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 \( B \).

\[
egin{align*}
A &= \neg(P \land Q) \Rightarrow (\neg R \land Q) \\
B &= (T \Rightarrow P) \land (T \Rightarrow \neg R) \\
C &= (P \lor \neg R) \\
A &= P \lor (Q \land R) \\
B &= P \lor \neg Q \\
C &= P \lor Q \\
A &= \neg(P \land Q) \implies (\neg R \land Q) \\
B &= (S \Rightarrow P) \lor (S \Rightarrow \neg R) \\
C &= P \lor \neg R
\end{align*}
\]

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 \( B \):
  \[ \text{atoms}(C) \subseteq \text{atoms}(A) \cup \text{atoms}(B) \]

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

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

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

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:

```c
void f(int n) {
  int x = n;
  int y = n + 1;
  assert(y == x + 1);
}
```

Goal: prove that the assertion on line 4 is never violated.

At line 4, we have the following premise \((A)\) and goal \((B)\):

\[ A = \{n \in \text{short} \land x = n \land y = n + 1\} \]
\[ B = y = x + 1 \]

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

2.1 Basic Strategy

Start by finding a path in the program to an \texttt{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: \(\text{atoms}(A) \setminus \text{atoms}(B)\)

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.

\[
\begin{align*}
A &= u = x \land f(u, y) = z \\
B &= v = y \land f(x, v) \neq z \\
C &= f(x, y) = z \\
A &= x \leq y \land y \leq z \\
B &= x - z - 1 \geq 0 \\
C &= x \leq z
\end{align*}
\]

2.2 While Programs

Take a small while program [1]:

```plaintext
int i = 0;
while (i < 1000)
  i += 1;
assert (i <= 1000);
```

Using interpolation,\textsuperscript{1} we can prove that the assertion on line 4 never fails.

First, the control-flow-graph of our program is:

![Control-flow-graph](image)

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:

![Diagram with states and edges](image)

\textsuperscript{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]:

```c
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. $(1) \land (2) \implies x = y$

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.

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

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

$$\text{post}_T(Q) = \exists S. Q \land T$$

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 pro-
grams. If $\text{post}_T$ is monotonic, the least fixed point of $Q$ exists and is the strongest invariant of the program.\footnote{Other fixed points are inductive invariants of the program.}

4 How to Derive Interpolants

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

4.1 Basic Logic

Start with a quantifier-free propositional logic and these rules for proving $\not\vdash A \land B$ \footnote{Other fixed points are inductive invariants of the program.}. Very important to divide rules based on where the variables occur.

\begin{align*}
\text{A-HYP} & \quad C \in A \quad \vdash C \\
\text{A-Res} & \quad x \in \text{atoms}(A) \setminus \text{atoms}(B) \quad \vdash C \lor x \quad \vdash \neg x \lor D \\
\text{B-Hyp} & \quad C \in B \quad \vdash C \\
\text{B-Res} & \quad x \in \text{atoms}(B) \quad \vdash C \lor x \quad \vdash \neg x \lor D
\end{align*}

Then annotate rules with partial interpolants.

\begin{align*}
\text{A-Hyp} & \quad C \in A \\
\text{A-Res} & \quad x \in \text{atoms}(A) \setminus \text{atoms}(B) \quad \vdash C \lor x \quad \vdash \neg x \lor D \quad [I_1] \quad [I_2] \\
\text{B-Hyp} & \quad C \in B \quad \vdash C \quad [\top] \\
\text{B-Res} & \quad x \in \text{atoms}(B) \quad \vdash C \lor x \quad \vdash \neg x \lor D \quad [I_1 \land I_2]
\end{align*}

4.1.1 Sample Proof

\begin{align*}
A & = (a_1 \lor \neg a_2) \land (\neg a_1 \lor \neg a_3) \land a_2 \\
B & = (\neg a_2 \lor a_3) \land (a_2 \lor a_4) \land \neg a_4
\end{align*}

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

\begin{align*}
\frac{a_1 \lor \neg a_2 \mid \neg a_2}{\neg a_2 \lor \neg a_3 \mid \neg a_2 \lor \neg a_3} \quad \frac{\neg a_1 \lor \neg a_3 \mid \neg a_3}{a_2 \mid \neg a_2 \lor a_3 \mid \neg a_3 \land a_2} \quad \frac{a_2 \lor a_4 \mid \top}{a_2 \mid a_2 \mid \top} \\
\frac{a_2 \mid \neg a_2}{\neg a_3 \mid \neg a_3 \land a_2} \quad \frac{a_2 \mid \neg a_2 \lor a_3 \mid \top}{a_2 \mid a_2 \mid \top} \quad \frac{a_2 \mid \top}{a_3 \mid \top} \quad \frac{\top \mid \top}{\vdash \bot \mid \neg a_3 \land a_2}
\end{align*}
4.2 Basic Arithmetic

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

\[
\begin{align*}
H &< A & 0 \leq x \in A & \vdash 0 \leq x [x] \\
H &< B & 0 \leq x \in B & \vdash 0 \leq x [\top] \\
\text{COMB} & & 0 \leq y - x [y - x] & \vdash 0 \leq z - y [z - y] \\
& & \vdash 0 \leq c_1 x + c_2 y [c_1 x' + c_2 y'] & \vdash 0 \leq z - x [z - x] \\
& & & \vdash 0 \leq -1 [z - x]
\end{align*}
\]

4.2.1 Example

\[
A = (0 \leq y - x) \land (0 \leq z - y) \\
B = 0 \leq x - z - 1
\]

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

\[
\begin{align*}
& \vdash 0 \leq y - x [y - x] \quad \vdash 0 \leq z - x [z - x] \\
& \vdash 0 \leq z - y [z - y] \quad \vdash 0 \leq x - z - 1 [\top] \\
\end{align*}
\]

4.3 Complexity Results

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

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

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

5 Applications of Craig Interpolation in Model Checking


1. Find invariants of program paths
2. Choose predicates to approximate a program state. Relies on interpolants not introducing new quantifiers.
3. Filter irrelevant details from a transition relation
6 Theories with Efficient Interpolants

- Resolution, bounded arithmetic theory, linear equational calculus, cutting planes [2].
- Linear Arithmetic with quantifiers (LA(Q)) [9].
- Datatype theories [2].
- Quatifier-free, linear inequalities, equality, uninterpreted functions [2].
- DL(Q), UTVPI
- Linear Diophantine & Linear Modular equations [5]
- Bit vectors [3].

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 Benthem, 2008

References

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 \( \vdash A \supset B \) 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 PCF 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.