

A Formalised Proof of Craig's Interpolation Theorem in Nominal Isabelle

Peter Chapman

Overview

We intend to:

- give a reminder of Craig's theorem, and the salient points of the proof
- introduce the proof assistant Isabelle, including the extension Nominal Isabelle
- show how this system can allow us to develop a formal proof which is similar to the informal approach

The Idea

Craig's Interpolation Theorem is about *implication*. Suppose we have a formula $A \supset B$ which is valid. Then, Craig's Theorem says that we can find a C satisfying

- both $A \supset C$ and $C \supset B$ are valid
- C is contained in the *common* language of A and B

The second condition deals with both polarities of formulae **and** their individual constants

Formally informal

Couched in the language of sequent calculi (specifically a first-order intuitionistic sequent calculus), we can state Craig's Theorem as follows:

Suppose that $\Gamma'' \Rightarrow D$. Then, for **any** splitting of the context $\Gamma'' \equiv \Gamma \cup \Gamma'$:

- $\exists C \text{ dl } dr. dl \vdash \Gamma \Rightarrow C \wedge dr \vdash \Gamma', C \Rightarrow D$
- **POL**: Any formula appearing positively (resp. negatively) in C occurs positively (resp. negatively) in Γ and D and negatively (resp. positively) in Γ'
- **CON**: The individual constants of C occurs in both Γ and $\Gamma' \cup D$

We get the theorem on the previous slide as a special case by setting $\Gamma' \equiv \emptyset$

An Induction

As is usual in proofs performed in sequent calculi, we proceed by induction on the height of the derivation, and case analysis on the last rule used. The cases split naturally into three:

- **Base case:** the no-premiss rules **Ax** and **L \perp**
- **Inductive Step - Propositional:** these cases, such as **R \vee** , are fairly straightforward
- **Inductive Step - First-Order:** these cases, such as **R \exists** , are where care must be taken to fully satisfy the theorem

We will give an example of valid derivations satisfying each of the above

Base Case - $L\perp$

Consider the case where the rule used was $L\perp$. We have two subcases: one where \perp is part of Γ , and one where it is part of Γ' . Note that we only have two cases where the rule is a *left* rule; when there is no principal formula on the right there is only one possibility for the splitting of $\Gamma'' \equiv \Gamma \cup \Gamma'$. Suppose $\perp \in \Gamma$. Then, we need two derivations, dl and dr , and a formula, C , such that

- $dl \vdash \Gamma \Rightarrow C$
- $dr \vdash \Gamma', C \Rightarrow D$
- The polarity and language invariants are satisfied

Base Case - $L\perp$

What to choose?

- How about \perp ?
- Clearly, $\Gamma \Rightarrow \perp$ is an instance of $L\perp$, since $\perp \in \Gamma$
- Furthermore, $\Gamma', \perp \Rightarrow D$ is also an instance of $L\perp$
- Even better, \perp has **no individual constants**

So, we have our two derivations and formula.

Propositional Fragment - R_{\wedge}

The problem can be stated as

$$\frac{\Gamma; \Gamma' \stackrel{C}{\Rightarrow} A \quad \Gamma; \Gamma' \stackrel{D}{\Rightarrow} B}{\Gamma; \Gamma' \stackrel{?}{\Rightarrow} A \wedge B}$$

where C and D are supplied by the induction hypothesis. Hence, we have 4 derivations with which to construct those needed by the theorem, with root sequents:

- 1 $\Gamma \Rightarrow C$
- 2 $\Gamma', C \Rightarrow A$
- 3 $\Gamma \Rightarrow D$
- 4 $\Gamma', D \Rightarrow B$

Propositional Fragment - R_{\wedge}

It makes sense to pair up the derivations according to their contexts. We see that the following derivations are both possible

$$\frac{\Gamma \Rightarrow C \quad \Gamma \Rightarrow D}{\Gamma \Rightarrow C \wedge D}$$

and

$$\frac{\frac{\Gamma', C \Rightarrow A}{\Gamma', C, D \Rightarrow A} \quad W \quad \frac{\Gamma', D \Rightarrow B}{\Gamma', C, D \Rightarrow B} \quad W}{\Gamma', C, D \Rightarrow A \wedge B} \\ \Gamma', C \wedge D \Rightarrow A \wedge B$$

But do they satisfy the polarity and language conditions?

Propositional Fragment - R_{\wedge}

The induction hypothesis also supplies

- C satisfies the polarity and language conditions for Γ, Γ' and A
- D satisfies the polarity and language conditions for Γ, Γ' and B

Therefore, $C \wedge D$ satisfies the polarity and language conditions for Γ, Γ' and $A \wedge B$

First-order - $R\exists$

The problem is as follows

$$\frac{\Gamma; \Gamma' \xRightarrow{C} [t/x]A}{\Gamma; \Gamma' \xRightarrow{?} \exists x.A}$$

The naïve approach would argue that C is a valid interpolant for the conclusion.

- C is certainly valid if the derivation is all that matters; nothing changes
- C is also valid with respect to the polarity condition, because the polarity of $[t/x]A$ and $\exists x.A$ are the same
- **However**, the language constraint **fails!**

First-order - $R\exists$

Suppose C contained some constants that were in t , but not in Γ , Γ' or A . Because every such constant will therefore **not** appear in the conclusion, we would have that C is **not contained** in the common language of Γ and Γ' , $\exists x.A$.

- How do we surmount this problem?
- **Answer:** We remove all such constants from C
- This is done using appropriate quantifications over the set of variables which have the above property

First-order - $R\exists$

We know that every variable in the set will be free, by definition, for Γ' and A . We can safely use the rule $L\exists$ on the premiss which uses Γ' :

$$\frac{\frac{\Gamma', [\vec{v}/\vec{u}]C \Rightarrow [t, x]A}{\Gamma', [\vec{v}/\vec{u}]C \Rightarrow \exists x.A}}{\Gamma', \exists \vec{u}.C \Rightarrow \exists x.A}$$

where \vec{v} is some set of fresh variables. The corresponding derivation for the premiss involving Γ is straightforward. Hence, we have that $\exists \vec{u}.C$, rather than just C , is the correct interpolant.

Embedding the sequent calculus

Isabelle, like any interactive environment, has some reserved words and symbols.

- Symbols like \wedge and \forall are **meta-level**
- So, we use $\wedge\wedge$ and \forall^* for their object level equivalents
- We also use \Rightarrow^* for the sequent arrow

Luckily, there is a package included in the Isabelle distribution to output these symbols!

Some details

We have that a formula is defined as a nominal datatype

```
nominal_datatype form = Atom "pred" "tm list"  
| Conj "form" "form" (infixr "^^" 110)  
| Disj "form" "form" (infixr "vv" 110)  
| Impl "form" "form" (infixr ">" 110)  
| Al "<var>form" ("∀* [] ._" (*<*) [100,100] 100 (*>*))  
| Exi "<var>form" ("∃* [] ._" (*<*) [100,100] 100 (*>*))  
| ff
```

and we also have a pair of functions that take a list of variables and quantify over each one. Here we see the \forall^L quantifier being defined

```
primrec  
  "∀L [] A = A"  
  "∀L (x # xs) A = ∀* [x] . (∀L xs A)"
```

Some details

A sequent is modelled as a set of formulae paired with a single formula, since we are using intuitionistic logic. We have a translation that makes this type more appealing to the eye

```
constdefs seqTrans :: "form set  $\Rightarrow$  form  $\Rightarrow$  seq" (infixr " $\Rightarrow^*$ " 50)  
seqT [simp] : " $(\Gamma \Rightarrow^* C) \equiv (\Gamma, C)$ "
```

Some details

We need to model substitution, since we have quantifiers.
Here, we use nominal Isabelle to substitute in a term list

```
consts substTerm :: "tm  $\Rightarrow$  var  $\Rightarrow$  tm list  $\Rightarrow$  tm list"  
primrec  
  "substTerm t x [] = []"  
  "substTerm t x (y # ys) = (if ( $\neg$  x # y) then (t # (substTerm t x ys)) else (y # (substTerm t x ys)))"
```

which we then extend to full first-order formulae as follows (only indicative or interesting cases are shown)

```
consts subst :: "tm  $\Rightarrow$  var  $\Rightarrow$  form  $\Rightarrow$  form" ("[( $\_$ ) , ( $\_$ ) ] ( $\_$ )" (*<*) [100,100] 100 (*>*))  
axioms  
subst_Atom(*<*) [simp] (*>*) : "[z,y] (Atom P ts) = Atom P (substTerm z y ts)"  
subst_Conj(*<*) [simp] (*>*) : "[z,y] (A  $\wedge$  B) = ([z,y]A)  $\wedge$  ([z,y]B)"  
subst_All(*<*) [simp] (*>*) : "[z,y] ( $\forall$ * [x].A) = (if (x # y) then ( $\forall$ * [x].([z,y]A)) else ( $\forall$ * [x].A))"  
subst_Ex(*<*) [simp] (*>*) : "[z,y] ( $\exists$ * [x].A) = (if (x # y) then ( $\exists$ * [x].([z,y]A)) else ( $\exists$ * [x].A))"
```

Some details

A derivation is a datatype, with constructors for all of the rules of the sequent calculus. For instance, the rule for conjunction on the right is

```
"is_wf (ConjR GammaC dl dr) = (let (Gamma, C) = GammaC in
  finite Gamma
  ^ (exists A B. C = A ^^ B
  ^ is_wf dl
  ^ root dl = (Gamma, A)
  ^ is_wf dr
  ^ root dr = (Gamma, B)))"
```

Some details

Here is a proof fragment in Isar, of the base case where the rule used is an Axiom

```
lemma craigIntAx1:
  fixes  $\Gamma_1 \Gamma_2$  :: "form set"
  and P :: "form"
  assumes As1: "P  $\in$   $\Gamma_1$ "
  and As2: "finite  $\Gamma_1 \wedge$  finite  $\Gamma_2$ "
  shows " $\exists A \text{ dl dr. (dl } \vdash (\Gamma_1 \Rightarrow^* A))$ 
         $\wedge$  (dr  $\vdash ((\Gamma_2 \cup \{A\}) \Rightarrow^* P))$ 
         $\wedge$  (A  $\subseteq$   $\Gamma_1 \Gamma_2$  P)"
proof-
  from As1 As2 have dl: "Init ( $\Gamma_1 \Rightarrow^* P$ )  $\vdash$  ( $\Gamma_1 \Rightarrow^* P$ )" by force
  from As2 have dr: "Init ( $\Gamma_2 \cup \{P\} \Rightarrow^* P$ )  $\vdash$  ( $\Gamma_2 \cup \{P\} \Rightarrow^* P$ )" by force
  from As1 have "con P  $\subseteq$  conIm  $\Gamma_1$ " by force
  moreover have "con P  $\subseteq$  conIm  $\Gamma_2 \cup$  con P" by force
  ultimately have cpn: "P  $\subseteq$   $\Gamma_1 \Gamma_2$  P" using As1 by force
  from dl dr cpn show ?thesis by blast
qed
```

Some details

Next we have a subcase of the inductive step. Here, the last rule used is conjunction on the left, and moreover the principal formula is in the left hand part of the context

```
lemma craigIntConjL1:
fixes  $\Gamma_1 \Gamma_2 :: \text{"form set"}$ 
    and A B C D :: "form"
    and dl dr :: "deriv"
assumes IHL : " $dl \vdash (\Gamma_1 \cup \{A,B\}, C)$ "
    and IHR : " $dr \vdash (\Gamma_2 \cup \{C\}, D)$ "
    and IH_cpn : " $C \in (\Gamma_1 \cup \{A,B\}) \Gamma_2 D$ "
    and as: " $(A \wedge B) \in \Gamma_1$ "
shows " $\exists E dl' dr'. (dl' \vdash (\Gamma_1, E)) \wedge (dr' \vdash (\Gamma_2 \cup \{E\}, D))$ "
     $\wedge (E \in \Gamma_1 \Gamma_2 D)$ "

proof-
from IHL as have dl: "ConjL ( $\Gamma_1, C$ )  $dl \vdash (\Gamma_1, C)$ " by force
have " $C \in \Gamma_1 \Gamma_2 D$ " using IH_cpn as by (simp only: containI2)
thus ?thesis using IHR dl
apply (rule_tac x=C in exI, rule_tac x="ConjL ( $\Gamma_1, C$ )  $dl$ " in exI, rule_tac x="dr" in exI)
by simp
qed
```

Some details

Finally, we move on to the interesting quantifier cases, where we must be careful about the constants. One of the important lemmata that we use is this one

```
lemma constant_Remove_All:
fixes A :: "form"
  and Y :: "var list"
shows "con ( $\forall^L$  Y A) = con A - (set Y)"
proof (induct Y)
case Nil
show ?case by force
next
case (Cons y ys)
have "con ( $\forall^L$  ys A) = con A - set ys" by fact
moreover have "con ( $\forall^L$  (y # ys) A) = con ( $\forall^L$  ys A) - {y}" by (simp add:con_All)
ultimately have "con ( $\forall^L$  (y # ys) A) = con A - set ys - {y}" by simp
thus ?case by force
qed
```

Some details

The assumptions for the case of existential quantification on the right are as follows

```
lemma craigIntExR:
fixes  $\Gamma_1 \Gamma_2$  :: "form set"
  and C A :: "form"
  and t :: "tm"
  and x :: "var"
  and X :: "var list"
  and dl dr :: "deriv"
assumes IHl: " $dl \vdash (\Gamma_1 \Rightarrow^* C)$ "
  and IHr: " $dr \vdash (\Gamma_2 \cup \{C\} \Rightarrow^* [t,x]A)$ "
  and var: " $set X = con C \cap con ([t,x]A)$ "
  and IH_cpn: " $C \sqsubseteq \Gamma_1 \Gamma_2 ([t,x]A)$ "
shows " $\exists dl' dr' E. (dl' \vdash (\Gamma_1 \Rightarrow^* E)) \wedge (dr' \vdash (\Gamma_2 \cup \{E\} \Rightarrow^* \exists^* [x].A))$ "
   $\wedge (E \sqsubseteq \Gamma_1 \Gamma_2 (\exists^* [x].A))$ "
```

Some details

Which we then prove using the following. We pay particular attention to the condition `con2`

```
have conIH:"con (∃L X C) ⊆ con C" by (simp add:constant_Remove_Ex,force)
then have con1:"con (∃L X C) ⊆ conIm Γ1" using IH_cpn by force
have "con (∃L X C) = con C - (con C ∩ con ([t,x]A))" using var by (simp add:constant_Remove_Ex)
then have "con (∃L X C) = con C - con ([t,x]A)" by force
then have "con (∃L X C) ⊆ (conIm Γ2 ∪ con ([t,x]A)) - con ([t,x]A)" using IH_cpn by force
then have "con (∃L X C) ⊆ conIm Γ2" by force
then have con2: "con (∃L X C) ⊆ (conIm Γ2 ∪ con (∃* [x].A))" by force
```

Some details

We put all of this together in a large induction, of which we show the base cases here

```
lemma craigInt: "  
  ∀ d Γ1 Γ2 c.  
    d ⊢ (Γ1 ∪ Γ2 ⇒* c)  
    →  
    (∃ E d1 dr. d1 ⊢ (Γ1 ⇒* E) ∧ dr ⊢ ({E} ∪ Γ2 ⇒* c)  
      ∧ (E ⊆ Γ1 Γ2 c))"
```

```
apply (rule) apply (induct_tac d) apply simp  
apply (intro allI impI, elim conjE)  
apply (simp_all, elim conjE) apply (rename_tac E)  
apply (simp add:craigIntAx[simplified])  
apply (intro allI impI, elim conjE) apply (simp add:Let_def, elim conjE) apply (rename_tac E)  
apply (simp add:craigIntFalse[simplified])
```

Unwanted Axioms

The development contains three axioms, which should be provable, but I cannot do it at the moment

```
axioms fresh_insert: "(y :: var) # (insert x (X)) ≡ ((y # x) ∧ (y # X))"  
fresh_subset: "(y :: var) # ((X) ∪ (Y)) ≡ ((y # X) ∧ (y # Y))"  
fresh_AllEx: "y # ([x].A) ≡ ((y # x) ∧ (y # A))"
```

Lemmata not being unified

At a certain point in the proof, we get the following premisses

```
is_wf d1l; is_wf d1r; root d1l = ( $\Gamma$ 1, E'); root d1r = ( $\Gamma$ 1, E''); is_wf d1x; root d1x = (insert E'  $\Gamma$ 2, B);  
pos E'  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 1. pos x); pos E''  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 2. NominalCraig.neg x)  $\cup$  pos B; NominalCraig.neg E'  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 1. NominalCraig.neg x);  
NominalCraig.neg E''  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 2. pos x)  $\cup$  NominalCraig.neg B; con E'  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 1. con x); con E''  $\subseteq$  ( $\bigcup_{x \in \Gamma$ 2. con x)  $\cup$  con B;  
is_wf d2r; root d2r = (insert E''  $\Gamma$ 2, A); pos E'''  $\subseteq$  UNION  $\Gamma$ 1 pos; pos E''''  $\subseteq$  UNION  $\Gamma$ 2 NominalCraig.neg  $\cup$  pos A;  
NominalCraig.neg E'''  $\subseteq$  UNION  $\Gamma$ 1 NominalCraig.neg; NominalCraig.neg E''''  $\subseteq$  UNION  $\Gamma$ 2 pos  $\cup$  NominalCraig.neg A;  
con E'''  $\subseteq$  UNION  $\Gamma$ 1 con; con E''''  $\subseteq$  UNION  $\Gamma$ 2 con  $\cup$  con A]
```

and the theorem I wish to apply is

```
[is_wf ?d1l  $\wedge$  root ?d1l = (? $\Gamma$ 1.0, ?E); is_wf ?d1r  $\wedge$  root ?d1r = (insert ?E ? $\Gamma$ 2.0, ?C); is_wf ?d1l  $\wedge$  root ?d1l = (? $\Gamma$ 1.0, ?F);  
is_wf ?d2r  $\wedge$  root ?d2r = (insert ?F ? $\Gamma$ 2.0, ?D);  
pos ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. pos x)  $\wedge$   
pos ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. NominalCraig.neg x)  $\cup$  pos ?C  $\wedge$   
NominalCraig.neg ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. NominalCraig.neg x)  $\wedge$   
NominalCraig.neg ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. pos x)  $\cup$  NominalCraig.neg ?C  $\wedge$  con ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. con x)  $\wedge$  con ?E  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. con x)  $\cup$  con ?C;  
pos ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. pos x)  $\wedge$   
pos ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. NominalCraig.neg x)  $\cup$  pos ?D  $\wedge$   
NominalCraig.neg ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. NominalCraig.neg x)  $\wedge$   
NominalCraig.neg ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. pos x)  $\cup$  NominalCraig.neg ?D  $\wedge$  con ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 1.0. con x)  $\wedge$  con ?F  $\subseteq$  ( $\bigcup_{x \in ?\Gamma$ 2.0. con x)  $\cup$  con ?D]
```

so the only difference is that the first context has, for instance
“*is-wf d ; root d = ...*”, whereas the second has “*is-wf d \wedge root
d = ...*”

Other Methatheoretical Results

- Craig's Interpolation Theorem is a simple result, and is a simple induction, but the conditions are tiresome to prove
- Cut Admissibility is also a straightforward result, and the derivations are clear with no extra checks being needed, but the induction is on a more complicated measure
- I am currently working on a proof of Cut Admissibility for the system **G3ip**, although at the minute it is only the conjunctive fragment
- In the long run, we hope to be able to automatically provide a framework for other such proofs (of Cut Admissibility) so that all we need do is enter appropriate derivations, and the system checks all of the details

Why do we do this?

Formalising mathematics has a number of benefits

- Can find flaws in proofs
- Forces us to fill in all blanks in a proof
- A proof script is a useful teaching tool

Any questions?