Abstract

Craig Interpolants: definitions, intuitions, and applications.

1 Definitions

The following sentences A, B, C are special. For each, A implies B. Furthermore, A implies C and C implies B.

$$\begin{array}{lcl} A & = & \neg(P \wedge Q) \Rightarrow (\neg R \wedge Q) \\ B & = & (T \Rightarrow P) \wedge (T \Rightarrow \neg R) \\ C & = & (P \vee \neg R) \end{array}$$

$$\begin{array}{lcl} A & = & P \vee (Q \wedge R) \\ B & = & P \vee \neg\neg Q \\ C & = & P \vee Q \end{array}$$

$$\begin{array}{lcl} A & = & \neg(P \wedge Q) \implies (\neg R \wedge Q) \\ B & = & (S \Rightarrow P) \vee (S \Rightarrow \neg R) \\ C & = & P \vee \neg R \end{array}$$

Definition. *Craig Interpolant (1957)*
Suppose A and B are logical formulas. An interpolant C for the pair (A, B) is:

- *Implied by A:* $\vdash A \Rightarrow C$
- *Sufficient to prove B:* $\vdash C \Rightarrow B$
- *Expressed over the common variables of A and B:*
  atoms$(C) \subseteq$ atoms$(A) \cup$ atoms(B)

Theorem. *If $\vdash A \Rightarrow B$ then an interpolant for (A, B) exists [2].*

Proof. By induction on the size of $V = \mathit{atoms}(A) \setminus \mathit{atoms}(B)$. If V is empty, A is an interpolant. Else choose any variable $v \in V$ and define $A' = A[\top/v] \vee A[\bot/v]$. By the induction hypothesis, an interpolant for (A', B) is an interpolant for (A, B).

If $\mathit{atoms}(A) \cap \mathit{atoms}(B) = \emptyset$ then either $\vdash \neg A$ or $\vdash B$. $\square$

Challenge. *Find an optimal interpolant for (A, B) i.e. smallest, least variables, quantifier-free.*

The proof above can make an exponentially large term. Craig's proof introduces quantifiers.

2 Craig Interpolants in Model Checking

Very simple program:

```
1 void f(int n) {
2     int x = n;
3     int y = n + 1;
4     assert(y == x + 1);
5 }
```

Goal: prove that the assertion on line 4 is never violated.
At line 4, we have the following premise (A) and goal (B):

$$\begin{aligned} A &= \{n \in \mathtt{short} \land x = n \land y = n + 1\} \\ B &= y = x + 1 \end{aligned}$$

A suitable interpolant for (A, B) is B.

2.1 Basic Strategy

Start by finding a path in the program to as `assert` statement. The path will be represented by primarily by transitions $T(s_i, s_j)$ from one state s_i to a successor state s_j. The final state in the path is the assertion C we wish to prove correct; in total we can represent the path as a formula:

$$p = T(s_0, s_1) \land T(s_1, s_2) \land \ldots \land T(s_{n-1}, s_n) \land C(s_n)$$

The formula should be true of a specific path, but we want to know whether it holds for all paths. The key idea of interpolation-based model checking is to use our proof that p is correct to find counterexamples to C.

We take the (false) formula p':

$$p' = T(s_0, s_1) \land T(s_1, s_2) \land \ldots \land T(s_{n-1}, s_n) \land \neg C(s_n)$$

and consider of each $\land$ from left to right in turn as a formula $A \land B$ where A and B are mutually inconsistent. Then we apply interpolation to get a formula A' that is:

- Implied by A
- Inconsistent with B
- Expressed over the common atoms: *atoms*$(A) \setminus$ *atoms*(B)

A good interpolant A' will contain no irrelevant information about B. In other words, A' contains only the facts about A necessary to prove the assertion C holds later in the path.

2.1.1 Equivalent Definitions

Note: this definition is classically equal to Craig's original, just switch $\neg B$ for B. It is also more common in the model-checking community [8]. Here are a few new-style interpolants.

$$
\begin{array}{rcl}
A & = & u = x \wedge f(u,y) = z \\
B & = & v = y \wedge f(x,v) \neq z \\
C & = & f(x,y) = z
\end{array}
$$

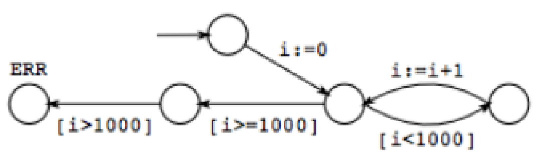
$$
\begin{array}{rcl}
A & = & x \leq y \wedge y \leq z \\
B & = & x - z - 1 \geq 0 \\
C & = & x \leq z
\end{array}
$$

2.2 While Programs

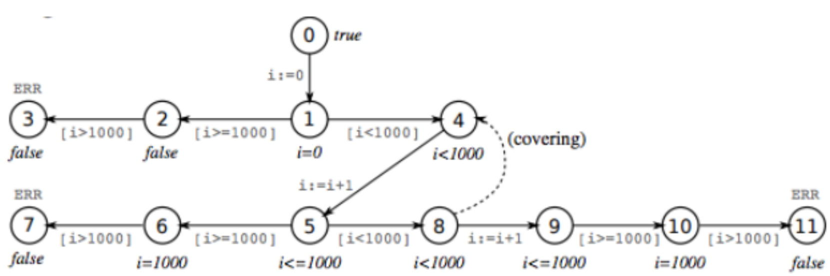
Take a small while program [1]:

```
1   int i = 0;
2   while (i < 1000)
3     i += 1;
4   assert(i <= 1000);
```

Using interpolation,[1] we can prove that the assertion on line 4 never fails. First, the control-flow-graph of our program is:



By unrolling the program, exploring paths, and computing interpolants for each state (by splitting the path formula into two inconsistent conjunctions) we get the tree:



[1] Also: bounded model checking, predicate abstraction, and lazy abstraction.

Correctness follows because the 3 error states are unreachable and state 4 *covers* state 8. The covering condition follows because states 4 and 8 refer to the same control-flow condition and the proposition $i < 1000$ at state 8 implies the proposition at state 4.

2.3 More Examples

From D'Silva [3]:

```
void g(int i, int j) {
  int x = i;
  int y = j;
  int tmp;
  while (*) {
    tmp = x;
    x = y + 1;
    y = tmp + 1;
  }
  if (i == j && x <= 10) {
    assert(y <= 10);
  }
}
```

A few possible interpolants:

1. $i = j \implies x \leq y$
2. $i = j \implies y \leq x$
3. $(1) \wedge (2) \implies x = y$

3 Alternative: Image Computation

Before interpolation, "the way" to annotate states was *image computation* i.e. computing all successors of each state [8]. One would compute the strongest invariant of a program with initial state I and transition relation T by taking the fixed point of all strongest postconditions at each reachable state.

$$R(I,T) = \mu U.\, I \vee \mathsf{post}_T(Q)$$

Each $\mathsf{post}_T(Q)$ for a state formula Q is easy to express in propositional logic, but difficult to compute:

$$\mathsf{post}_T(Q) = \exists S.\, Q \wedge T$$

where S is a signature representing the entire state space. At least we know how to compute it, but the process is very slow for all but the smallest programs. If post_T is monotonic, the least fixed point of Q exists and is the strongest invariant of the program.[2]

4 How to Derive Interpolants

Core idea for how to derive an interpolant from a refutation of $A \wedge B$.

4.1 Basic Logic

Start with a quantifier-free propositional logic and these rules for proving $\nvdash A \wedge B$ [3]. Very important to divide rules based on where the variables occur.

$$\textsc{A-Hyp}\ \frac{C \in A}{\vdash C} \qquad \textsc{A-Res}\ \frac{x \in \mathit{atoms}(A) \setminus \mathit{atoms}(B) \qquad \vdash C \vee x \qquad \vdash \neg x \vee D}{\vdash C \vee D}$$

$$\textsc{B-Hyp}\ \frac{C \in B}{\vdash C} \qquad \textsc{B-Res}\ \frac{x \in \mathit{atoms}(B) \qquad \vdash C \vee x \qquad \vdash \neg x \vee D}{\vdash C \vee D}$$

Then annotate rules with *partial interpolants*.

$$\textsc{A-Hyp}\ \frac{C \in A}{\vdash C \quad [\{\, C' \in C \mid \mathit{atoms}(C') \subseteq \mathit{atoms}(B)\,\}]}$$

$$\textsc{A-Res}\ \frac{x \in \mathit{atoms}(A) \setminus \mathit{atoms}(B) \qquad \vdash C \vee x \quad [I_1] \qquad \vdash \neg x \vee D \quad [I_2]}{\vdash C \vee D \quad [I_1 \vee I_2]}$$

$$\textsc{B-Hyp}\ \frac{C \in B}{\vdash C \quad [\top]} \qquad \textsc{B-Res}\ \frac{x \in \mathit{atoms}(B) \qquad \vdash C \vee x \quad [I_1] \qquad \vdash \neg x \vee D \quad [I_2]}{\vdash C \vee D \quad [I_1 \wedge I_2]}$$

4.1.1 Sample Proof

$$\begin{aligned} A &= (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2 \\ B &= (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4 \end{aligned}$$

An interpolant is $C = \neg a_3 \wedge a_2$, derived below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a_1 \vee \neg a_2 [\neg a_2]} \qquad \dfrac{}{\neg a_1 \vee \neg a_3 \quad [\neg a_3]}}{\neg a_2 \vee \neg a_3 \quad [\neg a_2 \vee \neg a_3]} \qquad \dfrac{}{a_2 \quad [a_2]}}{\neg a_3 \quad [\neg a_3 \wedge a_2]} \qquad \dfrac{\dfrac{}{\neg a_2 \vee a_3 \quad [\top]} \qquad \dfrac{\dfrac{}{a_2 \vee a_4 \quad [\top]} \qquad \dfrac{}{\neg a_4 \quad [\top]}}{a_2 \quad [\top]}}{a_3 \quad [\top]}}{\vdash \bot \quad [\neg a_3 \wedge a_2]}$$

[2]Other fixed points are inductive invariants of the program.

4.2 Basic Arithmetic

McMillan's simple rules for linear inequalities [9]:

$$\text{H<A}\ \frac{0 \le x \in A}{\vdash 0 \le x \quad [x]} \qquad\qquad \text{H<B}\ \frac{0 \le x \in B}{\vdash 0 \le x \quad [\top]}$$

$$\textsc{Comb}\ \frac{\vdash 0 \le y - x \quad [y - x] \qquad\qquad \vdash 0 \le z - y \quad [z - y]}{\vdash 0 \le c_1 x + c_2 y \quad [c_1 x' + c_2 y']}$$

4.2.1 Example

$$\begin{aligned} A &= (0 \le y - x) \wedge (0 \le z - y) \\ B &= 0 \le x - z - 1 \end{aligned}$$

Now we show that A and B are inconsistent and derive an interpolant.

$$\dfrac{\dfrac{\dfrac{}{\vdash 0 \le y - x \quad [y - x]} \qquad \dfrac{}{\vdash 0 \le z - x \quad [z - y]}}{\vdash 0 \le z - x \quad [z - x]} \qquad \dfrac{}{\vdash 0 \le x - z - 1 \quad [\top]}}{\vdash 0 \le -1 \quad [z - x]}$$

4.3 Complexity Results

At least one of the following is true [10]:

- $\mathsf{P} = \mathsf{NP}$
- $\mathsf{NP} \neq \mathsf{coNP}$
- Then interpolants in propositional logic are not in general computable in time polynomial in the size of (A, B).

If the propositional formula $A \wedge B$ has a refutation of size n there is an interpolant of circuit size $3n$ [7].

5 Applications of Craig Interpolation in Model Checking

McMillan [8] gives three examples of using interpolants to do model checking faster / more efficiently.

1. Find invariants of program paths
2. Choose predicates to approximate a program state. Relies on interpolants not introducting new quantifiers.
3. Filter irrelevant details from a transition relation

6 Theories with Efficient Interpolants

- Resolution, bounded arithmetic theory, linear equational calculus, cutting planes [7].
- Linear Arithmetic with quantifiers ($\mathsf{LA}(\mathsf{Q})$) [9].
- Datatype theories [6].
- Quatifier-free, linear inequalities, equality, uninterpreted functions [9].
- Quantifier-free Presburger Arithmetic with arrays [1].
- $\mathsf{DL}(\mathsf{Q})$, UTVPI
- Linear Diophantine & Linear Modular equations [5]
- Bit vectors [4].

7 Reflecting

> ”Craig’s Theorem is about the last significant property of first-order logic that has come to light. Is there something deeper going on here, and if so, can we prove it?” - Van Bentham, 2008

References

[1] Angelo Brillout, Daniel Kroening, Phillip Rümmer, and Thomas Wahl. Program verification via craig interpolation for presburger arithmetic with arrays. 2010. `http://www.ccs.neu.edu/home/wahl/Publications/bkrw10a.pdf`.

[2] William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. 1957.

[3] Vijay D’Silva. Interpolation: Theory and applications. 2015. `http://www.cs.nyu.edu/~barrett/summerschool/dsilva.pdf`.

[4] Alberto Griggio. Effective word-level interpolation for software verification. 2011.

[5] Himanshu Jain, Edmund Clarke, and Orna Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. 2008.

[6] Deepak Kapur, Rupak Majumdar, and Calogero G. Zarba. Interpolation for data structures. `https://www.cs.unm.edu/~kapur/mypapers/Interpolation_for_data_structures.pdf`.

[7] Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. 1997.

[8] K.L. McMillan. Applications of craig interpolants in model checking. 2005.

[9] K.L. McMillan. An interpolating theorem prover. 2005. `http://www.kenmcmil.com/pubs/TCS05.pdf`.

[10] Daniele Mundici. Tautologies with a unique craig interpolant vs. nonuniform complexity. 1984.

Appendix: Craig's Statement & Proof

THE JOURNAL OF SYMBOLIC LOGIC
Volume 22, Number 3, Sept. 1957

THREE USES OF THE HERBRAND-GENTZEN THEOREM IN RELATING MODEL THEORY AND PROOF THEORY

WILLIAM CRAIG

2. Lemma and extensions. We shall consider a system *PCI* of first-order predicate calculus without identity and a system *PCI=* of first-order predicate calculus with identity. *PCI* shall contain individual variables and constants, and predicate variables and constants of $n \geq 0$ arguments. *PCI* shall *not* contain a symbol for identity and shall *not* contain symbols for functions of $n \geq 1$ individual arguments. (Most results of this paper do not hold for first-order predicate calculus with function symbols but without axioms for identity.) The result of adding to *PCI* these further symbols, changing the formation rules accordingly, and adding also axioms for identity shall be *PCI=*. $\vdash$ and $\vdash_=$ shall stand for derivability in *PCI* or *PCI=* respectively. The letters A, B, C, etc. shall refer to the formulas of the system concerned. Those formulas in which no individual variable occurs free shall be *sentences*. (Hence any 0-place predicate symbol is a sentence.) The individual constants and the individual variables occurring free in a formula shall be the *individual parameters* of that formula. Likewise, the function symbols in a formula and the predicate symbols other than the identity sign shall be its *function* or *predicate parameters* [3] respectively. The identity sign shall not be a parameter but a logical constant, since its interpretation cannot vary (except for the range of definition). Thus among the parameters of a formula of *PCI* are all its predicate symbols, no function symbols, and none or more individual symbols. Among the parameters of a formula of *PCI=* are all its predicate symbols except the identity sign, all its function symbols, and none or more individual symbols.

LEMMA 1. *If* $\vdash A \supset A'$ *and if* A *and* A′ *have a predicate parameter in common, then there is an "intermediate" formula* B *such that* $\vdash A \supset B$, $\vdash B \supset A'$, *and all parameters of* B *are parameters of both* A *and* A′. *Also, if* $\vdash A \supset A'$ *and if* A *and* A′ *have no predicate parameter in common,* [4] *then either* $\vdash \neg A$ *or* $\vdash A'$.

---

[3] This usage is taken from [1].

[4] This case was first called to my attention by P. C. Gilmore. For the special case where in addition A and A′ are sentences, he has found a much simpler argument in terms of satisfiability.

PROOF. The results of [4] hold a fortiori for PC_I in place of first-order predicate calculus with function symbols. Hence by Theorem 5 of [4], which is derived from the Herbrand-Gentzen Theorem, there is a B* which satisfies all the requirements of the lemma except perhaps that B* may contain individual parameters which are not parameters of both A and A′. Now take each individual parameter of B* which is not a parameter of A, replace all its free occurrences in B* by a new individual variable, and then universally quantify this variable over the entire formula. In the resulting formula similarly replace by an existentially quantified variable any individual parameter which is not a parameter of A′. The formula B which thus finally results satisfies the lemma.

The methods of [4] allow a more detailed study than is needed here of how the structure of A and B are related. For example, if A is a formula in prenex normal form containing only universal quantifiers and containing all the individual parameters of A′, then B can easily be shown to be a formula of the same kind.