Isabelle work on Interpolation for Display Calculi

Jeremy E. Dawson

December 1, 2011

1 Introduction and Acknowledgements

This document describes proofs I did in Isabelle during a visit to Imperial College, London, during November and December 2010.

The purpose of the visit was to explore adapting my previous work in doing proofs in Isabelle of results in Display Logic (notably the Cut Elimination and Strong Normalisation results) to the results of the paper

James Brotherston & Rajeev Goré, Craig Interpolation in Displayable Logics

The paper used was a draft version, with changes made (including to numbering of Lemmas) during my visit, so note that Lemma numbers referenced below may have changed further.

I thank Rajeev Goré and James Brotherston for support during this period, and Imperial College for its support and hospitality.

See §4 for an account of some more work done on this topic in 2011.

2 Definitions and Lemmas

Much of this work built upon the previous Isabelle work for Display Logic, which is described in

Jeremy E. Dawson & Rajeev Goré, Formalised Cut Admissibility for Display Logic, In 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2002), LNCS 2410, 131-147, see `http://users.cecs.anu.edu.au/~jeremy/pubs/cutelim/tphols/final/`

We should emphasize that this is a deep embedding of rules and of the variables in them. 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, as would be shown in a paper describing a display calculus), and an infinite number of substitution instances of these rules.

Thus 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.

Note that this work built on the work for the Display Calculus for Relation Algebras, so the language as defined has relational connectives as well as logical ones. Thus it is sometimes necessary to use the predicate `strIsLog`, which says that a structure contains only logical, not relational, structural connectives. Also note that in that work the symbol $*$ (`Star`), not #, was used as the structural "not" symbol, and the structural boolean binary connective was ',' (`Comma`), not ';'. These usages have necessarily been continued in the work described here.

We define the sets of rules:

- `dps` is the set of six display postulates in Definition 2.5 (Display-equivalence)
- `aidps` is `dps`, their inverses, and the associativity rule (ie, 13 rules)
- `rlscf` is the set of all rules of the logic as shown in Figures 1 and 2, plus `aidps` (currently the rules for implication $\rightarrow$ are not included)

The definition explicitly excludes the inverse of the `assoc` rule, but we proved (as `inv_rules_aidps`) that the inverses of rules in `aidps` are derivable using `aidps`.

2.1 Definitions relating to interpolation poperties

```
interp :: "rule set => sequent => formula => bool"
edi :: "rule set => rule set => sequent => bool"
ldi :: "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. Note that this definition does not include the condition that $X \vdash Y$ be derivable.

`edi` *lrules drules* $(X \vdash Y)$ 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) 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, and `drules` is the set of rules of the logic, then the local AD-interpolation property as defined, for rule (ps, c), is that, if all $p \in ps$ are provable, then `ldi` *lrules drules* (ps, c) holds. Note that, unlike Definition 3.3, our definition of `ldi` is not not conditional on the premises ps being derivable.

See `Igen.thy` for these definitions.

Given the definition of a rule as containing variables which may be instantiated, we need lemmas such as the following. These follow easily from the definition of derivability from a set of rules, which, as noted earlier, involves instantiating those rules.

```
ldi_rulefs_l : "ldi (rulefs ?lrules) = ldi ?lrules"
ldi_rulefs : "ldi ?lrules (rulefs ?drules) = ldi ?lrules ?drules"
edi_rulefs_l : "edi (rulefs ?lrules) = edi ?lrules"
edi_rulefs : "edi ?lrules (rulefs ?drules) = edi ?lrules ?drules"
interp_rulefs : "interp (rulefs ?rules) = interp ?rules"
interp_sub : "interp ?rules ?seq ?intp ==>
  interp ?rules (seqSubst (?fs, ?ss) ?seq) (fmlSubst ?fs ?intp)"
```

These lemmas show that these properties are preserved by taking substitution instances of the given set of rules. The interpolation property is also preserved by substituting the sequent and the interpolating formula.

Lemma 3.4 says that if all rules satisfy the local AD-interpolation property, then the calculus has the interpolation property. We proved

```
ldi_derl :
  "[| ALL psc:?pscs. ldi ?lrules ?drules psc; (?ps, ?c) : derl ?pscs |] ==>
          ldi ?lrules ?drules (?ps, ?c)"
ldi_ex_interp :
  "[| (ALL psc : ?pscs. ldi ?lrules ?drules psc); ?c : derrec ?pscs {} |] ==>
    EX intp. interp ?drules ?c intp" : Thm.thm
```

2.2 Substitution of congruent occurrences; Lemma 3.7

Lemma 3.7 (Substitution Lemma) involves substituting Z for F in two different sequents C and C', where $C \rightarrow^*_{AD} C'$. But this means substituting only for particular occurrence(s) of F in C, and substituting for the *congruent* (corresponding) occurrence(s) of F in C', where congruence is determined by the course of the derivation of C' from C.

We could no doubt define congruent occurrences of a substructure in relation to a derivation of C' from C. However, the work on cut-elimination for display logic uses a related concept (the replacement of "related" occurrences of a formula A, occurring as a substructure, by a structure Y. In the proofs of cut-elimination we defined and used a relation `seqrep`, where $(U, V) \in \texttt{seqrep}\, b\, X\, Y$ 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 (we write $U \ {}^{X}\!\!\rightsquigarrow^{Y}\ V$, in which the appropriate value of b is understood)

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

So we attempted to see whether couching lemmas and theorems in terms of `seqrep`, rather than defining congruence, was adequate for the main results of this paper.

Analogous to Lemma 3.7 we proved the following result:

```
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)" : Thm.thm
```

This says that if `concl` is derivable from `prems`, and if `concl` $^{\texttt{fml}}\!\rightsquigarrow^{Z}$ `sconcl` then there exists `sprems` from which `sconcl` is derivable, and where corresponding members of `prems` and `sprems` are also in the same $^{\texttt{fml}}\!\rightsquigarrow^{Z}$ relation.

There are preconditions about the rules with which the derivations are done (in Lemma 3.7 that is the AD rules): these are that the conclusions of the rules do not contain formulae and their structure variables are distinct, and Belnap's $C4$ condition, that when the conclusion and a premise of a rule both contain a structure variable, then they are both in antecedent or both in succedent positions.

2.3 Proposition 3.8 :

The first case $((\equiv_D))$ of Proposition 3.8 is covered by the following result

```
bi_lrule_ldi : "[| premsRule ?rule ~= [];
       PC (invert ?rule) : derl (PC ‘ rulefs ?lrules) |] ==>
     ldi ?lrules ?drules (PC ?rule)" : Thm.thm
```

For a rule which has premises and whose inverse is derivable using `lrules` satisfies the `ldi` property for `lrules` and `drules` — even if `drules` is empty! So the last few words of the statment of the Proposition ("...in any extension of $\mathcal{D}_0 + (A)$") seem unnecessary.

The cases (Id), $(\top R)$ and $(\bot L)$ of Proposition 3.8 are proved similarly. They 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:

```
non_bin_lem "[| (?ps, ?concl) : PC ‘ rulefs aidps;
        ALL p:set ?ps. ~ seqHasComma p;
        ALL p:set ?ps. Ex (interp rlscf p) |] ==>
    Ex (interp rlscf ?concl)"
```

that is, where `ps` and `c` are the premise(s) and conclusion of a substitution instance of a rule in AD, and the premise(s) do not contain any comma, then if the premise(s) have interpolant(s) then so do the conclusions.

Recall that `aidps` is the set of all AD rules, and `rlscf` is the set of all rules of the calculus.

Then we get the three results

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

The remaining cases of Proposition 3.8 are the logical introduction rules with a single premise.

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

```
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)"
```

that is, 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 using AD and the logical introduction rules.

Then from these lemmas we get

```
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"
```

that is, 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 3.8 for single premise logical introduction rules.

```
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)"
```

This last result requires the use of `SF_some_sub` (above). Analogous results for a logical introduction rule for a formula on the right are `seqrep_interpS` and `logS_ldi`.

In the proof here I was able to make use of some results proved previously for the cut-elimination work, notably `extSubs`.

I think it would not be much more difficult to prove similar results for logical introduction rules with more than one premise provided there is a single structure variable on the other side, ie the additive rather than multiplicative rules. However the appropriate analogue of `extSubs` would need to be proved, as well as the analogues of the results proved during this visit.

Although noting that there are good reasons for using the rules as given in Figure 1, I spent a small amount of time exploring the possibility of adapting these results to the case of two-premise rules: the result `strrep_mderA` and `strrep_mderS` were part of this work. [1]

[1]Subsequently I have spent more time on this, see §4

2.4 Lemma 4.2 (Deletion Lemma)

Lemma 4.2 (Deletion Lemma): this result says that for F a formula substructure occurrence in C, or $\emptyset$, and $C \rightarrow^*_{AD} C'$, then (in the usual case) $C \setminus F \rightarrow^*_{AD} C' \setminus F$. But this means 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 §2.2.

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 some (this maybe none or all) occurrences in C of structures in the set *Fs*.

Then we proved the following result:

```
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
```

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

Thus the premise is that Cd is got from C by deleting instance(s) of the substructure formula F (`fml` in the code), possibly with some # symbols.

The main clause of the result says that there exists Cd' (this corresponds to $C' \setminus F$ in the paper) 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 the whole of 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, the second being the case where, after the preceding rule applications, the relevant occurrence of $\#^n F$ is the whole of one side of the sequent.

Then where, 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) trivially easy 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)"
```

Note that we did not prove this result for F being $\emptyset$; this should be no more difficult than for F a formula, except that some of our previous lemmas are for the case where F is a formula. [2]

3 Conclusion

In this work we attempted to reuse, wherever possible, the work we had previously done in Isabelle for proving cut-elimination for display logic, since that previous work was a significant part of a three-year project. As it turned out, a considerable amount of that work was relevant and used in this work on interpolation.

In some aspects the previous work influenced this present work, which would have been done differently were it being done from scratch. This includes the choice of names for the various structural / logical operators (for example, the structural boolean connectives being `Star` and `Comma`, for '#' and ';'. More significantly we were influenced by the previous work (and its successful conclusion) to avoid formally defining congruent occurrences of a substructure.

We proposed avoiding a definition of congruent occurrences of some substructure in the context of a derivation, as this could be quite complicated; hopefully it would not be necessary for completing the proofs, as was the case in the work on cut-elimination for display logics, where the relation `seqrep` was adequate to express the idea of substituting "appropriate" instances of a formula substructure.

[2]This proof was subsequently adapted to the case $F = \emptyset$, in the theorem `deletion_I`

We proved a result which is analogous to Lemma 3.7 (Substitution Lemma), using a statement involving `seqrep` rather than congruent occurrences. This was adequate for proving Lemma 3.8, that the single premise rules satisfy the local AD-interpolation property. The proof of Lemma 3.8 made use of a difficult lemma which had been proved previously as part of the work on cut-elimination in display logics.

Then we proved a version of Lemma 4.2 (Deletion Lemma), again avoiding a definition of congruence: instead of $C' \setminus F$ we proved the existence of a sequent which is obtained from C' by deleting some instances of F. This proof depended on some detailed ML programming so as to avoid having to prove a large number of cases individually.

4 Subsequent Work

4.1 Additive Logical Introduction Rules with more than one Premise

During 2011 I have worked on extending the results of §2.3 to additive logical introduction rules with more than one premise.

This involved, first, defining an analogue of `seqrep`, which we called `lseqrep`. Thus $(U, Us) \in$ `lseqrep` $b\ Y\ Ys$ means that occurrences of Y in U are replaced by the nth member of Ys in the nth member of Us. However we simplified it by requiring that this applies to exactly one occurrence of Y (and it is the *same* occurrence of Y for each member of Us).

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

I think that this simplification, that the replacement applies to exactly one occurrence of Y in U, simplified the proofs considerably compared with the previous proof of the theorem `extSubs`, however the complication, that we have lists Ys and Us, rather than single structures, complicated the proofs considerably. Thus we have a result textttmextSubs which is analogous to texttextSubs.

From there we proved `SF_some1sub`, analogous to `SF_some_sub`.

```
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))"
```

This says that

- assuming certain conditions on a set of rules (which are satisfied by the display postulates)
- if `concl` is derivable from `prems`
- if `concl` $^{\texttt{fml}}\!\leadsto^{Z}\ s$ `sconcls`

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

- for each `prem` in `prems`, let `sprems` be the corresponding member of `spremss`, then `prem` $^{\texttt{fml}}\!\leadsto^{Z}\ s$ `sprems`
- each member of `sconcls` is derivable from the corresponding member of each list in `spremss`

Then, corresponding to `sdA1` in §2.3, we have a lemma `msdA1` (and its three more counterparts). This is like `sdA1` except that, in its statement (see §2.3), Y' and W' can be lists. (It was proved from `strrep_mderA`, mentioned in §2.3).

Then, corresponding to `seqrep_interpA`, we have `lseqrep_interpA`: again, the difference is that in the statement of `seqrep_interpA` in §2.3, we replace Y' be a list of structures and $W' \vdash Z'$ by a list of sequents.

```
lseqrep_interpA : "[| rlscf <= ?rules;
    ALL U. (map (%Y'. $Y' |- $U) ?Ys, ?fml |- $U) : ?logI;
    seqIsLog ($?W |- $?Z); ALL Y:set ?Ys. strFVPPs Y <= fmlFVPPs ?fml;
    ($?W |- $?Z, ?Ss') : lseqrep False (Structform ?fml) ?Ys;
    ?logI <= PC ‘ rulefs ?rules; aidps <= ?rules;
    ALL S:set ?Ss'. Ex (interp ?rules S) |] ==>
  Ex (interp ?rules ($?W |- $?Z))"
```

There are two major cases in the proof: all the sequents W' are the same, or all the sequents Z' are the same. This is because the relation S $^{Z}\!\leadsto^{Z}\ s$ 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 Theorem 4.7 (Binary Rules) in the paper.

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`.

```
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)"
```

4.2 Local Display Interpolation for Structural Rules

We proved the local display interpolation property for the unit contraction rules and for the contraction rule.

The first lemma, `deletion_I_uc`, says that if sequent Cd is obtained from C by (possibly) deleting occurrences of $\#^n\emptyset$, and if $Cd' \rightarrow_{AD}^* Cd$, then there exists C', such that $C' \rightarrow_{AD}^* C$, and Cd' is obtained from C' by (possibly) deleting occurrences of $\#^n\emptyset$.

We accidentally made the error of first proving `deletion_I_uc'` (which swaps $C(Cd)$ with $C'(Cd')$) but subsequently discovered this was easier to prove anyway. But to fix the error we also needed `inv_der_aidps`.

```
inv_der_aidps :
   "?C : derivableR aidps {?C'} ==> ?C' : derivableR aidps {?C}"

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

The proof of this required a good deal of repetitive use of tactics and programming of complex tactics similar to those described in §2.4.

The two parts of the theorem `delI_der` shows that with Cd and C as above, Cd is derivable from C. Then `ldi_ila` and `ldi_ils` assert that the unit contraction rules satisfy the local display interpolation property. The rules for replacing $(\emptyset, X)$ by X on the left and the right are `ila` and `ils`.

```
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)"
```

Then we defined a relation `mseqctr`, where $(C, C') \in$ `mseqctr` means that C' is obtained from C, where they differ, by contraction of a subsructure (X, X) to X. The contraction may occur (of different substructures) and several places or none.

Then we obtained theorems `deletion_ctr`, `ctr_der` and `ldi_cA`, which correspond to the theorems mentioned above, but for contraction instead of unit contraction. (Note that the system does not give a rule for contraction on the right, it is derived from `cA`, the rule for contraction on the left).

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

At the time of writing we had looked at treating the weakening rule in a similar way, but it has extra difficulties.

4.3 Local Display Interpolation for Weakening

To handle weakening in a similar way, we considered two separate rules, one to weaken with instances of $\#^n\emptyset$ ($\#^n I$ in Isabelle) and one to change any instance of I to any formula.

We consider first replacing any instance of I with a formula. We got the theorem

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

Note that $(C, Cd) \in$ `seqrepI` Fs means that some occurrences in C of structures in Fs are replaced by I in Cd.

Since our formulation also contains structure variables (we don't distinguish the logical meta-language from the object language), we must get a similar theorem for them, which was no difficulty — like formulae, they are atomic so far as the structure langauge is concerned.

We proved that there are derived rules permitting replacing instances of I 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.

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

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

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

We use the variant of the theorem `deletion` (see §2.4) which applies to deletion of I rather than of a formula.

Then we have a theorem `mwk_der`, which says that the result of inserting occurrences of anything preserves derivability.

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

Then we have the result that such rules satisfy the local display interpolation property. In this case, though, where a sequent containing I is rearranged by the display postulates such that the I is alone on one side, (such as where $X \vdash Y, I$ is rearranged to $X, *Y \vdash I$, then the the local display interpolation requires using the derivability of $X \vdash Y$ rather than the fact that $X \vdash Y$ satisfies the local display interpolation property.

Thus we define a conditional local display interpolation property. Recall that, unlike Definition 3.3, our definition of `ldi` is not not conditional on the premises *ps* being derivable.

```
cldi_def : "cldi ?lrules ?drules (?ps, ?c) =
         (?c : derivableR ?drules {} --> ldi ?lrules ?drules (?ps, ?c))"

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

Thus we have proved that, provided the conclusion is derivable, a $\#^n I$-weakening rule satisfies the local display interpolation property.

4.4 The Interpolation Theorem

Now since we can obtain an arbitrary weakening by repeated $\#^n I$-weakenings and changing occurrences if I to formulae, we have a set of rules equivalent to the original set, of which all satisfy a sufficient local display interpolation property.

The details of this were rather tedious however.

First, that any weakening can be got by successive weakening of atoms (here, weakening means changing X to (X, Y) anywhere within the sequent, and atoms means not only formulae but also structure variables, since these are part of our language of structures).

```
rep_wk_atoms :
  "[| ?atoms = UNION (insert I (range SV) Un range Structform) stars;
      (?X, ?Y) : strdel UNIV; strIsLog ?X |] ==>
    (?X, ?Y) : (strdel ?atoms)^*"
```

We then defined a set of rules, called `ldi_rules`, to contain

- `aidps` (the display postulates and associativity)
- the structural rules which remove I, and contraction
- the identity rule
- substitutions of the above
- $\#^n I$-weakening
- replacing I by a structural atom (formula or structure variable)

We then proved that the set of rules `ldi_rules` is equivalent to the rules `rlscf` and all their substitutions.

This seems like enough to complete the proof of the interpolation theorem, however there were a number of complications which made it more difficult:

- the form of `ldi_wkI`: it requires that certain rules are in the set of rules in question: these rules are in `rlscf` but not in `ldi_rules` (they are derivable from `ldi_rules`) thus although we can get a counterpart of `ldi_ex_interp` for `cldi`, we couldn't use it directly

- the set of rules `ldi_rules` is not closed under substitution: some of our other development assumes that we are dealing with a set of rules which is closed under substitution

Finally, however, we got the following theorem:

```
ldi_rules_interp : "?x : derivableR rlscf {} ==> edi aidps rlscf ?x"
```

that is, Theorem 5.8 for the system containing all the rules (except the implication introduction rules) of Figures 1 and 2, and the display postulates and their inverses.

There was one fly in the ointment in all this: several of the lemmas were proved only for sequents satisfying the `seqIsLog` property of a sequent, which says that the only structural connectives are the boolean ones (ie the ones in the paper) — ie, the relational connectives (which were part of my earlier work on Display Logic, which I reused here) do not appear.

However this created difficulties and so I simply asserted an axiom saying that all structures have this property. The proofs depended on this axiom. Since, however this axiom is false (as the way I have defined structures means that non-logical structures exist), the proof depended on a false assumption.

Having looked at this situation, it seems that the easiest solution is to copy the work using a redefinition of structures which includes only the boolean structural connectives.

I have now done this, in new versions of `GDC.thy` and `GDC.ML` in the directory `interp`, distinct from the files of the same name used for the relation algebra work, which were formerly in directory `fdeep` and are now in `fdeep-only`. Similar new versions of files `GSub.thy` and `GRep.thy` were required.

5 Conclusion and Discussion

I have proved in Isabelle the interpolation theorem for the particular rule sets mentioned. It remains to examine how significant are the differences in the proofs from the paper, and whether I've achieved any significant simplification.

I have done additive binary logical rules in a way which is modelled on the treatment of unary rules, but using conjunction or disjunction of interpolants in a way motivated by reading the proof of Theorem 4.7. The proof was considerably more difficult than the proof for unary rules only because it happened that I already had some required lemmas for the unary case from my previous work.

Regarding Definition 4.4 ($\lhd$) and Lemmas 4.5 and 4.6, I have not formalised or used these, though it is likely that some similar ideas appear in my proofs.