Machine-checked Interpolation Theorems for Substructural Logics using Display Calculi

Jeremy E. Dawson[1]*, James Brotherston[2]**, and Rajeev Goré[1]

[1] Research School of Computer Science, Australian National University
[2] Queen Mary College, University of London, UK

Abstract. We give mechanised proofs of Craig interpolation for a class of propositional substructural logics using display calculi.
We discuss the difficulties caused by various rules, such as the binary logical introduction rules in their additive and multiplicative forms, and the weakening and unit weakening rules.
We also describe the differences between our proofs and those of Brotherston and Goré, differences motivated by the ease of formalising the definitions used in intermediate lemmas, and we detail how we proved the more difficult cases, such as the weakening rules and the multiplicative binary logical introduction rules.
Finally, we discuss the value for this work of using a prover with a programmable user interface (in our case, Isabelle with its Standard ML interface).

Keywords: Craig interpolation, display logic, interactive theorem proving, proof theory

1 Introduction

In calculi for logical entailment, *Craig interpolation* is the property that for any entailment $A \vdash B$ between formulae, there exists an *interpolant* formula I such that $A \vdash I$ and $I \vdash B$ are both entailments of the calculus, while I mentions only those variables or nonlogical constants that are common to both A and B. It has long been known that there are close connections between interpolation and other central logical concerns (see e.g. [7]); indeed, one of Craig's original applications of interpolation was to give a new proof of Beth's Definability Theorem [4]. More recently, though, it has transpired that interpolation has significant applications in program verification as well. For example, in inferring the invariants of program loops [9], and in model checking [3].

Recently, Brotherston and Goré [2] gave a modular proof of interpolation for a class of propositional substructural logics, based on Belnap's *display logic* [1]. Roughly speaking, display calculi are two-sided sequent calculi equipped with a richer-than-usual notion of sequent structure and a principle by which sequents can be rearranged so as to "display" any chosen substructure as the

* Supported by Australian Research Council Discovery Grant DP120101244
** Supported by an EPSRC Career Acceleration Fellowship

entire left or right hand side (much like rearranging a mathematical equation for a chosen variable). The main attraction of display calculi is Belnap's general cut-elimination result, which says that cut-elimination holds for any display calculus whose rules satisfy 8 easily verifiable syntactic conditions. Cut-elimination is generally essential to the standard proof-theoretic approach to interpolation, which is to proceed by induction on cut-free derivations. Despite the availability of a general cut-elimination result, however, there seem to have been no proofs on interpolation based on display calculi prior to [2], probably due to the inherent complexity of their sequent structure and display principles. Indeed, in line with this general expectation, Brotherston and Goré's proof is very technical, involving many case distinctions, and many intricate properties of substitutions. Moreover, due to space restrictions, most of the proofs are only sketched, leaving the potential for errors. Thus it makes sense to verify these intricate details using an interactive theorem prover, to give us greater confidence in their very general interpolation theorems.

In this paper, we report on the Isabelle formalisation of their results. We discuss the difficulties in formalising their proofs and describe the consequent differences between their proofs and ours. We also highlight the usefulness of a programmable user interface.

Isabelle is an automated proof assistant. 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 $\alpha_1, \alpha_2, \ldots, \alpha_n, \Longrightarrow \beta$ 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 which is an Isabelle theory based on the higher-order logic of Church HOL offers inductively defined sets, and recursive datatypes, which we use extensively.

In the paper we show some Isabelle code, edited to use mathematical symbols. In it, a '`?`' precedes a variable name. The Appendix gives the actual Isabelle text of many definitions and theorems. Our Isabelle code can be found at `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/interp/`. HOW MANY LINES FOR THIS WORK?

In Section 3, we briefly describe Display Logic, and the system defined in [2]. We outline our approach to proving interpolation for the Display Logic system of [2].

2 Display calculi for (some) substructural logics

We now recall the class of display calculi for which Craig interpolation is established in [2].

We assume a fixed infinite set of propositional variables. *Formulae* F and *structures* X are then given by the following grammars, where P ranges over

propositional variables:

$$F ::= P \mid \top \mid \bot \mid \neg F \mid F \,\&\, F \mid F \vee F \mid F \to F \mid \top_a \mid \bot_a \mid F \,\&_a\, F \mid F \vee_a F$$
$$X ::= F \mid \emptyset \mid \sharp X \mid X \,;\, X$$

Formula connectives with an "a" subscript stand for an *additive* version of that connective, while connectives without a subscript are construed as *multiplicative*. However, in the Isabelle formulation we do not duplicate the connectives in this way — rather, we identify which logical rules for which our various results apply. We write F, G etc. to range over formulae and W, X, Y, Z etc. to range over structures. If X and Y are structures then $X \vdash Y$ is a *consecution*.

The complete set of proof rules for our display calculi is given in Figure 1[3]. As usual, we begin by giving a set of *display postulates*, and taking the least equivalence closed under the postulates to be our notion of *display-equivalence*. We then have the usual *display theorem*, which says that for any structure occurrence Z in a consecution $X \vdash Y$, one has either $X \vdash Y \equiv_D Z \vdash W$ or $X \vdash Y \equiv_D W \vdash Z$ for some W, depending on whether Z occurs positively or negatively in $X \vdash Y$. Rearranging $X \vdash Y$ into $Z \vdash W$ or $W \vdash Z$ in this way is called *displaying* Z. We remark that the display postulates "build in" commutativity of the structural semi-colon, so that we consider only calculi for commutative logics.

Brotherston and Goré [2] consider the additive rules, collectively, and each structural rule, individually, to be *optional* inclusions in their calculi. At present, our mechanisation assumes the presence of the unit rules ($\emptyset\mathrm{W_L}$), ($\emptyset\mathrm{W_R}$), ($\emptyset\mathrm{C_L}$), ($\emptyset\mathrm{C_R}$) and the associativity rule (α). Thus the smallest display calculus we consider gives multiplicative linear logic MLL. By adding the additive logical rules we obtain multiplicative-additive linear logic MALL, and by adding the full weakening rule (W) or the full contraction rule (C) we obtain affine or strict variants of these logics, respectively. Note that rules for weakening and contraction on the right can be derived using the display postulates from the corresponding left rules. Of course, if we add *both* weakening and contraction then we obtain standard classical propositional logic.

No matter which variant of these display calculi we consider, we have the standard *cut-elimination* result due to Belnap. Since we omit the cut rule from our presentation of the display calculi in Figure 1, we state it here in the weaker form of *cut admissibility*:

Theorem 1 (cf. [2]). *If $X \vdash F$ and $F \vdash Y$ are both provable then so is $X \vdash Y$. Moreover, this property is not affected by the presence or otherwise of the additive logical rules (collectively), or of any of the structural rules.*

3 Interpolation for Display Calculi

In traditional sequent calculi, it is fairly straightforward to decorate each rule with interpolants by building up the interpolant for the conclusion sequent from

[3] how to stop this going to the end of the document?

the interpolants for the premise sequents. this approach is harder in display calculi since the sequent $X \vdash Y$ goes through many transformations while displaying some substructure Z. Brotherston & Goré therefore consider the following (LADI) property [2, Definition 3.4]:

(LADI): a rule with premises $\mathcal{C}_i$ and conclusion $\mathcal{C}$ satisfies the local AD display interpolation property (LADI), if given that, for all premises $\mathcal{C}_i$, all sequents $\mathcal{C}_i'$ such that $\mathcal{C}_i' \equiv_{AD} \mathcal{C}_i$ satisfy the interpolation property, all sequents $\mathcal{C}'$ such that $\mathcal{C}' \equiv_{AD} \mathcal{C}$ satisfy the interpolation property.

Although Brotherston and Goré [2] give the separate variants of the logical connectives $\top, \bot, \wedge$ and $\vee$ for the additive and multiplicative forms of the logical introduction rules, we just use one connective for each of $\top, \bot, \wedge$ and $\vee$. Although the additive and multiplicative forms are equivalent in the presence of contraction and weakening, [2] contains results which are relevant to the situation where not all structural rules are included. Thus they prove results for the both rules shown below, even though the second rule is much easier to deal with.

$$\frac{X \vdash A \quad Y \vdash B}{X, Y \vdash A \wedge B} \qquad \frac{X \vdash A \quad X \vdash B}{X \vdash A \wedge B}$$

In the work described here, we first considered the second (additive) rule shown; we subsequently developed a proof dealing with the first (multiplicative) rule directly.

4 The Isabelle mechanisation

Our mechanisation builds on the work of Dawson & Goré [5] in formalising Display Logic. Some of our notation and choices of properties, lemmas, etc, are attributable to this. In particular, we use `Comma`, `Star` and `I` for ';', '#' and '∅'.

The work in [5] is a deep embedding of rules and of the variables in them, and we have followed that approach here (see [6] for our understanding of what this means, and for more details). That is, we define a language of formulae and structures, which contains explicit structure and formula variables, for which we define explicit substitution functions. We also define the rules as specific data structures (of which there is a small finite number, such as those in Figure 1), and infinitely many substitution instances of these rules.

4.1 Formalising Display Logic in Isabelle

An actual derivation in a Display Calculus involves structures containing formulae which are composed of primitive propositions (which we typically represent by p, q, r). It uses rules which are *expressed* using structure and formula variables, typically X, Y, Z and A, B, C respectively, to represent structures and formulae made up from primitive propositions. We are using a "deep embedding" of variables, so our Isabelle formulation explicitly represents variables such as X, Y, Z and A, B, C, and defines substitution for them of given structures and formulae, which may themselves contain variables. In our Isabelle formulation we use `SV`

name and `FV` *name* to represent formula and structure variables. The constructor `PP` represents a primitive proposition variable p.

Formulae are therefore represented by the datatype below:

```
datatype formula = Btimes formula formula ("_ && _" [68,68] 67)
 | Bplus formula formula ("_ v _" [64,64] 63)
 | Bneg formula ("--_" [70] 70)
 | Btrue ("T") | Bfalse("F")
 | FV string | PP string
```

Structures are represented by the datatype below:

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

The operator `Structform` "casts" a formula into a structure, since a formula is a special case of a structure.

The notation in parentheses in the definition of datatype `formula` describe an alternative infix syntax, closer to normal logical syntax. Some complex manipulation of the syntax, available through Isabelle's "parse translations" and "print translations", allows structure variables and constants to be prefixed by the symbol `$`, and the notations `FV`, `SV` and `Structform` to be omitted.

Sequents and rules are represented by the Isabelle/HOL datatypes:

```
datatype sequent = Sequent structr structr
datatype rule = Rule (sequent list) sequent
```

`Rule prems concl` means a rule with list of premises `prems` and conclusion `concl`. A sequent `(Sequent X Y)` can also be represented as `$X |- $Y`. Thus the term `Sequent (SV ''X'') (Structform (FV ''A''))` is printed, and may be entered, as `($''X'' |- ''A'')`.

Since a "deep" embedding requires handling substitution explicitly, we defined functions to substitute for structure and formula variables, in structures, sequents and rules. In particular, we have a operator

```
rulefs :: "rule set => rule set"
```

where `rulefs` *rules* is the set of substitution instances of rules in the set *rules*. Also, when we refer to derivability using a set of rules, this allows inferences using substitution instances of these rules, and `derivableR` *rules sequents* means the set of sequents which can be derived from *sequents* using *rules*, instantiated.

```
derivableR  :: "rule set => sequent set => sequent set
```

We also use some general functions to describe derivability, for which we had many useful lemmas. A more detailed expository account of these is given in [8].

```
types 'a psc = "'a list * 'a"       (* single step inference *)
consts
  derl      :: "'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` while `derrec rls prems` is the set of sequents derivable using rules `rls` from the set `prems` of premises. We defined these using Isabelle's package for inductively defined sets.

Note that these functions do not envisage instantiation of rules. In fact, we have the following relationship between `derivableR` and `derrec`, using a function `PC`, defined as shown, to translate between types.

```
"PC (Rule ?prems ?concl) = (?prems, ?concl)"
"derivableR ?rules == derrec (PC ‘ rulefs ?rules)"
```

The "deep embedding" approach to rules enables us to express properties of rules, such as that no structure variable appears in both antecedent and succedent positions; some of our lemmas apply to all display postulates satisfying conditions of this sort. We used this in [5] in showing that cut-admissibility applies whenever the structural rules were all of a particular form (as in Belnap's cut elimination theorem). In regards to interpolation, possible future work may include showing that interpolation results hold whenever rules are of a particular form, but our present work (except for some lemmas) do not do this.

The work in [5] is also a deep embedding of proofs (where we took proof objects and explicitly manipulated them) but we have *not* done that here.

4.2 Definitions relating to interpolation

We define the following sets of rules:

`dps`: is the set of six display postulates in [2, Definition 2.5] (Display-equivalence)
`aidps`: is `dps`, their inverses, and the associativity rule (ie, 13 rules)
`ilrules`: is the unit-contraction and unit-weakening rules
`rlscf`: is the set of all rules of the logic as shown in [2, Figures 1 and 3], plus `aidps` (but, to shorten the verification task, we omitted the rules for implication $\rightarrow$)
`rlscf_nw`: is as `rlscf`, but excluding the weakening rule

We define several predicates to do with interpolation:

```
interp :: "rule set => sequent => formula => bool"
edi :: "rule set => rule set => sequent => bool"
ldi :: "rule set => rule set => sequent list * sequent => bool"
cldi :: "rule set => rule set => sequent list * sequent => bool"
```

`interp` *rules* $(X \vdash Y)$ *intp* says that `intp` is an interpolant for $X \vdash Y$, ie, that $X \vdash$ *intp* and *intp* $\vdash Y$ are derivable (using *rules*) and that the (formula) variables in *intp* are among the formula variables of the structures X and Y.

`edi` *lrules drules* $(X \vdash Y)$ (*Extended Display Interpolation*) says that for all sequents $X' \vdash Y'$ from which $X \vdash Y$ is derivable using *lrules*, $X' \vdash Y'$ has an interpolant (defined in terms of derivability using *drules*) (*lrules* would typically be a set of display postulates)

`ldi` *lrules drules* (ps, c) (*Local Display Interpolation*) says that the rule (ps, c) preserves the property `edi`: that is, if, for all $p \in ps$, `edi` *lrules drules* p holds, then `edi` *lrules drules* c holds. That is, if `lrules` is the set AD of rules (our `aidps`), and `drules` is the set of rules of the logic, then the local AD-interpolation (LADI) property as defined in [2, Definition 3.4], for rule (ps, c), is that `ldi` *lrules aidps* (ps, c) holds.

Note that none of these definitions involves a condition that $X \vdash Y$ be derivable. Of course, cut-admissibility would imply that if $X \vdash Y$ has an interpolant then $X \vdash Y$ is derivable, but we avoid proving or using cut-admissibility. Even so, in most cases we do not need such a condition. However we do need $X \vdash Y$ in the case of a sequent $I \vdash Y, \#X$ produced by weakening and displaying I. Thus we need a predicate with that condition:

`cldi` *lrules drules* (ps, c) (*Conditional Local Display Interpolation*) says that if c is is derivable using *drules*, then `ldi` *lrules drules* (ps, c) holds.

However we also need variants of these predicates, called `interpn`, `edin`, `ldin` and `cldin`, where the derivation of interpolated sequents is from a given set of rules, rather than from given rules and their substitution instances. We use these in some subsequent lemmas which involve rule sets which are not closed under substitution.

We mention here that many of our lemmas about these properties assume, although we do not specifically say so, that AD (rule set `aidps`) is used as *lrules* in the above definitions, and that the derivation rules, *drules* in the above definitions, contain the AD rules.

Lemma 3.5 of [2] says that if all rules satisfy the local AD-interpolation property, then the calculus has the interpolation property. In fact the stronger result, Lemma 1(a) is true, that LADI is preserved under derivation. But for the conditional local display interpolation property, a result analogous to the first-mentioned, only, of these results holds: see Lemma 1(b)¿

Lemma 1 (`ldi_derl, cldi_ex_interp`).

(a) if a set of rules each satisfies the local display interpolation property, then so does a rule derived from them

(b) if all the derivation rules satisfy the conditional local AD-interpolation property, then the calculus has the interpolation property

4.3 Substitution of congruent occurrences

In [2, Lemmas 3.6, 3.7] the concept of congruent occurrences of some structure Z is used, with substitution for such congruent occurrences. Where two sequents $\mathcal{C}$ and $\mathcal{C}'$ are related by a display postulate, or sequence of them, a particular occurrence of Z in $\mathcal{C}$ will correspond to a particular occurrence of Z in $\mathcal{C}'$, according to the sequence of display postulates used to obtain $\mathcal{C}'$ from $\mathcal{C}$.

This concept looked rather difficult to define and express precisely and formally (we note that the in the notation in [2], $\mathcal{C}[Z/A] \equiv_{AD} \mathcal{C}'[Z/A]$, the meanings of $\mathcal{C}[Z/A]$ and $\mathcal{C}'[Z/A]$ depend on each other).

So we adopted the alternative approach, used successfully in [5]: rather than trying to define $\mathcal{C}'[Z/A]$ we would prove that there exists a sequent (call it $\mathcal{C}'_{Z/A}$) satisfying $\mathcal{C}[Z/A] \equiv_{AD} \mathcal{C}'_{Z/A}$ and safisfying the property that some occurrences of A in $\mathcal{C}'$ are replaced by Z in $\mathcal{C}'_{Z/A}$. This approach turned out to be sufficient for all the proofs discussed in this paper.

In the proofs of cut-elimination in [5] we defined and used a relation `seqrep`, defined as follows.

```
seqrep : "bool => structr => structr => (sequent * sequent) set"
```

Definition 1 (`seqrep`). *$(U, V) \in$ `seqrep` b X Y means that some (or all or none) of the occurrences of X in U are replaced by Y in V; otherwise X and Y are the same; the occurrences of X which are replaced by Y must all be in succedent or antecedent position according to whether b is true or false*

For this we write $U \ {}^{X}\!\!\rightsquigarrow^{Y} \ V$, where the appropriate value of b is understood.

Analogous to Lemma 3.9 we proved the following result about derivation using a set of inference rules satisfying:

- their conclusions do not contain formulae
- their structure variables are distinct
- (Belnap's $C4$ condition) when the conclusion and a premise of a rule both contain a structure variable, then both occurrences are in antecedent or both are in succedent positions

(in Lemma 3.9 this sset of rules is the AD rules).

Lemma 2 (`SF_some_sub`). *Where, for a formula F and structure Z,*

- *"derivable" means using a set of rules satisfying the conditions above*
- *if* `concl` *is derivable from* `prems`
- *if* `concl` ${}^{F}\!\!\rightsquigarrow^{Z}$ `sconcl`

then there exists a list `sprems` *such that*

- `sconcl` *is derivable from* `sprems`
- *if* $\mathtt{prem}_n$ *and* $\mathtt{sprem}_n$ *are corresponding members of* `prems` *and* `sprems`*, then* $\mathtt{prem}_n \ {}^{F}\!\!\rightsquigarrow^{Z} \ \mathtt{sprem}_n$

Note that although we will use this where `?rules` is a set of display postulates, this theorem does not require that the rules have only a single premise.

The proof of this result used some results proved previously for the cut-elimination work [5], notably `extSubs`, which gives, essentially, a constructive expression for `sprems`..

4.4 Local Display Interpolation for Unary Rules

Proposition 3.10 of [2] covers the display postulates, the associativity rule, and the nullary or unary logical introduction rules.

The first case ($(\equiv_D)$, that is, any sequence of display postulates) of [2, Proposition 3.10] is covered by the following result (which holds independent of the choice of set of derivation rules).

Lemma 3 (`bi_lrule_ldi_lem`). *Let rule ρ be a substitution instance of a rule in AD. Then ρ has the LADI property.*

The cases (Id), $(\top R)$ and $(\bot L)$ of Proposition 3.10 would be trivial if it were true that nothing else is display-equivalent to their conclusions; this is not so, but we can use this lemma:

Lemma 4 (`non_bin_lem_gen`). *Assume the derivation rules include the rules for $\neg$. Consider a substitution instance ρ of a rule in AD, whose premise does not contain any comma. Then, if the premise of ρ has an interpolant then so does the conclusion of ρ.*

Since the conclusions of the three nullary rules (Id), $(\top R)$ and $(\bot L)$ clearly themselves have interpolants, Lemma 4 shows they satisfy the extended display interpolation property, and so the rules have the LADI property.

Proposition 1. *The rules (Id), $(\top R)$ and $(\bot L)$ satisfy the LADI property.*

The remaining cases of Proposition 3.10 are the logical introduction rules with a single premise.

For these we use the four lemmas (of which one is shown)

Lemma 5 (`sdA1`). *if $\dfrac{Y' \vdash U}{Y \vdash U}$ is a logical introduction rule, and $W \ {}^{Y}\!\rightsquigarrow^{Y'} W'$, then $\dfrac{W' \vdash Z}{W \vdash Z}$ is derivable (ie, using AD and the logical introduction rules)*

Then from these lemmas we get

Lemma 6 (`seqrep_interpA`). *if $\dfrac{Y' \vdash U}{Y \vdash U}$ is a logical introduction rule, formula variables in Y' also appear in Y, $W \vdash Z \ {}^{Y}\!\rightsquigarrow^{Y'} W' \vdash Z'$ (in antecedent positions), and I is an interpolant for $W' \vdash Z'$, then I is also an interpolant for $W \vdash Z$*

Finally we get the following result which gives Proposition 3.10 for single premise logical introduction rules.

Proposition 2 (`logA_ldi`). *if $\dfrac{Y \vdash U}{F \vdash U}$ (F a formula) is a logical introduction rule where the formula variables in Y are also in F, then that rule satisfies the LADI property*

This last result requires the use of `SF_some_sub` (above). We have analogous results for a logical introduction rule for a formula on the right.

4.5 Binary Additive Logical Introduction Rules

We now discuss extending the results for unary logical introduction rules to the binary rules in the additive form; that is, where the rule contains a single structure variable which appears uniformly in the premises and conclusion.

This involved, first, defining an analogue of `seqrep`, which we called `lseqrep`. As with `seqrep`, we use the notation $U\ ^{Y}\!\!\rightsquigarrow^{Ys}\ Us$

```
lseqrep : "bool => structr => structr list => (sequent * sequent list) set"
```

Definition 2 (`lseqrep`). *$(U, Us) \in$ `lseqrep` $b\ Y\ Ys$ means that there is some occurrence of Y in U such that the nth member of Us is obtained from U by changing the occurrence of Y to the nth member of Ys*

Note that, in contrast to `seqrep`, this definition involves exactly one occurrence of Y in U.

Thus we have a result `mextSubs` which is analogous to `extSubs`, a complicated result which gives, constructively, the `spremss` of the following theorem.

From there we proved `SF_some1sub`, analogous to `SF_some_sub`. In this case the rules must satisfy a slightly stricter set of requirements:

- their conclusions do not contain formulae
- each premise contains the same structure variables, in antecedent positions and in succedent positions, as the conclusion
- the structure variables of the conclusion and of each premise are distinct

Lemma 7 (`SF_some1sub`). *Where, for a formula F and list Z of structures,*

- *"derivable" means using a set of rules satisfying the conditions above*
- *if `concl` is derivable from `prems`*
- *if `concl` $^{F}\!\!\rightsquigarrow^{Zs}$ `sconcls`*

then there exists `spremss` (this is a list of lists of sequents) where

- *for each $\texttt{prem}_n$ in `prems`, let $\texttt{sprems}_n$ be the corresponding member of `spremss`, then $\texttt{prem}_n\ ^{F}\!\!\rightsquigarrow^{Zs}\ \texttt{sprems}_n$*
- *each member of `sconcls` is derivable from the corresponding member of each list in `spremss`*

Then, corresponding to Lemma 5 (`sdA1`) in §4.4, we have a lemma `msdA1` (and its three more counterparts). This is like Lemma 5 except that, in its statement, Y' and W' can be lists.

Then, corresponding to Lemma 6 (`seqrep_interpA`), we have the following lemma: again, the difference is that in the statement of Lemma 6, we replace Y' by a list of structures and $W' \vdash Z'$ by a list of sequents.

Lemma 8 (`lseqrep_interpA`). *if a logical introduction rule has conclusion $Y \vdash U$ and premises $Y_i \vdash U$ for $Y_i \in Ys$, formula variables in each Y_i also appear in Y, $W \vdash Z\ ^{Y}\!\!\rightsquigarrow^{Ys}\ Ss$ (in antecedent positions), and for each $S_i = W_i \vdash Z_i \in Ss$ there exists an interpolant, then there exists an interpolant for $W \vdash Z$*

There are two major cases in the proof: all the sequents W_i are the same, or all the sequents Z_i are the same. This is because the relation S $\overset{Z}{\rightsquigarrow}^{Zs}$ Ss means that there is exactly one location in S where the Ss differ from S. In those cases the proof uses the conjunction or disjunction, respectively, of a list of interpolants. Of course this idea is taken from the proof of [2, Theorem 3.10].

From there we get the result `mlogA_ldi`, analogous to `logA_ldi`, which basically says that additive logical rules satisfy the local display interpolation property. Analogous results for a logical introduction rule for a formula on the right are `lseqrep_interpS` and `mlogS_ldi`.

Proposition 3 (`mlogA_ldi`). *if a logical introduction rule has conclusion $F \vdash U$ (F a formula), and premises $Y_i \vdash U$ for $Y_i \in Ys$, and formula variables in each Y_i also appear in Y, then that rule satisfies the LADI property*

5 Structural Rules

At this point we have a general modus operandi for proving local display interpolation for a given rule ρ, with premises ps_ρ and conclusion c_ρ: identify a relation rel such that

(a) $(ps_\rho, c_\rho) \in rel$
(b) whenever $c \equiv_{AD} c_\rho$, we can find a list ps (often got from sequents in ps_ρ using the same sequence of display postulates which get c from c_ρ) such that $p \equiv_{AD} p_\rho$ for each $p \in ps$ and corresponding $p_\rho \in ps_\rho$
(c) whenever $(ps, c) \in rel$, c is derivable from ps (not used except to prove (d))
(d) whenever $(ps, c) \in rel$, and each $p \in ps$ has an interpolant, then c has an interpolant (proof of this will normally use (c))

5.1 Local Display Interpolation for Unit-Contraction, Contraction

This is relatively easy for the unit-contraction rule: the relation rel is given by: $(p, c) \in rel$ if p is obtained from c by deleting, somewhere in c, some $\#^n\emptyset$, and we get (b) using roughly the same sequence of display postulates.

Lemma 9 (`ex_box_uc`). *if sequent Cd is obtained from C by deleting one occurrence of some $\#^n\emptyset$, and if $Cd' \rightarrow^*_{AD} Cd$, then there exists C', such that $C' \rightarrow^*_{AD} C$, and Cd' is obtained from C' by deleting one occurrence of $\#^n\emptyset$.*

The proof of this required a good deal of programming repetitive use of complex tactics similar to (but less complex than) those described in §5.2.

The following lemma gives (c) of the general proof method above.

Lemma 10 (`delI_der`). *If $(p, c) \in rel$ (defined above), and if the derivation rules include AD and the unit contraction rules, then c is derivable from p*

Proposition 4 (`ldi_ila, ldi_ils`). *The unit contraction rules satisfy LADI.*

For the case of contraction, we defined a relation `mseqctr`: $(C, C') \in$ `mseqctr` means that C' is obtained from C, by contraction of substructures (X, X) to X. Contractions may occur (of different substructures) in several places or none.

Lemma 11 (`ex_box_ctr`). *if sequent Cd is obtained from C by contraction(s) of substructure(s), and if $Cd' \rightarrow^*_{AD} Cd$, then there exists C', such that $C' \rightarrow^*_{AD} C$, and Cd' is obtained from C' by substructure contraction(s).*

The proof of `ex_box_ctr` is a little more complex that that for unit-contraction, because (for example) when $X; Y \vdash Z \equiv_{AD} X \vdash Z; \#Y$, and $X; Y \vdash Z$ is obtained by contracting $(X; Y); (X; Y) \vdash Z$, we need to show $(X; Y); (X; Y) \vdash Z \equiv_{AD} X; X \vdash Z; \#(Y; Y)$.

Lemma 12 (`ctr_der`). *If $(p, c) \in$ `mseqctr` (defined above), and if the derivation rules include AD and the left contraction rule, then c is derivable from p*

Proposition 5 (`ldi_cA`). *The left contraction rule satisfies the LADI property.*

5.2 Lemma 4.2 (Deletion Lemma)

For weakening or unit-weakening, it is more difficult: a sequence of display postulates applied to the conclusion $X; \emptyset \vdash Y$ may give $\emptyset \vdash Y; \#X$, so the same or similar sequence cannot be applied to the premise $X \vdash Y$.

For this situation we need Lemma 4.2 (Deletion Lemma): this result says that for F a formula sub-structure occurrence in C, or $F = \emptyset$, and $C \rightarrow^*_{AD} C'$, then (in the usual case) $C \setminus F \rightarrow^*_{AD} C' \setminus F$, where $C \setminus F$ and $C' \setminus F$ mean deleting only particular occurrence(s) of F in C, and deleting the *congruent* (corresponding) occurrence(s) of F in C', where congruence is determined by the course of the derivation of C' from C.

We did not define congruent occurrences in this sense: see the general discussion of this issue in §4.3. We thought it would be easier to define and use a relation `seqdel`, where $(C, C') \in$ `seqdel` Fs means that C' is obtained from C by deleting one occurrence in C of a structure in the set Fs.

Then we proved the following result about deletion of a formula:

Lemma 13 (`deletion`). *Let F be a formula or $F = \emptyset$. If sequent Cd is obtained from C by deleting an occurrence of some $\#^i F$, and if $C \rightarrow^*_{AD} C'$, then either*

(a) there exists Cd', such that $Cd \rightarrow^*_{AD} Cd'$, and Cd' is obtained from C' by deleting an occurrence of some $\#^j F$, or
(b) C' is of the form $\#^n F \vdash \#^m(Z_1; Z_2)$ or $\#^m(Z_1; Z_2) \vdash \#^n F$, where $Cd \rightarrow^*_{AD} (Z_1 \vdash \#Z_2)$, or $Cd \rightarrow^*_{AD} (\#Z_1 \vdash Z_2)$

Thus the premise is that Cd is got from C by deleting instance(s) of the substructure formula F, possibly with some # symbols. The main clause of the result says that there exists Cd' (this corresponds to $C' \setminus F$ in [2]) which is got from Cd by deleting instance(s) of $\#^n F$ (for some n), but there is also an exceptional case where $\#^n F$ is alone on one side of the sequent.

The proof of this result required considerable ML programming of proof tactics. The file `Del.ML` is devoted to these tactics and the intermediate results proved for the proof of Lemma 4.2.

When we get cases as to the last rule used in the derivation $C \rightarrow^*_{AD} C'$, this gives 13 possibilities ("main cases").

For each rule there are two main cases for the shape of the sequent after the preceding rule applications: in the first, $\#^n F$ appears in $\#^n F, Z$ or $Z, \#^n F$ and so could be deleted (F is "delible"), and in the second, the relevant occurrence of $\#^n F$ is the whole of one side of the sequent.

Then where, in the case of the associativity rule for example, the sequent which is $(X; Y); Z \vdash W$ (instantiated) has F delible, $\#^n F$ may be equal to X, Y or Z, or may be delible from X, Y, Z or W. (Certain further cases, such as that $\#^n F$ is $(X; Y)$, get eliminated automatically using results such as `stars_Sf_not_Comma`, below). The tactics `dvitacs` have been written to provide one set of tactics which handle all these cases. The key component is the recursive tactic `sdvitac` which searches for a way of showing that F is delible from a given sequent (say $X; (Y; Z) \vdash W$, in the above example). Without the possibility of programming a tactic of this sort in Standard ML, each of these seven cases, and a similar (less numerous) set of cases for each of the other 12 main cases, would require its own separate proof.

For the second case, where $\#^n F$ is equal to one side of the sequent (W in the above example), a variety of tactics is required: for those display rules which move the comma from one side to the other the tactics `mdiatacs` works for all, but the other cases have to be done individually.

The proof threw up a number of (logically) trivial cases which nonetheless needed particular results to be included as lemmas to be used automatically in simplification, such as that we cannot have $X; Y = \#^n F$, for F a formula.

We then proved this result for $F = \emptyset$ instead of a formula, to give a theorem `deletion_I`; the changes required in the proof were trivial.

5.3 Local Display Interpolation for Unit-Weakening and Weakening

To handle weakening in a similar way, we considered two separate rules, one to weaken with instances of $\#^n \emptyset$ and one to change any instance of $\emptyset$ to any formula. Thus, where $Y_\emptyset$ means a structure like Y but with every formula or structure variable in it changed to $\emptyset$, a weakening is produced as shown:

$$X \vdash Z \Longrightarrow X, Y_\emptyset \vdash Z \Longrightarrow X, Y \vdash Z$$

We first consider the second of these, replacing any instance of $\emptyset$ with a structural atom, that is, a formula or a structure variable which are atomic so far as the structure language is concerned.

We use the relation `seqrepI str_atoms`: $(c, p) \in$ `seqrepI str_atoms` means that some occurrences of $\emptyset$ in p are changed to structural atoms in c.

Lemma 14 (`ex_box_repI_atoms`). *If sequent C is obtained from Cd by replacing $\emptyset$ by structural atoms, and if $C' \rightarrow^*_{AD} C$, then there exists Cd', such that $Cd' \rightarrow^*_{AD} Cd$, and C' is obtained from Cd' by replacing $\emptyset$ by structural atoms.*

For this relation, property (b) was quite easy to prove, since exactly the same sequence of AD-rules can be used.

We proved that there are derived rules permitting replacing instances of $\emptyset$ by anything, and this gave us that such rules, where the replacement structure is a formula or structure variable, have the the local display interpolation property.

Lemma 15 (`seqrepI_der`). *If the derivation rules include weakening and unit-contraction, and $(c, p) \in$ `seqrepI` Fs, ie some occurrences of $\emptyset$ in p are replaced by anything in c, then c is derivable from p.*

The next lemma gives the LADI property, not for a rule of the system, but for inferences $([p], c)$ where $(c, p) \in$ `seqrepI str_atoms`.

Proposition 6 (`ldi_repI_atoms`). *Where $(c, p) \in$ `seqrepI str_atoms`, ie some occurrences of $\emptyset$ in p are replaced by structural atoms in c, $([p], c)$ has the LADI property.*

Next we consider the structural rules allowing insertion of $\#^n\emptyset$.

We use the variant of the theorem `deletion` (see §5.2) which applies to deletion of $\emptyset$ rather than of a formula.

Then we show that inserting occurrences of anything preserves derivability.

Lemma 16 (`seqwk_der`). *If the derivation rules include weakening, and $(c, p) \in$ `seqdel`Fs, c is obtained from p by a weakening of a substructure, then c is derivable from p.*

Then we need the result that such rules satisfy the local display interpolation property. In this case, though, where a sequent containing $\emptyset$ is rearranged by the display postulates such that the $\emptyset$ is alone on one side (such as where $X \vdash Y; \emptyset$ is rearranged to $X; \#Y \vdash \emptyset$), then the LADI property requires using the derivability of $X \vdash Y$ rather than the fact that $X \vdash Y$ satisfies LADI. Thus we can prove only the conditional local display interpolation property.

Proposition 7 (`ldi_wkI`). *Provided the conclusion is derivable, and assuming that the derivation rules include weakening, unit contraction, $(\emptyset \vdash \top)$ and $(\bot \vdash \emptyset)$, a rule allowing $\#^n\emptyset$-weakening of a substructure satisfies the conditional LADI property.*

From here we define a set of rules called `ldi_rules`, which does not contain the weakening rules or the multiplicative binary logical rules, but does contain the relations of Propositions 7 and 6. We have that all of its rules satisfy the conditional LADI property, so the system has interpolants. We show this gives a deductive system equivalent to the given set of rules `rlscf`, which system therefore also has interpolants. Details are similar to the derivation of Theorem 3.

Theorem 2 (`rlscf_interp`). *The system of substitutable rules `rlscf` satisfies display interpolation*

6 Binary Multiplicative Logical Introduction rules

As discussed above, these were handled, indirectly, using the corresponding binary additive logical introduction rules, and weakening. However the possibility arises of a system which doesn't have the weakening rules and uses the multiplicative rules. So in this section we deal with these rules directly.

We dealt with these rules in two stages — firstly, weakening in occurrences of $\#^n\emptyset$, then changing any occurrence of $\emptyset$ to any structural atom, as shown below.

$$\frac{\dfrac{X \vdash A}{X, Y_\emptyset \vdash A}\, wk^*_\emptyset \quad \dfrac{Y \vdash B}{X_\emptyset, Y \vdash B}\, wk^*_\emptyset}{X, Y \vdash A \wedge B}\,(\texttt{ands\_rep})$$

Here $X_\emptyset$ and $Y_\emptyset$ mean the structures X and Y, with each structural atom (formula or uninterpreted structure variable) replaced by $\emptyset$. The first stage, the inferences labelled $wk^*_\emptyset$ above, are obtained by repeatedly weakening by occurrences of $\#^n\emptyset$ in some substructure. The second stage (for which we define the relation `ands_rep`), consists of changing the $X_\emptyset$ of one premise, and the $Y_\emptyset$ of the other premise, to X and Y respectively. For the second of these stages, then, when any sequence of display postulates is applied to $X, Y \vdash A \wedge B$, the same sequence can be applied to the two premises, $X, Y_\emptyset \vdash A$ and $X_\emptyset, Y \vdash B$. This simplifies the proof of local display interpolation for these rules.

For the first stage we proceed as described for §5.3, except that we have the result for a system containing the unit-weakening rules rather than a general weakening rule, thus getting the theorem

Proposition 8 (`ldi_wkI_alt`). *Provided the conclusion is derivable, and assuming that the derivation rules include unit weakening, unit contraction, and the left and right introduction rules for $\top$ and $\bot$, a rule allowing $\#^n I$-weakening of a substructure satisfies the conditional LADI property.*

The second stage consists of the rule shown as `ands_rep` in the diagram. Considering the four points at the start of §5, since any display postulate applied to the conclusion can be appplied to the premises, we need to define a suitable relation between conclusion and premises which is preserved by applying any display postulate to them. For this we define a relation `lseqrepm` between sequents, analgous to `lseqrep` discussed in §4.5:

```
lseqrepm      :: "(structr * structr list) set =>
    bool => [structr, structr list] => (sequent * sequent list) set"
```

Definition 3 (`lseqrepm, repnI_atoms`).

(a) $(U, Us) \in$ *`lseqrepm`* *orel b Y Ys means that (as for* `lseqrep`*) there is one occurrence of Y in U which is replaced by the nth member of Ys in the nth member of Us; this occurrence is at an antecedent or succedent position,*

according to whether b is `True` *or* `False`*. However elsewhere in U, each structural atom A in U is replaced by the nth member of As in the nth member of Us, where $(A, As) \in orel$;*

(b) $(A, As) \in$ `repnI_atoms` *iff one of the As is A and the rest of the As are $\emptyset$.*

We use `lseqrepm` only with $orel =$ `repnI_atoms`. For example, for the ($\wedge$R) rule, we use `lseqrepm repnI_atoms True` $(A \wedge B)$ $[A, B]$ as the relation rel of the four points at the start of §5. We get the following lemmas.

Lemma 17 (`repm_some1sub`). *Whenever Y is a formula, and $(U, Us) \in$ `lseqrepm` $orel\ b\ Y\ Ys$, and U is manipulated by a display postulate (or sequence of them) to give V, then the Us can be manipulated by the same display postulate(s) to give Vs (respectively), where $(V, Vs) \in rel$*

The following lemmas refer to derivation in the system containing the `ands_rep` rule (not the regular ($\wedge$R) rule), and also unit-weakening and unit-contraction.

Lemma 18 (`ands_mix_gen`). *Whenever $(V, Vs) \in rel$, then V can be derived from the Vs.*

As with Lemma 8, this lemma relies on taking the conjunction or disjunction of interpolants of premises. So the next two results require a deductive system containing the `ands_rep` and `ora_rep` rules, and also the ($\vee$R) and ($\wedge$L) rules.

Lemma 19 (`lseqrepm_interp_andT`). *Whenever $(V, Vs) \in rel$, then we can construct an interpolant for V from interpolants for the Vs*

Proposition 9 (`ldin_ands_rep`). *The rule* `ands_rep` *satisfies LADI.*

We then defined a set of rules called `ldi_add` which contains the rules of Definition 2.4 and Figures 1 and 3 of [2], except the weakening rule and the binary logical rules ($\vee L$) and ($\wedge R$), with the rule `ands_rep` (see diagram above) and a corresponding rule `ora_rep` included. Note that these latter rules are not closed under substitution; therefore we defined `ldi_add` to consist of the rules, including their substitution instances where relevant. (Note that, as mentioned earlier, our formalisation had not included the connective $\rightarrow$, or any rules for it). Meanwhile we also defined the set `rlscf_nw` of substitutable rules, the rules of Definition 2.4 and Figures 1 and 3 of [2], except weakening.

Lemma 20 (`ldi_add_equiv`). *Let C be the calculus consisting of the rules of Definition 2.4 and Figures 1 and 3 of [2] minus the weakening rule. Then the calculi $C \setminus \{(\wedge\mathrm{R}), (\vee\mathrm{L})\}$ and $C \cup \{$*`ands_rep`*,* `ora_rep`*$\}$ are deductively equivalent.*

Theorem 3 (`ldi_add_interp, rlscf_nw_interp`).

(a) the system `ldi_add` *satisfies display interpolation*
(b) the system of substitutable rules `rlscf_nw` *satisfies display interpolation*

Proof. We have all rules in `ldi_add` satisfying at least the conditional local display interpolation property (`ldi_add_cldin`). By `cldin_ex_interp`, this gives us that the system `ldi_add` satisfies display interpolation `ldi_add_interp`, and so therefore does the equivalent system of substitutable rules `rlscf_nw`.

7 Conclusions

TO BE WRITTEN

References

1. N D Belnap. Display logic. *J. of Philosophical Logic*, 11:375–417, 1982.
2. James Brotherston & Rajeev Goré. Craig Interpolation in Displayable Logics In *Proceedings of TABLEAUX-20*, LNAI, pages 88–103. Springer, 2011.
3. Nicolas Caniart. MERIT: an interpolating model-checker. In *Proceedings of CAV-22*, volume 6174 of *LNCS*, pages 162–166. Springer, 2010.
4. William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.
5. J E Dawson and R Goré. Formalised Cut Admissibility for Display Logic. In Proc. TPHOLS'02, LNCS 2410, 131–147, Springer, 2002.
6. J E Dawson and R Goré. Generic Methods for Formalising Sequent Calculi Applied to Provability Logic. In Proc. Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2010), LNCS 6397, 263-277.
7. Solomon Feferman. Harmonious logic: Craigs interpolation theorem and its descendants. *Synthese*, 164:341–357, 2008.
8. R Goré. Machine Checking Proof Theory: An Application of Logic to Logic. Invited talk, Third Indian Conference on Logic and Applications, Chennai, January 2009.
9. Kenneth L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *Proceedings of TACAS-14*, volume 4963 of *LNCS*, pages 413–427, 2008.

A Isabelle text of selected definitions and theorems

A.1 Isabelle text of definitions and basic lemmas

```
ldi_derl :
  "[| ALL psc:?pscs. ldi ?lrules ?drules psc; (?ps, ?c) : derl ?pscs |] ==>
          ldi ?lrules ?drules (?ps, ?c)"
cldi_ex_interp :
  "[| (ALL psc : ?pscs. cldi ?lrules ?drules psc);
         ?c : derivableR ?drules {} |] ==> edi ?lrules ?drules ?c"
```

A.2 Isabelle text of lemmas for §4.4, unary logical rules

Belnap's C4 property is used in the proof of cut-elimination, it being one of the properties that structural rules must satisfy for Belnap's cut-elimination theorem to apply to a Display Calculus.

Here, `seqSVs'` b *seq* is a list of the structural variables in succedent (if $b = True$) or antecedent (if $b = False$) position in a sequent *seq*

```
C4_def : "C4 ?rule == ALL prem:set (premsRule ?rule).
  ALL b. ALL s:set (seqSVs' b (conclRule ?rule)).
    s ~: set (seqSVs' (~ b) prem)"

SF_some_sub :
   "[| ALL (ps, c):PC ‘ ?rules. ~ seqCtnsFml c & distinct (seqSVs c);
         ALL r:?rules. C4 r; (?prems, ?concl) : derl (PC ‘ rulefs ?rules);
         (?concl, ?sconcl) : seqrep ?sa (Structform ?fml) ?Z |]
      ==> EX sprems.
         (?prems, sprems) : seqreps ?sa (Structform ?fml) ?Z &
         (sprems, ?sconcl) : derl (PC ‘ rulefs ?rules)"
```

Lemma 3 is actually proved more generally. We define an *invertible set* of rules to be a set of unary rules such that the inverse of any of them is derivable from the substitution instances of them. The set `aidps` of rules satisfies this property (theorem `inv_rules_aidps`)

```
inv_rules_def : "inv_rules ?rules == ALL r:?rules. EX p.
  premsRule r = [p] & ([conclRule r], p) : derl (PC ‘ rulefs ?rules)"

inv_rules_aidps : "inv_rules aidps"
```

Then Lemma 3 actually holds for any invertible set of rules.

```
bi_lrule_ldi_lem : "[| ?r : rulefs ?lrules; inv_rules ?lrules |] ==>
  ldi ?lrules ?drules (PC ?r)"
```

```
non_bin_lem_gen "[| aidps <= ?drules; {nota, nots} <= ?drules;
        (?ps, ?concl) : PC ‘ rulefs aidps;
        ALL p:set ?ps. ~ seqHasComma p;
        ALL p:set ?ps. Ex (interp ?drules p) |] ==>
     Ex (interp ?drules ?concl)"

tS_ldi : "ldi aidps rlscf ([], $I |- T)"
fA_ldi : "ldi aidps rlscf ([], F |- $I)"
idf_ldi : "ldi aidps rlscf ([], ?A |- ?A)"

sdA1 : "[| ALL U. ([$?Y' |- $U], $?Y |- $U) : ?logI; strIsLog ?W;
          (True, ?W, ?W') : strrep ?Y ?Y' |] ==>
      ([$?W' |- $?Z], $?W |- $?Z) : derl (?logI Un PC ‘ rulefs aidps)"

seqrep_interpA : "[| ALL U. ([$?Y' |- $U], $?Y |- $U) : ?logI;
     seqIsLog ($?W |- $?Z); strFVPPs ?Y' <= strFVPPs ?Y;
     ($?W |- $?Z, $?W' |- $?Z') : seqrep False ?Y ?Y';
     ?logI <= PC ‘ rulefs ?rules; aidps <= ?rules;
     interp ?rules ($?W' |- $?Z') ?intp |] ==>
  interp ?rules ($?W |- $?Z) ?intp"

logA_ldi : "[| ALL (ps, c):PC ‘ aidps. ~ seqCtnsFml c & distinct (seqSVs c);
         Ball aidps C4; strFVPPs ?Y <= fmlFVPPs ?fml; seqIsLog (?fml |- $?U);
         ALL U. ([$?Y |- $U], ?fml |- $U) : ?logI;
         ?logI <= PC ‘ rulefs ?rules; aidps <= ?rules |] ==>
     ldi aidps ?rules ([$?Y |- $?U], ?fml |- $?U)"
```

We have results analogous top the above for a logical introduction rule for a formula on the right, are `seqrep_interpS` and `logS_ldi`.

A.3 Isabelle text of lemmas for §4.5, additive binary logical rules

```
SF_some1sub : "[| ALL (ps, c):PC ‘ ?rules.
    ~ seqCtnsFml c & distinct (seqSVs c) & seqIsLog c &
    Ball (set ps) seqIsLog &
    (ALL p:set ps.  distinct (seqSVs p) &
      (ALL b. set (seqSVs' b p) = set (seqSVs' b c)));
    Ball ?rules C4; (?prems, ?concl) : derl (PC ‘ rulefs ?rules);
    (?concl, ?sconcls) : lseqrep ?sa (Structform ?fml) ?Zs |] ==>
  EX spremss. (?prems, spremss) : lseqreps ?sa (Structform ?fml) ?Zs &
    (ALL n<length ?Zs. (map (%l. l ! n) spremss, ?sconcls ! n) :
      derl (PC ‘ rulefs ?rules))"

lseqrep_interpA : "[| rlscf <= ?rules;
   ALL U. (map (%Y'. $Y' |- $U) ?Ys, $?Y |- $U) : ?logI;
   seqIsLog ($?W |- $?Z); ALL Y':set ?Ys. strFVPPs Y' <= strFVPPs ?Y;
   ($?W |- $?Z, ?Ss') : lseqrep False ?Y ?Ys;
```

```
   ?logI <= PC ‘ rulefs ?rules; aidps <= ?rules;
   ALL S:set ?Ss'. Ex (interp ?rules S) |] ==>
 Ex (interp ?rules ($?W |- $?Z))"

mlogA_ldi : "[| ALL (ps, c):PC ‘ aidps.  ~ seqCtnsFml c &
    distinct (seqSVs c) & seqIsLog c & Ball (set ps) seqIsLog &
    (ALL p:set ps.  distinct (seqSVs p) &
      (ALL b. set (seqSVs' b p) = set (seqSVs' b c)));
    Ball aidps C4; seqIsLog (?fml |- $?U);
    ALL U. (map (%Y'. $Y' |- $U) ?Ys, ?fml |- $U) : ?logI;
    ALL Y:set ?Ys. strFVPPs Y <= fmlFVPPs ?fml;
    ?logI <= PC ‘ rulefs ?rules; aidps <= ?rules; rlscf <= ?rules |] ==>
   ldi aidps ?rules (map (%Y'. $Y' |- $?U) ?Ys, ?fml |- $?U)"
```

A.4 Isabelle text of lemmas for §5.1, Unit-Contraction and Contraction

The set `stars` S is the set of all structures which consist of the structure S preceded by any number of occurrences of `Star` (ie, of # symbols).

The relation `seqdel (stars I)` will be the relation used for unit-contraction, where $(p, c) \in$ `seqdel` Fs if p is obtained from c by deleting (in any number of places) a structure in Fs.

```
ex_box_uc : "[| ?atom = I; (?C, ?Cd) : seqdel (stars ?atom);
    ?Cd : derivableR aidps {?Cd'} |] ==>
   EX C'.  (C', ?Cd') : seqdel (stars ?atom) &
     C : derivableR aidps {?C'}"
```

The rules for replacing $(\emptyset, X)$ by X on the left and the right of the $\vdash$ are `ila` and `ils`, and the left contraction rule is `cA`.

```
delI_der : "[| (?Y, ?Y') : strdel (stars I); aidps <= ?rules;
    {ila, ils} <= ?rules |] ==>
   {([$?X |- $?Y], $?X |- $?Y'), ([$?Y |- $?X], $?Y' |- $?X)} <=
     derl (PC ‘ rulefs ?rules)"

ldi_ila : "[| aidps <= ?rules; {ila, ils} <= ?rules |] ==>
   ldi aidps ?rules (PC ila)"

ctr_der : "[| (?Y, ?Y') : mstrctr; aidps <= ?rules; {cA} <= ?rules |] ==>
  {([$?X |- $?Y], $?X |- $?Y'), ([$?Y |- $?X], $?Y' |- $?X)} <=
    derl (PC ‘ rulefs ?rules)" :

ldi_cA : "[| aidps <= ?rules; {cA} <= ?rules |] ==> ldi aidps ?rules (PC cA)"
```

Note that the theorem `ctr_der` needs to be applied twice (once for each side of the $\vdash$) to give the result in the main text.

A.5 Isabelle text of lemmas for §5.2, the Deletion Lemma

The proof threw up a number of (logically) trivial cases which nonetheless needed particular results to be included as lemmas to be used automatically in simplification, such as:

```
stars_Sf_not_Comma : "Comma ?X ?Y ~: stars (Structform ?fml)"
Stars_Sf_ne_Comma : "Comma ?X ?Y ~= funpow Star ?n (Structform ?fml)"
Stars_eq_Comma_iff : "(Comma ?X ?Y = funpow Star ?n (Comma ?U ?V)) =
    (?n = 0 & ?X = ?U & ?Y = ?V)"

deletion :
   "[| ?atom = Structform ?fml; (?C, ?Cd) : seqdel ?pn (stars ?atom);
         ?C' : derivableR aidps {?C} |]
      ==> (EX Cd'.
          (?C', Cd') : seqdel ?pn (stars ?atom) &
          Cd' : derivableR aidps {?Cd}) |
      (EX n m Z1 Z2.
          ?C' = ($(funpow Star n ?atom) |- $(funpow Star m (Comma Z1 Z2))) &
          (if odd m then $Z1 |- * $Z2 else * $Z1 |- $Z2)
          : derivableR aidps {?Cd} |
          ?C' = ($(funpow Star m (Comma Z1 Z2)) |- $(funpow Star n ?atom)) &
          (if even m then $Z1 |- * $Z2 else * $Z1 |- $Z2)
          : derivableR aidps {?Cd})" : Thm.thm
```

A.6 Isabelle text of lemmas for §5.3, Unit-Weakening and Weakening

```
ex_box_repI_atoms :
    "[| (?C, ?Cd) : seqrepI str_atoms; ?C : derivableR aidps {?C'} |] ==>
    EX Cd'. (?C', Cd') : seqrepI str_atoms & ?Cd : derivableR aidps {Cd'}"

seqrepI_der : "[| (?S', ?S) : seqrepI ?Fs; aidps <= ?rules;
    {ila, ils, mra} <= ?rules |] ==>
  ([?S], ?S') : derl (PC ` rulefs ?rules)"

ldi_repI_atoms : "[| aidps <= ?rules; {ila, ils, mra} <= ?rules;
             (?c, ?p) : seqrepI str_atoms |] ==>
    ldi aidps ?rules ([?p], ?c)"

seqwk_der : "[| (?S', ?S) : seqdel ?Fs;
      aidps <= ?rules; {mra} <= ?rules |] ==>
    ([?S], ?S') : derl (PC ` rulefs ?rules)"

ldi_wkI : "[| aidps <= ?rules; {mra, ila, ils, tS, fA} <= ?rules;
            (?c, ?p) : seqdel (stars I) |] ==>
          cldi aidps ?rules ([?p], ?c)"
```

A.7 Isabelle text of lemmas for §6, Binary Multiplicative Logical Introduction rules

```
wkI_der : "[| (?Y', ?Y) : strdel (stars I); aidps <= ?rules;
         {iila, iils} <= ?rules |] ==>
   {([$?X |- $?Y], $?X |- $?Y'), ([$?Y |- $?X], $?Y' |- $?X)}
      <= derl (PC ‘ rulefs ?rules)"

ldi_wkI_alt : "[| aidps <= ?drules;
     {iila, iils, tS, fA, ila, ils, tA, fS} <= ?drules;
     (?c, ?p) : seqdel (stars I) |] ==>
   cldi aidps ?drules ([?p], ?c)"
```

The following result applies to display postulates satisfying a set of standard set of display postulates properties.

```
dp_props_def : "dp_props (?ps, ?c) =
  (length ?ps = 1 & ~ seqCtnsFml ?c & distinct (seqSVs ?c) & seqIsLog ?c &
  (ALL p:set ?ps. seqIsLog p & distinct (seqSVs p) & ~ seqCtnsFml p &
           (ALL b. set (seqSVs' b p) = set (seqSVs' b ?c))))"

repm_some1sub :
   "[| ALL rule:?rules. dp_props (PC rule); ?As ~= [];
       ([?prem], ?concl) : derl (PC ‘ rulefs ?rules);
       (?concl, ?sconcls) : lseqrepm ?rsa ?sa (Structform ?fml) ?As |]
    ==> EX sprems.
       (?prem, sprems) : lseqrepm ?rsa ?sa (Structform ?fml) ?As &
       (ALL k<length ?As.
   ([sprems ! k], ?sconcls ! k) : derl (PC ‘ rulefs ?rules))"
```

We mention that the proof of `repm_some1sub` involved very considerable effort; it used the following lemma (see the proofs of it in `GRepm.ML`)

```
repm_seq_sub :
   "[| ~ seqCtnsFml ?pat; distinct (seqSVs ?pat);
     (seqSubst ?suba ?pat, ?Ys) : lseqrepm ?rsa ?pn (Structform ?A) ?Xs |]
   ==> EX subys. map (%suby. seqSubst suby ?pat) subys = ?Ys"

ands_mix_gen :
   "[| PC ‘ rulefs aidps <= ?rules; ands_rep <= ?rules;
      PC ‘ rulefs ilrules <= ?rules;
    (?Z, [?X, ?Y]) : lseqrepm repnI_atoms True (Structform (?A && ?B))
      [Structform ?A, Structform ?B] |] ==> ?Z : derrec ?rules {?X, ?Y}"

lseqrepm_interp_andT :
   "[| ands_rep <= ?rules; ora_rep <= ?rules;
         PC ‘ rulefs {anda} <= ?rules; PC ‘ rulefs {ors} <= ?rules;
         PC ‘ rulefs aidps <= ?rules; PC ‘ rulefs ilrules <= ?rules;
```

```
 (?WZ, [?pa, ?pb]) : lseqrepm repnI_atoms True
    (Structform (?A && ?B)) [Structform ?A, Structform ?B];
         ALL S:set [?pa, ?pb]. Ex (interpn ?rules S) |]
      ==> Ex (interpn ?rules ?WZ)"

ldin_ands_rep : "[| (?ps, ?WZ) : ands_rep; PC ‘ rulefs ilrules <= ?drules;
            PC ‘ rulefs aidps <= ?drules; PC ‘ rulefs {ors} <= ?drules;
   PC ‘ rulefs {anda} <= ?drules; ora_rep <= ?drules;
  ands_rep <= ?drules; ?lrules <= aidps |] ==>
ldin ?lrules ?drules (?ps, ?WZ)"

ldi_add_equiv : "(?c : derrec ldi_add {}) = (?c : derivableR rlscf_nw {})"
ldi_add_cldin : "?rule : ldi_add ==> cldin aidps ldi_add ?rule"
ldi_add_interp : "Ball (derrec ldi_add {}) (edin aidps ldi_add)"
rlscf_nw_interp : "Ball (derivableR rlscf_nw {}) (edi aidps rlscf_nw)"
```

Display postulates:

$$X;Y \vdash Z \quad \rightleftarrows_D \quad X \vdash \sharp Y;Z \quad \rightleftarrows_D \quad Y;X \vdash Z$$

$$X \vdash Y;Z \quad \rightleftarrows_D \quad X;\sharp Y \vdash Z \quad \rightleftarrows_D \quad X \vdash Z;Y$$

$$X \vdash Y \quad \rightleftarrows_D \quad \sharp Y \vdash \sharp X \quad \rightleftarrows_D \quad \sharp\sharp X \vdash Y$$

Identity rules:

$$\frac{}{P \vdash P}\,(\mathrm{Id}) \qquad \frac{X' \vdash Y'}{X \vdash Y}\;\; X \vdash Y \equiv_D X' \vdash Y' \;\; (\equiv_D)$$

Multiplicative logical rules:

$$\frac{\emptyset \vdash X}{\top \vdash X}\,(\top\mathrm{L}) \qquad \frac{}{\emptyset \vdash \top}\,(\top\mathrm{R}) \qquad \frac{F;G \vdash X}{F \,\&\, G \vdash X}\,(\&\mathrm{L}) \qquad \frac{X \vdash F \quad Y \vdash G}{X\,;\,Y \vdash F \,\&\, G}\,(\&\mathrm{R})$$

$$\frac{}{\bot \vdash \emptyset}\,(\bot\mathrm{L}) \qquad \frac{X \vdash \emptyset}{X \vdash \bot}\,(\bot\mathrm{R}) \qquad \frac{F \vdash X \quad G \vdash Y}{F \vee G \vdash X\,;\,Y}\,(\vee\mathrm{L}) \qquad \frac{X \vdash F;G}{X \vdash F \vee G}\,(\vee\mathrm{R})$$

$$\frac{\sharp F \vdash X}{\neg F \vdash X}\,(\neg\mathrm{L}) \qquad \frac{X \vdash \sharp F}{X \vdash \neg F}\,(\neg\mathrm{R}) \qquad \frac{X \vdash F \quad G \vdash Y}{F \to G \vdash \sharp X\,;\,Y}\,(\to\mathrm{L}) \qquad \frac{X\,;\,F \vdash G}{X \vdash F \to G}\,(\to\mathrm{R})$$

Additive logical rules:

$$\frac{}{\bot_a \vdash X}\,(\bot_a\mathrm{L}) \qquad \frac{F_i \vdash X}{F_1 \,\&_a\, F_2 \vdash X}\;\; i \in \{1,2\}\,(\&_a\mathrm{L}) \qquad \frac{F \vdash X \quad G \vdash X}{F \vee_a G \vdash X}\,(\vee_a\mathrm{L})$$

$$\frac{}{X \vdash \top_a}\,(\top_a\mathrm{R}) \qquad \frac{X \vdash F \quad X \vdash G}{X \vdash F \,\&_a\, G}\,(\&_a\mathrm{R}) \qquad \frac{X \vdash F_i}{X \vdash F_1 \vee_a F_2}\;\; i \in \{1,2\}\,(\vee_a\mathrm{R})$$

Structural rules:

$$\frac{\emptyset;X \vdash Y}{X \vdash Y}\,(\emptyset\mathrm{C_L}) \qquad \frac{X \vdash Y;\emptyset}{X \vdash Y}\,(\emptyset\mathrm{C_R}) \qquad \frac{X \vdash Y}{\emptyset;X \vdash Y}\,(\emptyset\mathrm{W_L}) \qquad \frac{X \vdash Y}{X \vdash Y;\emptyset}\,(\emptyset\mathrm{W_R})$$

$$\frac{(W;X);Y \vdash Z}{W;(X;Y) \vdash Z}\,(\alpha) \qquad \frac{X \vdash Z}{X;Y \vdash Z}\,(\mathrm{W}) \qquad \frac{X;X \vdash Y}{X \vdash Y}\,(\mathrm{C})$$

Fig. 1. Display calculus proof rules In the display rule $(\equiv_D)$, the relation $\equiv_D$ is the least equivalence containing the relation $\rightleftarrows_D$ given by the display postulates.