary top-most cut, by turning it into several left-principal cuts and using `elimAllLP` to eliminate them . . .

`elimAllLP` eliminates several left-principal cuts, by repeatedly using `elimLP` to eliminate the top-most remaining one . . .

`elimLP` eliminates a top-most left-principal cut, by turning it into several principal cuts and using `elimAllLRP` to eliminate them . . .

all new cuts are left-and-right principal

Cut-Admissibility By Structural Induction

`elim` eliminates a single arbitrary top-most cut, by turning it into several left-principal cuts and using `elimAllLP` to eliminate them . . .

`elimAllLP` eliminates several left-principal cuts, by repeatedly using `elimLP` to eliminate the top-most remaining one . . .

`elimLP` eliminates a top-most left-principal cut, by turning it into several principal cuts and using `elimAllLRP` to eliminate them . . .

`elimAllLRP` eliminates several principal cuts, by repeatedly using `elimLRP` to eliminate the top-most remaining one . . .

`elimLRP` eliminates a top-most principal cut, by turning it into several cuts on *smaller* cut-formulae, and using `elimAll` to eliminate them . . .

all new cuts are on strictly smaller cut-formulae

Cut-Admissibility By Structural Induction

`elim` eliminates a single arbitrary top-most cut, by turning it into several left-principal cuts and using `elimAllLP` to eliminate them . . .

`elimAllLP` eliminates several left-principal cuts, by repeatedly using `elimLP` to eliminate the top-most remaining one . . .

`elimLP` eliminates a top-most left-principal cut, by turning it into several principal cuts and using `elimAllLRP` to eliminate them . . .

`elimAllLRP` eliminates several principal cuts, by repeatedly using `elimLRP` to eliminate the top-most remaining one . . .

`elimLRP` eliminates a top-most principal cut, by turning it into several cuts on *smaller* cut-formulae, and using `elimAll` to eliminate them . . .

`elimAll` eliminates several arbitrary cuts, by repeatedly using `elim` to eliminate the top-most remaining one . . . *termination*

Putting It All Together

```
  canElim      :: "(dertree => bool) => bool"
  canElimAll   :: "(dertree => bool) => bool"
```

`canElim f` means that if a derivation tree `pt` has no cuts other than at the bottom, and satisfies `f`, then there is an equivalent cut-free derivation tree.

`canElimAll f` means that if every subtree of a given tree `pt` satisfies `f`, then there is a cut-free derivation tree `pt'` equivalent to `pt` (we also require that if the bottom rule of `pt` is not (*cut*), then the bottom rule of `pt'` is the same).

```
allLP' = "canElimAll (cutIsLP ?A) ==> canElim (cutOnFmls {?A})"
```

Here `f` is `(cutIsLP ?A)` [`(cutOnFmls {?A})`] which means that if the lowest rule application is cut then it is a left-principal cut with cut formula `?A` [if the lowest rule application is cut then the cut formula is from set `{?A}`]

`allLP'` means "if you can eliminate several cuts, all left-principal on A, then you can eliminate a single cut, at the bottom, on A" (its backwards!)

Putting It All Together ...

```
allLP' = "canElimAll (cutIsLP ?A) ==> canElim (cutOnFmls {?A})"
allLRP' = "canElimAll (cutIsLRP ?A) ==> canElim (cutIsLP ?A)"
elimLRP = "canElim (cutIsLRP ?A) ==> canElimAll (cutIsLRP ?A)"
elimLP = "canElim (cutIsLP ?A) ==> canElimAll (cutIsLP ?A)"
```

Chaining these implications gives

```
canElim (cutIsLRP ?A) ==> canElim (cutOnFmls ?A)
```

Dealing with left-right-principal cuts gives

```
canElimAll (cutOnFmls {?A, ?B}) ==>
   canElim (cutIsLRP (?A v ?B))

elimFmls = "canElim (cutOnFmls ?s) ==> canElimAll (cutOnFmls ?s)'

"[| canElim (cutOnFmls {?A}); canElim (cutOnFmls {?B}) |]
    ==> canElim (cutOnFmls {?A v ?B})"
```


Finally ...

A proof by induction on the structure of a formula then gives

```
canElimFml = "canElim (cutOnFmls {?fml})"
```

("you can eliminate one cut on any given formula `fml`"),

```
canElimAny = "canElim (cutOnFmls UNIV)",
```

("you can eliminate one cut whatever the cut-formula"), and, using `elimFmls`,

```
canElimAll = "canElimAll (cutOnFmls UNIV)"
```

Finally we convert `canElimAll` into

```
cutElim = "IsProvableR rls {} ?concl  ==>
           IsProvableR (rls - {cut}) {} ?concl"
```


Strong Normalisation

Cut-admissibility proceeds by eliminating the top-most cut

Strong normalisation: define reduction steps, eliminate any cut that can be reduced, and prove that this cut-elimination procedure terminates:

- $\Pi_n < \Pi_{n-1} < \cdots < \Pi_2 < \Pi_0$
- Π_n is cut-free
- n is finite
- weak-normalisation (attacking top most cut) is an instance of this method

Usually define reductions as either a parametric or principal move, with proviso that we must not cross a cut in a parametric move

Why Formalise (in Isabelle) ?

Strong-normalisation proofs are notoriously difficult involving many cases

We tried to formalise Heinrich Wansing's proof of strong-normalisation for display logic, but got stuck at one particular case

Why Formalise (in Isabelle) ?

Strong-normalisation proofs are notoriously difficult involving many cases

We tried to formalise Heinrich Wansing's proof of strong-normalisation for display logic, but got stuck at one particular case

Found a problem ... Wansing assumes that Π_L^1 is unchanged!

V_A is $(*A \,,\, X)$ $\qquad$ V_Y is $(*Y \,,\, X)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_L^1}{V_A \vdash A}\,(\vdash A)}{X \vdash A, A}\,(\mathrm{dp})}{X \vdash A}\,(\mathrm{ctr}) \qquad \dfrac{\Pi_R}{A \vdash Y}}{X \vdash Y}\,(cut)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_L'^1}{V_Y \vdash A}\,(\vdash A) \qquad \dfrac{\Pi_R}{A \vdash Y}}{V_Y \vdash Y}\,(cut)}{X \vdash Y, Y}\,(\mathrm{dp})}{X \vdash Y}\,(\mathrm{ctr})$$

Branch in left subtree — before and after

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi'}{X' \vdash A} \quad \dfrac{\Pi_R}{A \vdash Y}}{X' \vdash Y}\,(cut) \qquad \dfrac{\dfrac{\Pi''}{X'' \vdash A} \quad \dfrac{\Pi_R}{A \vdash Y}}{X'' \vdash Y}\,(cut)}{\Pi_L[A ?= Y]}\,(\pi)}{X \vdash Y}\,(\rho)$$

Two successive introductions – before

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi}{X, B \vdash *B}\,(\rho)}{X, B \vdash \neg B}\,(\vdash \neg)}{*\neg B, X \vdash *B}\,(\mathrm{dp})}{*\neg B, X \vdash \neg B}\,(\vdash \neg)}{X \vdash \neg B, \neg B}\,(\mathrm{dp})}{X \vdash \neg B}\,(\vdash \mathrm{cont}) \qquad \dfrac{\Pi_R}{\neg B \vdash Y}}{X \vdash Y}\,(cut)$$

Two successive introductions – after

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi}{X,B\vdash *B}(\rho)}{X,B\vdash\neg B}(\vdash\neg)\quad\dfrac{\Pi_R}{\neg B\vdash Y}}{X,B\vdash Y}(cut)}{*Y,X\vdash *B}(\mathrm{dp})}{*Y,X\vdash\neg B}(\vdash\neg)\quad\dfrac{\Pi_R}{\neg B\vdash Y}}{*Y,X\vdash Y}(cut)}{X\vdash Y,Y}(\mathrm{dp})}{X\vdash Y}(\vdash \mathrm{cont})$$

Defining Reductions

Definition 2 *Assuming a 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 of the immediate subtrees of 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')) )"
```


Defining Strongly Normalising Derivation Trees

Definition 3 *The set* `sn_set` *is the smallest set of derivation trees such that:*

(a) if Π_0 cannot be reduced then $\Pi_0 \in$ `sn_set`

(b) 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"
```

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.

Various Binary Orderings

1. $\Pi_1 <_{\mathtt{LRP}} \Pi_0$ if the bottom inferences of derivations Π_1 and Π_0 are both cut, and either
   - **a.** the cut in Π_1 is left-principal or right-principal, and the cut in Π_0 is neither, or
   - **b.** the cut in Π_1 is (left- and right-)principal, and the cut in Π_0 is not (left- and right-)principal.
2. $\Pi_1 <_{\mathtt{cut}} \Pi_0$ if the bottom inferences of derivations Π_1 and Π_0 are both cut, and if either
   - **a.** the size of the cut-formula of Π_1 is smaller than that of Π_0, or
   - **b.** each derivation has the same cut-formula, and $\Pi_1 <_{\mathtt{LRP}} \Pi_0$.

Various Binary Orderings

1. $\Pi_1 <_{\mathtt{sn1}} \Pi_0$ 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.
2. $\Pi_1 <_{\mathtt{dt}} \Pi_0$ iff any of the following hold:
   - **a.** the bottom inference of Π_1 is not cut, but that of Π_0 is
   - **b.** $\Pi_1 <_{\mathtt{cut}} \Pi_0$
   - **c.** $\Pi_1 <_{\mathtt{sn1}} \Pi_0$.

Theorem 2 *The relations* `LRPorder`*,* `cutorder`*,* `sn1order`*, and* `dtorder` *are well-founded*

Despite 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.

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 4 *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 3 *A derivation tree Π is strongly normalisable iff every subtree of Π has the property* `snHered`*.*

Proof: The "only if" part uses a clever instantiation of a theorem generated automatically by Isabelle from the inductive definition of `sn_set`.

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

Defining Cut-Reductions

Definition 5 *Two derivation trees Π_0 and Π_1 satisfy* `nparRedP` *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 $\Pi_1^s <_{\mathtt{LRP}} \Pi_0$.

Definition 6 *Two derivation trees Π_0 and Π_1 satisfy* `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 that of the lowest rule of Π_0.

Defining Cut-Reduction and Hence Reduction

Definition 7 *The derivation tree Π_0 cut-reduces to Π_1 if the following hold simultaneously:*

(a) Π_0 and Π_1 satisfy either `nparRedP` *(for a parametric reduction) or* `c8redP` *(for a principal reduction)*

(b) Π_0 and Π_1 have the same conclusion

(c) the bottom rule of Π_0 is cut

(d) Π_0 and Π_1 are not identical

(e) Π_1 does not consist solely of an unfinished leaf

(f) and Π_0 has at least one immediate subtree

Strong-Normalisation

Lemma 4 *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 3 *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 4 that every derivation tree satisfies `snHered`; the result follows from Lemma 3.

Where are we?

We have actually shown using Isabelle that a class of reductions which we have defined (in terms of *cutReduces*, and hence of *nparRedP* and *c8redP*) is strongly normalising. We would like to show that this class of reductions contains the ones to which Wansing's theorem refers.

We can't do that, but we can at least show that this class contains enough reductions to eliminate any cuts.

Dealing With Parametric Cuts

Theorem 4 *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 transformation can be performed without traversing another cut. Also assume the cut is not already left-principal. Then the reduction satisfies the condition* `nparRedP`.

Theorem 5 *The same as Theorem 4, except with conclusion "Then the reduction satisfies the condition* `cutReduces`*".*

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.

Dealing With Principal Cuts

Theorem 6 *Case for* $\vee$*: 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*).*

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

Theorem 7 *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.

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

Cut-Elimination Via Strong Normalisation

Definition 8 *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.

Theorem 8 *Parametric and principal moves give valid reductions.*

Theorem 9 *For any valid tree which contains a cut, there is available at least one valid reduction (reduce a top-most cut).*

Corollary: 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.

Theorem 10 *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
```


Belnap's Conditions and New C8

Appendix: Belnap's Conditions.

For every sequent rule Belnap82, page 388 first defines the following notions: in an application *Inf* of a sequent rule (ρ), "constituents occurring as part of occurrences of structures assigned to structure-variables are defined to be **parameters** of *Inf*; all other constituents are defined as **nonparametric**, including those assigned to formula-variables. Constituents occupying similar positions in occurrences of structures assigned to the same structure-variable are defined as **congruent** in *Inf*". The eight (actually seven) conditions shown below are from [Kracht96]:

(C1) Each formula which is a constituent of some premiss of a rule ρ is a subformula of some formula in the conclusion of ρ.

(C2) Congruent parameters are occurrences of the same structure.

(C3) Each parameter is congruent to at most one constituent in the conclusion. Equivalently, no two constituents of the conclusion are congruent.

(C4) Congruent parameters are either all antecedent parts or all succedent parts of their respective sequent.

(C5) If a formula is non-parametric in the conclusion of a rule ρ, it is either the entire antecedent, or the entire succedent (called a **principal** formula).

(C6/7) Each rule is closed under simultaneous substitution of arbitrary structures for congruent parameters.

(C8) If there are inference rules ρ_1 and ρ_2 with respective conclusions $X \vdash \varphi$ and $\varphi \vdash Y$ with φ principal in both inferences (in the sense of C5), and if (cut) is applied to yield $X \vdash Y$ then, either $X \vdash Y$ is identical to $X \vdash \varphi$ or to $\varphi \vdash Y$; or it is possible to pass from the premisses of ρ_1 and ρ_2 to $X \vdash Y$ by means of inferences falling under (cut) where the cut-formula is always a proper subformula of φ.