OpenSMT and Applications

Roberto Bruttomesso

Università della Svizzera Italiana (USI), Lugano, Switzerland

Abstract. These notes are accompanying material for the lecture entitled "OpenSMT and Applications" held at MIT during the First Summer School on SAT and SMT. The purpose of this document is mainly to provide exhaustive bibliographic references for the topics discussed during the presentation.

1 OpenSMT

The OpenSMT project started in 2008 in the Formal Verification and Security Group at USI (Università della Svizzera Italiana, Lugano, Switzerland) as an attempt of building an open-source, easy to extend, but at the same time efficient, SMT-Solver. As of 2010 (see SMT-COMP'10) OpenSMT is the fastest open-source SMT-Solver in several logics (QF_UF, QF_IDL, QF_RDL, QF_UFIDL). Details about OpenSMT architecture and applications can be found in [3, 5–8, 18]

2 Interpolants

The notion of "interpolant" is connected with Craig's Interpolation Theorem [10], that shows that in first order logic, for every pair of unsatisfiable formulæ A and B, there exists an interpolant, i.e., a formula I[1] such that

(i) $T \vdash A \rightarrow I$;
(ii) $B \wedge I$ is T-unsatisfiable;
(iii) I is defined over the common non-logical symbols.[2]

Interpolants find application in model-checking as an alternative/improvement over the use of quantifier-elimination methods for computing the "post" states of a transition system [14], or to improve abstraction-refinement paradigm in the context of software verification by means of lazy abstraction [12, 16]. In both cases the verifcation effort is carried out by means of SAT/SMT-Solvers.

In order to keep everything decidable, it is important to work at the quantifier-free level. Notice that Craig's theorem does not guarantee that if A and B are

[1] Throughout these notes we will use red for formula A and symbols local to it, blue for formula B and symbols local to it, and green for interpolants.

[2] This formulation of the theorem is actually slightly different from (but equivalent to) the original one, in which an interpolant is a formula I for a pair of formulæ $A \rightarrow B$, such that $A \rightarrow I \rightarrow B$.

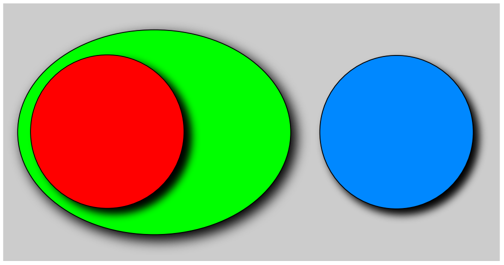


Fig. 1. A graphical representation of the notion of interpolant.

quantifier-free then I is so. It is the case for some theories, such as $\mathcal{EUF}$ (Equality with Uninterpreted Functions) [11,15], $\mathcal{LRA}$ (Linear Rational Arithmetic) [15], $\mathcal{LIA}$ (Linear Integer Arithmetic) if we allow the use of $\equiv_n$ predicates [1], $\mathcal{AX}$ (Arrays with extensionality) if we allow the use of a `diff` function symbol [4]. It is also interesting to notice that theory combination does not "transfer" to computing quantifier-free interpolants, because even though $\mathcal{EUF} \cup \mathcal{LRA}$ [15] admits quantifier-free interpolants, $\mathcal{EUF} \cup \mathcal{LIA}$ does not [2].

2.1 Examples and Exercizes

Example 1. If $A \equiv \bot$ an interpolant is $\bot$. If $B \equiv \bot$ and interpolant is $\top$.

Example 2. An interpolant for

$$A \equiv \{\neg a \wedge (a \vee c_1) \wedge (a \vee c_2)\},\;\; B \equiv \{\neg b \wedge (b \vee \neg c_1) \wedge (b \vee \neg c_2)\}$$

is $I \equiv c_1$.

Example 3. An interpolant for

$$A \equiv \{(x - y \leq 2) \wedge (y - z \leq 1)\},\;\; B \equiv \{(z - w \leq 0) \wedge (w - x \leq -10)\}$$

is $I \equiv x - z \leq 8$.

Exercise 1. Verify that the interpolants in the above examples verify properties $(i) - (iii)$.

Exercise 2. Find alternative interpolants for Examples 2 and 3.

2.2 Interpolants in OpenSMT

Interpolants can be computed in OpenSMT by means of some "SMT-LIB2-like" commands [3]:

[3]It is important to precise that no standard command for interpolation exists in the current SMT-LIB2 standard [17].

– `(set-option :produce-interpolants <bool>)` tells OpenSMT to compute interpolants;
– `(assert-partition <formula>)` tells OpenSMT about a partition;
– `(get-interpolant <n>)` commands to retrieve interpolant.

Example 4. The following script computes an interpolant of example 2.

```
(set-option :print-success false)
(set-logic QF_IDL)
(set-option :produce-interpolants true)
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c_1 () Bool)
(declare-fun c_2 () Bool)
; Partition A
(assert-partition (and
   (not a)
   (or a c_1)
   (or a c_2)
  ))
; Partition B
(assert-partition (and
   (not b)
   (or b (not c_1))
   (or b (not c_2))
  ))
(check-sat)
(get-interpolant 1)
(exit)
```

Example 5. The following script computes an interpolant of example 3.

```
(set-option :print-success false)
(set-logic QF_IDL)
(set-option :produce-interpolants true)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(declare-fun w () Int)
; Partition A
(assert-partition (and
   (<= (- x y) 2)
   (<= (- y z) 1)
  ))
; Partition B
(assert-partition (and
   (<= (- z w) 0)
```

```
    (<= (- w x) (- 10))
  ))
(check-sat)
(get-interpolant 1)
(exit)
```

Exercise 3. Write a script to compute an interpolant for

$$\begin{aligned} A &\equiv \{(x + y \leq 0) \wedge (-2x - y + z \leq -5)\} \\ B &\equiv \{(x - z \leq 3)\} \end{aligned}$$

where x, y, z are variables of type `Real` (i.e., logic `QF_LRA` has to be used) and run it through OpenSMT.

3 Application to Program Verification

Before discussing the use of interpolants in program verification with model checking, we need to give a more generic definition of interpolants. The definition involves several ($n \geq 2$) partitions $A_1, \ldots, A_n$ whose conjunction is unsatisfiable, and defines $n+1$ interpolants $I_0, \ldots, I_n$ such that:

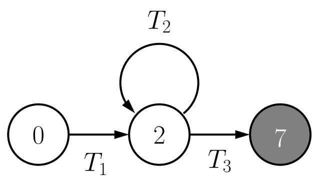
(*i*) $I_0 = \top$, $I_n = \bot$;
(*ii*) $T \vdash (I_k \wedge A_{k+1}) \rightarrow I_{k+1}$;
(*iii*) I_k defined over shared symbols of A_k and A_{k+1}.

Intuitively, each A_k is used to encode a set of instructions in the program, while interpolants represent overapproximations of postconditions, resulting from the application of the instructions.

We show the application of interpolants by means of an example. Consider the following program fragment defined over integer variables

```
1: y = x;
2: while ( x >= 1 ) {
3:   x = x - 1;
4:   y = y - 1;
5: }
6: if ( y >= 1 )
7:   ERROR;
```

which contains an error location at line 7. The location is however unreachable. In fact if $x \leq 0$, then the loop is not taken, and since $y = x$ by the first instruction, the if branch is also not taken. If $x \geq 1$ instead, the loop will decrease both x and y initially equal, and so when the loop condition is falsified the value of y will prevent taking the if branch. In order to formally analyze the program, we need to extract its control-flow graph and translate its instructions into a transition system, as shown in Figure 2 below.



$$T_1\colon \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Fig. 2. Control-flow graph and transitions.

This translation process may be carried out in several ways, for instance T_2 could have been split into a T_2' updating only x and a T_2'' updating only y. Notice that some translations are better than others, w.r.t. the way the lazy abstraction framework performs.

Once the translation is done, we may start building the unwinding of the program, based on the control graph. We begin from location 0 (actually not listed in the program, we assume it is the initial state) and following the control-flow graph we unwind T_1 and T_3 (Figure 3a).

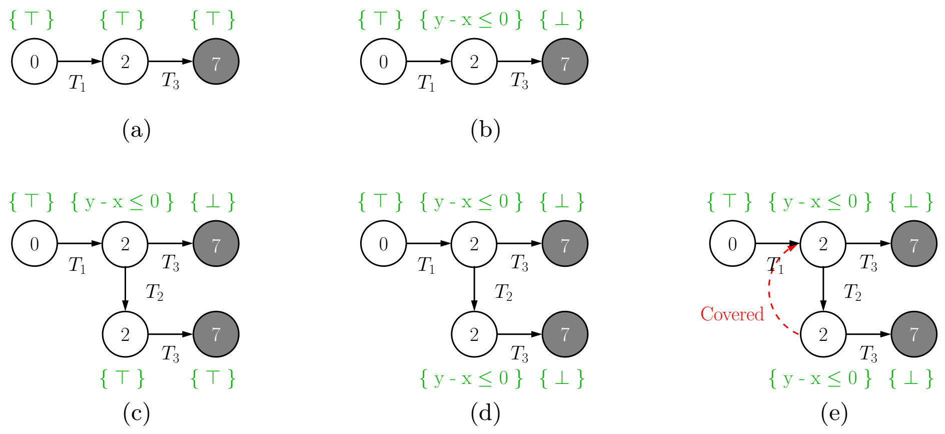


Fig. 3. Unwinding of the program.

Notice that each node has a *label* on top of it, within curly brackets. In the lazy abstraction labels are initially abstracted away to $\top$. The provided trace however is infeasible. It can be checked by means of a symbolic execution of the formula in Figure 4a. The interpolants we may compute "around" the two partitions are shown in Figure 4b, and result into the unwinding in Figure 3.

$$T_1 \begin{cases} \top \\ x_1 = x_0 \\ y_1 = x_0 \end{cases}$$

$$T_3 \begin{cases} x_1 \leq 0 \\ y_1 \geq 1 \\ x_2 = x_1 \\ y_2 = y_1 \end{cases}$$

(a)

$$\{\top\}$$
$$T_1 \begin{cases} \top \\ x_1 = x_0 \\ y_1 = x_0 \end{cases}$$
$$\{y_1 - x_1 \leq 0\}$$
$$T_3 \begin{cases} x_1 \leq 0 \\ y_1 \geq 1 \\ x_2 = x_1 \\ y_2 = y_1 \end{cases}$$
$$\{\bot\}$$

(b)

Fig. 4. Counterexample for the first trace.

Another round of abstraction-refinement will produce the unwinding in Figure 3d, and we may stop expanding the tree because of covering (Figure 3e). For details refer to [16].

4 Computing Interpolants

4.1 Quantifier Elimination

In a recursively enumerable theory T there is a tight connection between the property of eliminating quantifiers and that of being quantifier-free interpolating: the two are in fact equivalent [13]. This means that, in principle, quantifier elimination algorithms can be used to compute interpolants. However, it is well known that quantifier elimination techniques are very expensive: in fact the whole idea about interpolants is to avoid quantifier elimination !

Consider the interpolation problem $A \wedge B$, and let $\boldsymbol{a}$ the tuple of variables local to A: an interpolant can be obtained by performing quantifier elimination on $\exists \boldsymbol{a}.A$, to obtain an interpolant I. Notice that I is the strongest possible interpolant, because quantifier elimination preserves equivalence (i.e., $\exists \boldsymbol{a}.A \leftrightarrow I$), and therefore we cannot find a stronger formula that is still implied by A.

4.2 Extension of known techniques

The common way to obtain interpolating procedures is to modify existing ones. There are two school of thoughts in this respect. In the first class of approaches, a calculus or a set of inference rules is extended with side conditions that describe how the interpolant is modified during the proof [1, 15]. For example, in [1] a sequent-like calculus for $\mathcal{LIA}$ is extended as shown in Figure 5.

Exercise 4. Show that the rule in Figure 5b produce a correct (partial) interpolant, provided that $b_1 \vee b_2$ are local to B and that I_1 and I_2 are correct partial interpolants for the premises.

$$\frac{\Gamma, b_1 \vdash \Delta \qquad \Gamma, b_2 \vdash \Delta}{\Gamma, b_1 \vee b_2 \vdash \Delta}\ \vee\text{-Left}$$

(a)

$$\frac{\Gamma, b_1 \vdash \Delta \parallel I_1 \qquad \Gamma, b_2 \vdash \Delta \parallel I_2}{\Gamma, b_1 \vee b_2 \vdash \Delta \parallel I_1 \wedge I_2}\ \vee\text{-Left}$$

(b)

Fig. 5. Extension of a sequent calculus. The constraints for the partitions A and B are "hidden" inside the sequents $\Gamma \vdash \Delta$.

The other method to obtain interpolants is to modify an existing decision procedure [9, 11, 19]. Figure 6 shows how a witness of unsatisfiability extracted from the simplex algorithm can be used to compute an interpolant.

A	$x + y + z \leq 0$	1
	$-2y + 3z \leq 0$	1/2
B	$-\frac{3}{5}x - \frac{3}{2}z \leq -3$	5/3

(a)

A	$x + y + z \leq 0$	1
	$-2y + 3z \leq 0$	1/2
I	$x + \frac{5}{2}z \leq 0$	
B	$-\frac{3}{5}x - \frac{3}{2}z \leq -3$	5/3

(b)

Fig. 6. Interpolant from Simplex proof.

5 The Two-Provers Paradigm

Interpolation can be seen as a game between two entities, a prover for A and a prover for B. This view has first proposed in [11] for the theory of equality with uninterpreted functions. The idea is that starting from a pair $\langle A, B\rangle$, each prover in turn is allowed to perform a step to produce a new pair $\langle A', B'\rangle$. Steps could be of different types, for instance, if $A \vdash \phi$ *derive* something in A to get $\langle A \cup \{\phi\}, B\rangle$. Another step could be a *propagation* of a formula $\phi \in A$ to B, if ϕ is on the shared language. This view is very similar to that of the Nelson-Oppen schema for combining theories.

Now suppose that after a number of steps we derive $\bot$ in A or B, then we can reconstruct an interpolant for the original pair $\langle A, B\rangle$ by giving an *interpolating instruction* to each of the steps. The step plus the interpolating instruction is called interpolating *metarule*: metarules that suffice in most cases are shown in table 1 (notice that what is shown in the table is slightly different from what you can find in the slides, but the idea is exactly the same, it is just a syntactic difference).

Notice that the rules do not provide a complete calculus, nor they suggest any strategy to be applied to reach unsatisfiabililty. The idea is that taken any

Close1	Close2	Propagate1	Propagate2
$\dfrac{}{A \mid B}$ *Prv.*: A is unsat. *Int.*: $\phi' \equiv \bot$.	$\dfrac{}{A \mid B}$ *Prv.*: B is unsat. *Int.*: $\phi' \equiv \top$.	$\dfrac{A \mid B \cup \{\psi\}}{A \mid B}$ *Prv.*: $A \vdash \psi$ and ψ is AB-common. *Int.*: $\phi' \equiv \phi \wedge \psi$.	$\dfrac{A \cup \{\psi\} \mid B}{A \mid B}$ *Prv.*: $B \vdash \psi$ and ψ is AB-common. *Int.*: $\phi' \equiv \psi \to \phi$.

Define0	Define1	Define2
$\dfrac{A \cup \{a = t\} \mid B \cup \{a = t\}}{A \mid B}$ *Prv.*: t is AB-common, a fresh. *Int.*: $\phi' \equiv \phi(t/a)$.	$\dfrac{A \cup \{a = t\} \mid B}{A \mid B}$ *Prv.*: t is A-local and a is fresh. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B \cup \{a = t\}}{A \mid B}$ *Prv.*: t is B-local and a is fresh. *Int.*: $\phi' \equiv \phi$.

Disjunction1	Disjunction2
$\dfrac{\cdots \quad A \cup \{\psi_k\} \mid B \quad \cdots}{A \mid B}$ *Prv.*: $\bigvee_{k=1}^{n} \psi_k$ is A-local and $A \vdash \bigvee_{k=1}^{n} \psi_k$. *Int.*: $\phi' \equiv \bigvee_{k=1}^{n} \phi_k$.	$\dfrac{\cdots \quad A \mid B \cup \{\psi_k\} \quad \cdots}{A \mid B}$ *Prv.*: $\bigvee_{k=1}^{n} \psi_k$ is B-local and $B \vdash \bigvee_{k=1}^{n} \psi_k$. *Int.*: $\phi' \equiv \bigwedge_{k=1}^{n} \phi_k$.

Redplus1	Redplus2	Redminus1	Redminus2
$\dfrac{A \cup \{\psi\} \mid B}{A \mid B}$ *Prv.*: $A \vdash \psi$ and ψ is A-local. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B \cup \{\psi\}}{A \mid B}$ *Prv.*: $B \vdash \psi$ and ψ is B-local. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B}{A \cup \{\psi\} \mid B}$ *Prv.*: $A \vdash \psi$ and ψ is A-local. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B}{A \mid B \cup \{\psi\}}$ *Prv.*: $B \vdash \psi$ and ψ is B-local. *Int.*: $\phi' \equiv \phi$.

ConstElim1	ConstElim2	ConstElim0
$\dfrac{A \mid B}{A \cup \{a = t\} \mid B}$ *Prv.*: a is A-strict and does not occur in A, t. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B}{A \mid B \cup \{b = t\}}$ *Prv.*: b is B-strict and does not occur in B, t. *Int.*: $\phi' \equiv \phi$.	$\dfrac{A \mid B}{A \cup \{c = t\} \mid B \cup \{c = t\}}$ *Prv.*: c, t are AB-common, c does not occur in A, B, t. *Int.*: $\phi' \equiv \phi$.

Table 1. Interpolating Metarules: each rule has a proviso *Prv.* and an instruction *Int.* for recursively computing the new interpolant ϕ' from the old one(s) $\phi, \phi_1, \ldots, \phi_k$. These rules appeared in [4].

algorithm or calculus that leads to unsatisfiable state, one can reconstruct an interpolant by matching each step of the algorithm/calculus with one or more metarule, without modifying the algorithm/calculus directly.

Example 6. Consider again the partitions from Example 2.

$$A \equiv \{\neg a \wedge (a \vee c_1) \wedge (a \vee c_2)\},\ B \equiv \{\neg b \wedge (b \vee \neg c_2) \wedge (b \vee \neg c_2)\}$$

There are several strategies that can lead to unsatisfiability. For instance

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle \ldots \wedge c_1, \ldots \wedge \bot \rangle}\ (\text{Close2})}{\langle \ldots \wedge c_1, \neg b \wedge (b \vee \neg c_1) \wedge (b \vee \neg c_2) \wedge c_1 \wedge b \rangle}\ (\text{RedPlus2})}{\langle \ldots \wedge c_1, \neg b \wedge (b \vee \neg c_1) \wedge (b \vee \neg c_2) \wedge c_1 \rangle}\ (\text{RedPlus2})}{\langle \ldots \wedge c_1, \neg b \wedge (b \vee \neg c_1) \wedge (b \vee \neg c_2) \rangle}\ (\text{Propagate1})}{\langle \neg a \wedge (a \vee c_1) \wedge (a \vee c_2), \neg b \wedge (b \vee \neg c_1) \wedge (b \vee \neg c_2) \rangle}\ (\text{RedPlus1})$$

Exercise 5. Compute the interpolant using the metarules for the derivation strategy in the previous example.

Exercise 6. Find alternative strategies and compute corresponding interpolants.

Exercise 7. Try to apply the metarules approach to the algorithms and calculi in [1, 9, 11].

References

1. A. Brillout, D. Kroening, P. Rümmer, and W. Thomas. An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic . In *IJCAR*, 2010.
2. A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic . In *VMCAI*, 2011.
3. R. Bruttomesso. An Extension of the Davis-Putnam Procedure and its Application to Preprocessing in SMT. In *SMT*, 2009.
4. R. Bruttomesso, S. Ghilardi, and S. Ranise. Rewriting-based Quantifier-free Interpolation for a Theory of Arrays. In *RTA*, 2011.
5. R. Bruttomesso, E. Pek, and N. Sharygina. A Flexible Schema for Generating Explanations in Lazy Theory Propagation. In *MEMOCODE*, 2010.
6. R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The OpenSMT Solver. In *TACAS*, pages 150–153, 2010.
7. R. Bruttomesso, S. Rollini, N. Sharygina, and A. Tsitovich. Flexible Interpolation with Local Proof Transformations. In *ICCAD*, 2010.
8. R. Bruttomesso and N. Sharygina. A Scalable Decision Procedure for Fixed-Width Bit-Vectors. In *ICCAD*, 2009.
9. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, pages 397–412, 2008.
10. W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, pages 269–285, 1957.
11. A. Fuchs, A. Goel, J. Grundy, S. Krstić, and C. Tinelli. Ground Interpolation for the Theory of Equality. In *TACAS*, pages 413–427, 2009.
12. T. Henzinger and K. L. McMillan R. Jhala, R. Majumdar. Abstractions from Proofs. In *POPL*, 2004.
13. D. Kapur, R. Majumdar, and C. Zarba. Interpolation for Data Structures. In *SIGSOFT'06/FSE-14*, pages 105–116, 2006.

14. K. L. McMillan. Interpolation and SAT-Based Model Checking. In *CAV*, pages 1–13, 2003.
15. K. L. McMillan. An Interpolating Theorem Prover. In *TACAS*, pages 16–30, 2004.
16. K. L. McMillan. Lazy Abstraction with Interpolants. In *CAV*, 2006.
17. S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.
18. S. Rollini, R. Bruttomesso, and N. Sharygina. An Efficient and Flexible Approach to Resolution Proof Reduction. In *HVC*, 2010.
19. G. Yorsh and M. Musuvathi. A Combination Method for Generating Interpolants. In *CADE*, pages 353–368, 2005.