Instantiations, Zippers and EPR Interpolation

Nikolaj Bjørner, Arie Gurfinkel, Konstantin Korovin and Ori Lahav

Microsoft Research, Software Engineering Institute, University of Manchester, Tel Aviv University

Abstract

This paper describes interpolation procedures for EPR. In principle, interpolation for EPR is simple: It is a special case of first-order interpolation. In practice, we would like procedures that take advantage of properties of EPR: EPR admits finite models and those models are sometimes possible to describe very compactly. Inspired by procedures for propositional logic that use models and cores, but not proofs, we develop a procedure for EPR that uses just models and cores.

1 Introduction

EPR has interesting applications in shape [7] and hardware analysis [5]. EPR offers a degree of succinctness over propositional logic that can be a significant advantage. EPR is often also hidden in plain sight [9]: bit-vector logic turns out to be much more succinct than propositional logic and can be translated polynomially back and forth into EPR. Current applications in verification uses EPR as a language for describing assertions and systems. We can go much further, however, and use EPR as a target of invariant synthesis. In other words, we are interested in automatically inferring invariants that can be expressed in EPR for systems whose transitions and assertions are already expressed in EPR. An important tool for formula based invariant synthesis is to use interpolants [10], and as the reference indicates EPR interpolants from Z3 proofs is a solved problem (presuming interpolants are also EPR formulas). Nevertheless, we press the issue and ask the question if there are interpolation procedures for EPR that can take advantage of model and core producing facilities, in contrast to relying on resolution proofs. We call the resulting procedure *zipper interpolation* because it zips between formulas A and B.

This paper considers the problem of computing so called reverse interpolants. Given two formulas A and B a *reverse interpolant* between A and B is a formula I in the shared signature such that $A \implies I$ and $B \wedge I$ is unsatisfiable.

Example 1. *The formula I is a reverse interpolant of formulas A and B:*

$$\underbrace{\exists y \;.\; \forall x \;.\; p(y,x)}_{I} : \quad \underbrace{\forall x \;.\; p(a,x)}_{A} \;\wedge\; \underbrace{\forall x \;.\; \neg p(x,b)}_{B}$$

$$\underbrace{\exists x, y \;.x \neq y}_{I} : \quad \underbrace{a_1 \neq a_2}_{A} \;\wedge\; \underbrace{\forall x, y \;.\; x = y}_{B}$$

This short paper is organized as follows. We first describe a straight-forward Inst-Gen interpolation procedure that is bootstrapped on top of propositional interpolation. Section 3 reviews methods for propositional interpolation that are based on models and cores. Section 4 extends the approach for propositional interpolants to EPR.

2 Bootstrapped Inst-Gen Interpolation

Inst-Gen is an instantiation-based search procedure. It is complete for first-order logic [8]. In a nutshell, given a set of clauses S, Inst-Gen is searching for a ground witness for unsatisfiability

of S as follows. First, a ground abstraction $S\bot$ of S is obtained by mapping all variables in S into a designated constant $\bot$. If $S\bot$ is unsatisfiable then S is also unsatisfiable. Otherwise, new (possibly non-ground) instances of clauses in S are generated, guided by a propositional model of $S\bot$ and added to S. The process is repeated until either an unsatisfiable ground abstraction is obtained or a saturation is reached. The process (together with redundancy elimination) is sound and complete for first-order logic. In particular, S is unsatisfiable if and only if Inst-Gen terminates with a set of possibly non-ground instantiations S' of clauses from S such that $S'\bot$ is unsatisfiable. It is easy to see that Inst-Gen is a decision procedure for the clausal EPR fragment.

The *Inst-Gen bootstrapped interpolation procedure* will be parametrised by a complete interpolation procedure for propositional logic $\mathcal{IP}$. We assume that for any sets of propositional clauses A and B such that $A \wedge B$ is unsatisfiable $\mathcal{IP}(A, B)$ returns a propositional formula I over shared propositional variables such that $A \implies I$ and $I \wedge B$ is unsatisfiable. Propositional interpolation procedures have been intensively investigated in the recent years. We will not distinguish between ground atoms and propositional variables they represent. Therefore we assume that a propositional interpolation procedure applied to ground formulas returns a ground interpolant.

The Inst-Gen interpolation procedure consists of the following steps. Consider sets of EPR clauses A and B.

1. Apply the Inst-Gen procedure to $A \cup B$ obtaining a set of clauses $A' \cup B'$ such that i) A' is a set of instances of A and B' is a set of instances of B, ii) $A'\bot \cup B'\bot$ is satisfiable if and only if $A \cup B$ is satisfiable. If $A \cup B$ is satisfiable return the model. Otherwise got to step 2.

2. Let $I_g(\bar{a}, \bar{b}, \bar{c}, \bot)$ be the ground interpolant returned by $\mathcal{IP}(A'\bot, B'\bot)$, where $\bar{a}$ are constants local to A, $\bar{b}$ constants local to B, and $\bar{c}$ are shared constants. Then, an EPR formula $I = \exists \bar{a}\ \forall \bar{b}\ \forall \bot\ I_g(\bar{a}, \bar{b}, \bar{c}, \bot)$ is the required reverse interpolant.[1]

Theorem 1. *The Inst-Gen bootstrapped interpolation procedure computes a reverse EPR interpolant between A and B.*

3 Propositional Zipper Interpolation

We now describe a simple approach to obtain interpolants for Propositional logic. It is a variant of [4] and is a special case of interpolants generated as a side-effect of IC3 [3, 2]. Variants and extensions have been examined for richer theories, such as arithmetic [1, 6]. Let A and B be two propositional formulas, whose conjunction is unsatisfiable. Algorithm 1 procedure computes an interpolant I using only models and unsatisfiable cores. The dual, Algorithm 2, uses models from A instead of B and computes a disjunctive interpolant.

Theorem 2. *The Propositional Zipper algorithms compute a reverse propositional interpolant between A and B.*

The propositional Zipper interpolation algorithms can be used as part of the solution in Section 2.

[1] Notice that we quantify over $\bar{a}$ and $\bar{b}$ that were previously free in the formulas A and B. Identifiers that are free in formulas are called *constants*; identifiers that are bound by quantifiers become bound variables.

Initialize $I := \top$

1. If $B \wedge I$ is unsatisfiable, return I.
2. Extract a model M for $B \wedge I$.
3. Let L be a set of literals consisting of: the atomic formulas of B which hold in M, and the negations of the atomic formulas of B which do not hold in M.
4. Check satisfiability of $A \wedge L$. If satisfiable, then $A \wedge B$ are satisfiable, and there is no interpolant. Otherwise, let C be the unsatisfiable core from L.
5. Let C' be the literals in C consisting of atomic formulas from A.
6. Update $I := I \wedge \neg C'$. Go to step 1.

Algorithm 1: Propositional Zipper Interpolation

Initialize $I := \bot$

1. If $A \wedge \neg I$ is unsatisfiable, return I.
2. Extract a model M for $A \wedge \neg I$.
3. Let L be a (prime) implicant of A consistent with M.
4. Check satisfiability of $B \wedge L$. If satisfiable, then $A \wedge B$ are satisfiable, and there is no interpolant. Otherwise, let C be the unsatisfiable core from L.
5. Let C' be the literals in C consisting of atomic formulas from A.
6. Update $I := I \vee C'$. Go to step 1.

Algorithm 2: Dual Propositional Zipper Interpolation

4 EPR Zipper Interpolation

Satisfiable EPR formulas have small models whose domain is the set of constants used in the formulas. The models can be expressed as a finite conjunction of literals where the literals just have to completely cover the set of combinations of arguments to each predicate. Representing models in this way is of course impractical for large domains or predicates with large arities. It is often possible to represent models in a much more economical way when predicates are almost always false or almost always true. In this case it suffices to represent just the cases where the predicate is true (respectively false), and the default case can be summarized as the complement. Z3 uses this model representation and leverages it for model-based quantifier instantiation. Let us here describe these models in the context of EPR as what we call *defaulted cubes*:

Definition 1 (Defaulted Cube)**.** *A pair $\langle L, D\rangle$ is called a defaulted cube over a vocabulary Σ if L is a conjunction of ground literals over Σ, and D is a conjunction of formulas of the following form: For every predicate p of arity n in Σ, D includes a conjunct of the form*

$$\forall \overline{x}.\neg Ground_p^L \rightarrow p(\overline{x}) \quad or \quad \forall \overline{x}.\neg Ground_p^L \rightarrow \neg p(\overline{x}) \ ,$$

where $Ground_p^L$ *is the disjunction of all formulas of the form* $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ *for every ground terms* $t_1 \ldots t_n$ *such that either* $p(t_1 \ldots t_n)$ *or* $\neg p(t_1 \ldots t_n)$ *occur in* L.

4.1 Constructing Defaulted Cubes

We here show how to construct defaulted cubes from models for a formula B in such a way that the constructed defaulted cube remains an implicant of B. Roughly, Z3 models (we use $\mathcal{M}$ to refer to models) for EPR formulas are of the form

$$
\begin{array}{lcl}
p(\bar{x}) & := & p'(\pi(\bar{x}))
\end{array}
\qquad
\begin{array}{lcl}
\pi(\bar{x}) & := & \textbf{match } \bar{x} \textbf{ with} \\
 & & \mid \bar{s}_1 \ \rightarrow \ \bar{t}_1 \\
 & & \mid \bar{s}_2 \ \rightarrow \ \bar{t}_2 \\
 & \vdots & \\
 & & \mid \_ \ \rightarrow \ \bar{t}_n
\end{array}
\qquad
\begin{array}{lcl}
p'(\bar{x}) & := & \textbf{match } \bar{x} \textbf{ with} \\
 & & \mid \bar{t}_1 \ \rightarrow \ \top \\
 & & \mid \bar{t}_2 \ \rightarrow \ \bot \\
 & \vdots & \\
 & & \mid \_ \ \rightarrow \ \top
\end{array}
$$

In other words, Z3 creates interpretations of EPR predicates as a composition of a projection operation π and an interpretation values in the projection. Models can be inspected, so we can enumerate the terms $\bar{s}_1, \bar{s}_2, \ldots$ and extract truth values for p on the non-default branches. The default evaluation then becomes the value of $p'(\bar{t}_n)$.

The model construction can also be minimized by using information about B. The construction works by iterating over the clauses in B.

1. Set $L_B := \top$ and $D_B := \bigwedge_{p \in \Sigma} \forall \bar{x} \ . \ \pm p(\bar{x})$, where the sign of p in D_B is given by the default branch in $\mathcal{M}$ (this is easy to obtain).

2. For ground clauses, pick some literal that is true in the model $\mathcal{M}$ and add it to L_B.

3. For non-ground clauses of the form $\forall \bar{y}. \ell_1 \vee \ldots \vee \ell_n$ we repeat a refinement loop as follows: Suppose $L_B \wedge D_B \wedge \bigwedge_{1 \leq i \leq n} \neg \ell_i[\bar{y}]$ is satisfiable in a model over the vocabulary of B (we assume Σ contains at least one constant). Say these values are $\bar{b}$. Choose a literal $\ell_i[\bar{b}]$ that is true in $\mathcal{M}$ and add it to L_B.

Example 2. *Let* $B = \neg p(b,b) \wedge (\forall x \ . \ p(a,x))$. *A model for* B *may consist of two element* $\{a,b\}$ *and assign* p *to be always* $\top$ *except for* $p(b,b) = \bot$. *Thus we begin with* $L_B := \top$ *and* $D_B := \forall x,y \ . \ p(x,y)$. *Then the ground clause* $\neg p(b,b)$ *is added to* L_B. *For the non-ground clause* $\forall x \ . \ p(a,x)$, *we have that* $L_B \wedge D_B \wedge \neg p(a,x)$ *is satisfiable, by a model with one element* $\{b\}$ *and* $p(b,b) = \bot$, *and an assignment of* x *to* b. *Thus we add* $p(a,b)$ *to* L_B, *and obtain that* $L_B \wedge D_B$ *imply* B.

4.2 Algorithm

We now describe an algorithm that is based on Algorithm 1 but works directly with an EPR decision procedure. It does not require assembling instantiations provided by the decision procedure. Instead it relies on models and cores. The main idea of the algorithm is to incrementally extract a ground model for $A \wedge B$ using models for B and models for A. The intermediary ground models may not extend to full models so the algorithm has to *repair* such intermediary models. Algorithm 3 contains the overall description and we motivate it in the following.

Let us illustrate one aspect of the algorithm using an example.

1. Initialize $I := \top$.
2. While $B \wedge I$ is satisfiable, do:
   (a) Construct $\langle L_B, D_B \rangle$ such that $L_B \wedge D_B$ implies $B \wedge I$.
   (b) Initialize $J := \top$.
   (c) While $L_B \wedge J \wedge A$ is satisfiable, do:
      i. Extend L_B to L_A and a D_A such that $\langle L_A, D_A \rangle$ implies A. Add also definitions for predicates from D_B that are not mentioned in A, so that we can use D_A to evaluate B.
      ii. If $L_A \wedge D_A$ implies B, then $A \wedge B$ is satisfiable. Exit.
      iii. If $L_A \wedge B$ is unsatisfiable, then let C be a minimal subset of $L_A \setminus L_B$ such that $C \wedge L_B \wedge B$ is unsatisfiable. Update $J := J \wedge \neg C$.
      iv. If $L_A \wedge B$ is satisfiable and L_A is *complete* for Σ_{AB}, then $A \wedge B$ is satisfiable. Exit.
      v. Otherwise, $L_A \wedge D_A$ implies $\neg B$:
      Then there is some conjunct $\forall \bar{b}.\ell_1 \vee \ldots \vee \ell_n$ in B that is false in the model induced by $L_A \wedge D_A$ ($\ell_1 \ldots \ell_n$ are literals). In other words,
      $L_A \wedge D_A \wedge \bigwedge_{1 \leq i \leq n} \neg \ell_i[\bar{y}]$ is satisfiable and we can extract an assignment $\bar{d}$ for $\bar{y}$ by inspecting $\langle L_A, D_A \rangle$ as we did when extracting minimized defaulted cubes from models. One of the literals $\ell_i[\bar{d}]$ is going to evaluate to $\top$ under D_B (because $\langle L_B, D_B \rangle$ is a model for B). Pick this literal and update $L_B := L_B \wedge \ell_i[\bar{d}]$ and continue the loop (at step 2 (c)).
   (d) Extract a minimal subset C from $L_B \wedge J$, such that $C \wedge A$ is unsatisfiable.
   (e) Update $I := I \wedge \forall \bar{b}\ \neg C$, where $\bar{b}$ are the constants in B that do not occur in A.
3. Return $\exists \bar{a}\ I$, where $\bar{a}$ are the constants in A that do not occur in B.

Algorithm 3: EPR Zipper Interpolation Algorithm

Example 3. *Take $A := q(c)$ and $B := p(c) \wedge \forall x\ .\ \neg p(x) \vee \neg q(x)$. Suppose that initially $L_B := \top$ and $D_B := \forall x\ .\ p(x), \forall x\ .\ \neg q(x)$. Then L_A is still $\top$ and $D_A := \forall x\ .\ q(x), \forall x\ .\ p(x)$. Note that D_A inherits interpretations of predicates that are mentioned in B but not mentioned in A. In this case $L_A \wedge B$ is satisfiable, but L_A is not complete and $L_A \wedge D_A$ implies $\neg B$. Therefore the last case in step 2 (c) applies. The last case checks satisfiability of $D_A \wedge p(y) \wedge q(y)$ with an interpretation where $y := c$. The assignment $\neg p(c)$ is inconsistent with $\langle L_B, D_B \rangle$ because D_B requires $\forall x\ .\ p(x)$. On the other hand, the assignment $\neg q(c)$ is consistent with D_B. Therefore we add $\neg q(c)$ to L_B and repeat step 2 (c). This time, $L_B \wedge A$ is unsatisfiable, and the resulting interpolant I is (simply) $q(c)$. Furthermore $B \wedge I$ is also unsatisfiable, so the algorithm terminates.*

The algorithm terminates at the latest when all groundings have been created. This criteria is based on the following definition and observation:

Definition 2 (Ground complete sets of literals)**.** *We say that a set of literals L is* complete *with respect to a vocabulary Σ if for every ground $p(\bar{t})$ over Σ, L contains either $p(\bar{t})$ or its negation.*

Proposition 1. *A ground complete set of literals L that is consistent with an EPR formula φ can be extended to a model for φ.*

Proof. The only thing that is not forced by L is the domain. Set the domain to be the term model given by Σ. Then the fact that $L \wedge \varphi$ is satisfiable means that every ground clause from φ contains a literal in L. □

However, terminating by the time of ground complete literals is only a worst case scenario. There are other cases where the exchange of facts between A and B can ensure termination. In contrast to Algorithm 1 we drop the requirement that literals created from B are implicants of B. This means that A may extend the partial assignment learned from B in a way that is inconsistent with B. To avoid such extensions, the new algorithm (Algorithm 3) accumulates a set of constraints that we call J that prevent A to extract such assignments. Recall that the idea of Algorithm 1 is to extract consequences of A, so the new algorithm has to ensure A eventually learns such a consequent.

Example 4 (Using J)**.** *Take the same example as before: $A := q(c)$ and $B := p(c) \wedge \forall x \,.\, \neg p(x) \vee \neg q(x)$. As before $L_B := \top$ and $D_B := \forall x \,.\, p(x), \forall x \,.\, \neg q(x)$ but this time $L_A := q(c)$ and $D_A := \forall x \,.\, x \neq c \rightarrow \neg q(x), \forall x \,.\, p(x)$. It is now the case that $L_A \wedge B$ is unsatisfiable and the subset of L_A that is used for unsatisfiability comprises of $q(c)$. Set $J := \neg q(c)$ and repeat step 2 (c). This time $L_B \wedge J \wedge A$ is unsatisfiable and the core from $L_B \wedge J$ is $\neg q(c)$. The interpolant is updated $I := q(c)$ and the algorithm terminates because $B \wedge I$ is unsatisfiable.*

Note that in general J is a conjunction of clauses (not just literals), whereas L_A and L_B are conjunctions of literals.

Example 5. *Take $A := \forall x \,.\, p(a, x)$ and $B := \forall x \,.\, \neg p(x, b)$. Suppose that initially $L_B := \top$ and $D_B := \forall x, y \,.\, \neg p(x, y)$. Then L_A is still $\top$ and $D_A := \forall x, y \,.\, p(x, y)$. Thus $L_A \wedge B$ is satisfiable, but L_A is not complete and $L_A \wedge D_A$ implies $\neg B$. Therefore the last case in step 2 (c) applies. This checks satisfiability of $D_A \wedge p(x, b)$. A model may assign x to b, resulting in $L_B := \neg p(b, b)$, and we return to step 2 (c). Still $L_B \wedge J \wedge A$ is satisfiable. Now there is no D_A such that $L_B \wedge D_A$ would imply A, so L_B should be extended, and we may obtain $L_A := \neg p(b, b) \wedge p(a, a)$ and $D_A := \forall x, y \,.\, (x \neq a \vee y \neq a) \wedge (x \neq b \vee y \neq b) \rightarrow p(x, y)$. Again we arrive at step 2 (c), that results with $L_B := \neg p(a, b)$. Now $L_B \wedge J \wedge A$ is unsatisfiable, and $C = \neg p(a, b)$ is a minimal subset from $L_B \wedge J$, such that $C \wedge A$ is unsatisfiable. We obtain $I := \exists a \forall b \,.\, p(a, b)$, and terminate since $B \wedge I$ is unsatisfiable.*

Note that the algorithm uses minimal cores. Using minimal cores is an automating way to ensure that the predicates are in the shared vocabulary between A and B. In summary,

Theorem 3. *The EPR Zipper algorithm computes a reverse EPR interpolant between A and B.*

5 Summary

We developed a method for EPR interpolation based on models and cores. It zips between satisfiability checks for formulas A and B. In contrast to propositional interpolants we suggested to relax the use of implicants. Instead, the procedure communicates sets of literals that are consistent with the formulas A or B, and refines these literals until either augmenting the generated interpolant or determining that the formulas are satisfiable. In future work we are interested in extending the notion of EPR Zipper Interpolants to somewhat richer logics that include arithmetic or arrays.

References

[1] A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In Sharygina and Veith [11], pages 313–329.

[2] S. Bayless, C. Val, T. Ball, H. Hoos, and A. Hu. Efficient Modular SAT Solving for IC3. 2013.

[3] A. R. Bradley. SAT-Based Model Checking without Unrolling. In *VMCAI*, pages 70–87, 2011.

[4] H. Chockler, A. Ivrii, and A. Matsliah. Computing interpolants without proofs. In A. Biere, A. Nahir, and T. Vos, editors, *Hardware and Software: Verification and Testing*, volume 7857 of *Lecture Notes in Computer Science*, pages 72–85. Springer Berlin Heidelberg, 2013.

[5] M. Emmer, Z. Khasidashvili, K. Korovin, C. Sticksel, and A. Voronkov. Epr-based bounded model checking at word level. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 2012.

[6] K. Hoder and N. Bjørner. Generalized Property Directed Reachability. In *SAT*, pages 157–171, 2012.

[7] S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In Sharygina and Veith [11], pages 756–772.

[8] K. Korovin. Inst-Gen - a modular approach to instantiation-based automated reasoning. In A. Voronkov and C. Weidenbach, editors, *Programming Logics*, volume 7797 of *Lecture Notes in Computer Science*, pages 239–270. Springer, 2013.

[9] G. Kovásznai, A. Fröhlich, and A. Biere. BV2EPR: A Tool for Polynomially Translating Quantifier-Free Bit-Vector Formulas into EPR. In M. P. Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 443–449. Springer, 2013.

[10] K. L. McMillan. Interpolants from Z3 proofs. In P. Bjesse and A. Slobodová, editors, *FMCAD*, pages 19–27. FMCAD Inc., 2011.

[11] N. Sharygina and H. Veith, editors. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.