Efficient Interpolation for the Theory of Arrays

(work in progress)

Jochen Hoenicke and Tanja Schindler*

Department of Computer Science,
University of Freiburg
{hoenicke,schindle}@informatik.uni-freiburg.de

Abstract

Existing techniques for Craig interpolation for the quantifier-free fragment of the theory of arrays require a special solver. The solver needs to know in advance the partitioning (A, B) of the interpolation problem and needs to avoid creating AB-mixed terms to be suitable for interpolation. This limits the efficiency of these solvers especially when computing sequence and tree interpolants. We present a new approach using Proof Tree Preserving Interpolation and an array solver based on Weak Equivalence on Arrays. We give an interpolation algorithm for the lemmas produced by the array solver.

1 Introduction

Several model-checkers [1, 2, 5, 11, 12, 13, 14, 15, 17, 18] use interpolants to find candidate invariants to prove the correctness of software. They require efficient tools to check satisfiability of a formula in a decidable theory and to compute interpolants (usually sequence or tree interpolants) for unsatisfiable formulas. Moreover, they often need to combine several theories, e.g., integer or bitvector theory for reasoning about numeric variables and array theory for reasoning about pointers. In this paper we present an interpolation procedure for the quantifier-free fragment of the theory of arrays that allows for the combination with other theories and that can reuse an existing unsatisfiability proof to compute interpolants efficiently.

Our method is based on the array solver presented in [6], which fits well into existing Nelson-Oppen frameworks. The solver generates lemmas (valid in the theory of arrays) that explain equalities between variables shared between different theories. The variables do not necessarily belong to the same formula in the interpolation problem and the solver does not need to know the partitioning. Instead, we use the technique of Proof Tree Preserving Interpolation [9], which can produce interpolants from existing proofs propagating equalities between symbols from different partitions.

The contribution of this paper is an algorithm to interpolate the lemmas produced by the solver of the theory of arrays. This solver only produces two types of lemmas, namely a variant of the read-over-write axiom and a variant of the extensionality axiom. However, the lemmas can contain array store chains of arbitrary length which need to be handled by the interpolation procedure. Bruttomesso et al. [4] showed that adding a diff function is sufficient to get a theory of arrays that is closed under interpolation and in principle their algorithm can be used to interpolate the lemmas. We give a more efficient algorithm that exploits the special shape of these axioms.

Related Work. Brillout et al. [3] give an interpolation procedure for the combination of Presburger Arithmetic and the quantifier-free theory of arrays. However, their procedure produces quantified interpolants. For certain cases, their algorithm removes quantifiers by simple syntactic transformations, but in general this is not possible.

*This work is supported by the German Research Council (DFG) under HO 5606/1-1.

Equality interpolating theories [21, 4] define a general framework for interpolation in the combination of quantifier-free theories. This framework handles theory lemmas that propagate equalities between variables shared between theories, enabling Nelson-Oppen theory combination. The equalities may relate variables that come from different formulas in the interpolation problem. A theory is equality interpolating if it can find an interpolating term for these equalities that is expressed using only the symbols occuring in both parts of the interpolation problem.

The algorithm of Yorsh and Musuvathi [21] only supports convex theories and is not applicable to the theory of arrays. Bruttomesso et al. [4] extended the framework to non-convex theories. They also present a complete interpolation procedure for the quantifier-free theory of arrays that works for theory combination. However, their solver depends on the partioning of the interpolation problem and this can lead to exponential blow-up of the solving procedure. Our interpolation procedure works on a proof produced by a more efficient array solver that is independent of the partioning of the interpolation problem.

Totla and Wies [20] present an interpolation method for arrays based on complete instantiations. It combines the idea of [4] with local theory extension [19]. Given an interpolation problem A and B, they define two sets $\mathcal{K}[W(A,B)]$ and $\mathcal{K}[W(B,A)]$, each using only symbols from A resp. B, that contain the instantiations of the array axioms needed to prove unsatisfiabilty. Then an existing solver and interpolation procedure for uninterpreted functions can be used to compute the interpolant. The procedure produces a quadratic blow-up on the input formulas. We also found that their procedure fails for some extensionality lemmas, when we used it to create candidate interpolants.

The last two techniques require to know the partitions at solving time. Thus, when computing sequence interpolants or tree interpolants, they would require either an adapted interpolation procedure or the solver has to run multiple times. In contrast, our method can be easily extended to tree interpolation [7].

2 Notation

We assume standard first order logic. A theory $\mathcal{T}$ is given by a signature Σ and a set of axioms. The theory of arrays $\mathcal{T}_{\mathcal{A}}$ is parameterized by an index theory and an element theory. The signature $\Sigma_{\mathcal{A}}$ of $\mathcal{T}_{\mathcal{A}}$ contains the *select* (or *read*) function $\cdot[\cdot]$ and the *store* (or *write*) function $\cdot\langle\cdot \lhd \cdot\rangle$. In the following, a, b, s, t will denote array terms, i, j, k index terms and v, w element terms. For an array a, index i and element v, $a[i]$ returns the element stored in a at i, and $a\langle i \lhd v\rangle$ returns a copy of a where the element at index i is replaced by the element v, leaving a unchanged. The functions are defined by the following axioms proposed by McCarthy [16].

$$\forall a\ i\ v.\ a\langle i \lhd v\rangle[i] = v \qquad \text{(idx)}$$

$$\forall a\ i\ j\ v.\ i \neq j \rightarrow a\langle i \lhd v\rangle[j] = a[j] \qquad \text{(read-over-write)}$$

We consider the variant of the extensional theory of arrays proposed by Bruttomesso et al. [4] where the signature is extended by the function $\text{diff}(\cdot,\cdot)$ which for distinct arrays a and b returns an index where a and b differ, and an arbitrary index else. The extensionality axiom then becomes

$$\forall a\ b.\ a[\text{diff}(a,b)] = b[\text{diff}(a,b)] \rightarrow a = b\ . \qquad \text{(ext-diff)}$$

The authors of [4] have shown that the quantifier-free fragment of the theory of arrays with diff, $\mathcal{T}_{AxDiff}$, is closed under interpolation. To express the interpolants conveniently we use the notation from [20] for rewriting arrays. For $k \geq 0$ we define $a \overset{k}{\leadsto} b$ for two arrays a and b

inductively as

$$a \overset{0}{\rightsquigarrow} b := a \qquad a \overset{k+1}{\rightsquigarrow} b := a\langle \mathrm{diff}(a,b) \lhd b[\mathrm{diff}(a,b)]\rangle \overset{k}{\rightsquigarrow} b \ .$$

Thus, $a \overset{k}{\rightsquigarrow} b$ changes the values in a at k indices to the values stored in b. The equation $a \overset{k}{\rightsquigarrow} b = b$ holds if and only if a and b differ at at most k indices. The indices where they differ are the diff terms occuring in $a \overset{k}{\rightsquigarrow} b$.

An interpolation problem (A, B) is a pair of formulas where $A \wedge B$ is unsatisfiable. A *Craig interpolant* for (A, B) is a formula I such that (i) A implies I, (ii) I and B are unsatisfiable and (iii) I contains only symbols shared between A and B. Given an interpolation problem (A, B), the symbols shared between A and B are called *shared*, symbols only occuring in A are called *A-local* and symbols only occuring in B, *B-local*. A literal, e.g. $a = b$, that contains A-local and B-local symbols is called *mixed*.

3 Preliminaries

3.1 Proof Tree Preserving Interpolation

In this section we give a short overview of the proof tree preserving interpolation framework presented by Christ et al. [9]. This method defines two projections $\cdot \downharpoonleft A$ and $\cdot \downharpoonleft B$ that project a literal to its A-part resp. B-part. For a literal ℓ occurring only in the formula A, the projections are $\ell \downharpoonleft A \equiv \ell$ and $\ell \downharpoonleft B \equiv \top$, and similar for a literal occurring only in B. This projection can be naturally extended to conjunctions of literals. Then a *partial interpolant* of a clause C occurring in the proof tree is defined as the interpolant of $A \wedge (\neg C) \downharpoonleft A$ and $B \wedge (\neg C) \downharpoonleft B$. The paper shows that these partial interpolants can be computed inductively over the proof tree and the partial interpolant of the root is the interpolant of A and B. For a theory lemma C, a partial interpolant is computed from the interpolation problem $(\neg C \downharpoonleft A, \neg C \downharpoonleft B)$.

The core idea of proof tree preserving interpolation is a scheme to handle mixed equalities $a = b$ where a is A-local and b is B-local. For these a fresh variable x_{ab} is introduced and the projections are defined as follows.

$$(a = b) \downharpoonleft A :\equiv (a = x_{ab}) \qquad (a = b) \downharpoonleft B :\equiv (x_{ab} = b)$$

Thus, $a = b$ is equivalent to $\exists x_{ab}.(a = b) \downharpoonleft A \wedge (a = b) \downharpoonleft B$ and x_{ab} is a new shared variable that may occur in interpolants. For disequalities we follow [10] and use an auxiliary variable x_{ab} and a Boolean auxiliary variable $p_{x_{ab}}$. We define $\mathrm{EQ}(x, s) :\equiv p_x \,\mathsf{xor}\, x = s$ and define the projections for $a \neq b$ as

$$(a \neq b) \downharpoonleft A :\equiv \mathrm{EQ}(x_{ab}, a) \qquad (a \neq b) \downharpoonleft B :\equiv \neg\, \mathrm{EQ}(x_{ab}, b) \ .$$

For an interpolation problem $(A \wedge (\neg C) \downharpoonleft A, B \wedge (\neg C) \downharpoonleft B)$ where $\neg C$ contains $a \neq b$ we require as additional symbol condition that the interpolant has the form $I[\mathrm{EQ}(x_{ab}, s_1)] \dots [\mathrm{EQ}(x_{ab}, s_n)]$[1], where $s_1, \dots, s_n$ are shared terms and each EQ term occurs positively in I. For a resolution step on the pivot literal $a = b$ the following interpolation rule combines the partial interpolants of the input clauses to a partial interpolant of the resolvent.

$$\frac{C_1 \vee a = b : I_1[\mathrm{EQ}(x_{ab}, s_1)] \dots [\mathrm{EQ}(x_{ab}, s_n)] \qquad C_2 \vee a \neq b : I_2(x_{ab})}{C_1 \vee C_2 : I_1[I_2(s_1)] \dots [I_2(s_n)]}$$

[1] One can show that such an interpolant exists for every equality interpolating theory in the sense of Definition 4.1 in [4]. The terms $s_1, \dots, s_n$ are the terms $\underline{v}$ in that definition.

3.2 Weakly Equivalent Arrays

In this section, we revisit the definitions and results about weakly equivalent arrays that are used in the decision procedure for the theory of arrays presented by Christ and Hoenicke [6].

For a formula F, let V be the set of terms that contains the array terms in F and in addition the select terms $a[i]$ and their indices i and for every store term $a\langle i \lhd v\rangle$ in F the terms i, v, $a[i]$ and $a\langle i \lhd v\rangle[i]$. Let $\sim$ be the equivalence relation on V representing equality. The *weak equivalence graph* G^W is defined by its vertices, the array-valued terms in V, and its undirected edges of the form (i) $s_1 \leftrightarrow s_2$ if $s_1 \sim s_2$ and (ii) $s_1 \overset{i}{\leftrightarrow} s_2$ if s_1 has the form $s_2\langle i \lhd \cdot\rangle$ or vice versa. If two arrays a and b are connected in G^W by a path P they are called *weakly equivalent.* We write $a \overset{P}{\Leftrightarrow} b$. Weakly equivalent arrays can differ only at finitely many positions given by $\text{Stores}\,(P) := \{i \mid \exists s_1\ s_2.\ s_1 \overset{i}{\leftrightarrow} s_2 \in P\}$. Two arrays a and b are called *weakly equivalent on* i, denoted by $a \approx_i b$, if there exists a path P between them such that $k \not\sim i$ holds for every $k \in \text{Stores}\,(P)$. If a and b are weakly equivalent on i, they must store the same value at i. Two arrays a and b are called *weakly congruent on* i, $a \sim_i b$, if the equality $a'[j] = b'[k]$ holds for $j \sim k \sim i$ and $a' \approx_i a$, $b' \approx_i b$. Also in this case they must store the same value at i.

We use $\text{Cond}(a \overset{P}{\Leftrightarrow} b), \text{Cond}(a \approx_i b), \text{Cond}(a \sim_i b)$ to denote the conjunction of the literals $v = v'$ (resp. $v \neq v'$), $v, v' \in V$, such that $v \sim v'$ (resp. $v \not\sim v'$) is necessary to show the corresponding property. Instances of array lemmas are generated according to the following rules:

$$\frac{i \sim j \qquad a \approx_i b \qquad a[i], b[j] \in V}{i \neq j \vee \neg\,\text{Cond}(a \approx_i b) \vee a[i] = b[j]} \tag{read-over-weakeq}$$

$$\frac{a \overset{P}{\Leftrightarrow} b \qquad \forall i \in \text{Stores}\,(P)\,.\ a \sim_i b \qquad a, b \in V}{\neg\,\text{Cond}(a \overset{P}{\Leftrightarrow} b) \vee \bigvee_{i \in \text{Stores}(P)} \neg\,\text{Cond}(a \sim_i b) \vee a = b} \tag{weakeq-ext}$$

The first rule, based on (read-over-write), propagates equalities between select terms and the second, based on extensionality, propagates equalities on array terms. In the following, we will describe how we can derive partial interpolants for these array lemmas.

4 Interpolants for Read-Over-Weakeq Lemmas

In this section, we show how to interpolate lemmas generated by (read-over-weakeq). A lemma of this type (see Figure 1) explains a conflict of the form

$$i = j \wedge \text{Cond}(a \approx_i b) \wedge a[i] \neq b[j]\ \ .$$

The weak equivalence $a \approx_i b$ ensures that a and b are equal at $i = j$ which contradicts $a[i] \neq b[j]$.

If the index equality $i = j$ is mixed, the interpolation problem contains the shared auxiliary variable x_{ij} for $i = j$. If i (resp. j) is shared, we call i (resp. j) the shared term for $i = j$. We then identify four basic cases: (i) there exists a shared term for $i = j$ and $a[i] \neq b[j]$ is in B or mixed, (ii) there is a shared term for $i = j$ and $a[i] \neq b[j]$ is A-local, (iii) both i and j are B-local, and (iv) both i and j are A-local.

4.1 There is a Shared Term for $i = j$ and $a[i] \neq b[j]$ is in B or mixed

If there exists a shared term x for the index equality $i = j$, the interpolant I can contain terms $s[x]$ for shared array terms s occuring in the weak path between a and b. The basic idea is to summarize the weak A-paths by applying rule (read-over-weakeq) on their end terms.

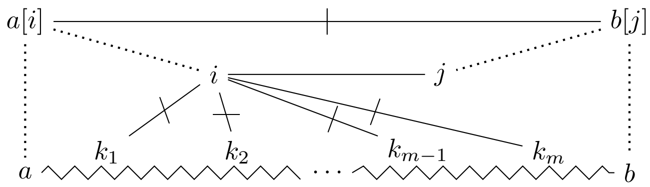

Figure 1: A read-over-weakeq conflict. Solid lines represent strong (dis-)equalities, dotted lines function-argument relations, and zigzag lines represent weak paths consisting of store steps and array equalities.

Example 1. Consider the following read-over-weakeq conflict:

$$i = j \wedge a = s_1 \wedge s_1\langle k_1 \lhd v_1\rangle = s_2 \wedge s_2\langle k_2 \lhd v_2\rangle = s_3 \wedge s_3 = b \wedge i \neq k_1 \wedge i \neq k_2 \wedge a[i] \neq b[j]$$

where a, k_2, v_2, i are A-local, b, k_1, v_1, j are B-local, and s_1, s_2, s_3 are shared. Projecting the mixed literals on A and B as described in Section 3.1 yields the interpolation problem

$$A : i = x_{ij} \wedge a = s_1 \wedge s_2\langle k_2 \lhd v_2\rangle = s_3 \wedge \mathrm{EQ}(x_{ik_1}, i) \wedge i \neq k_2 \wedge \mathrm{EQ}(x_{a[i]b[j]}, a[i])$$
$$B : x_{ij} = j \wedge s_1\langle k_1 \lhd v_1\rangle = s_2 \wedge s_3 = b \wedge \neg\,\mathrm{EQ}(x_{ik_1}, k_1) \wedge \neg\,\mathrm{EQ}(x_{a[i]b[j]}, b[j])\ \ .$$

An interpolant is

$$I \equiv \mathrm{EQ}(x_{a[i]b[j]}, s_1[x_{ij}]) \wedge s_2[x_{ij}] = s_3[x_{ij}] \wedge \mathrm{EQ}(x_{ik_1}, x_{ij})\ \ .$$

A implies I. As $a = s_1$ holds, we get $a[x_{ij}] = s_1[x_{ij}]$. With $\mathrm{EQ}(x_{a[i]b[j]}, a[i])$ and $i = x_{ij}$ follows $\mathrm{EQ}(x_{a[i]b[j]}, s_1[x_{ij}])$. The equality $s_2[x_{ij}] = s_3[x_{ij}]$ follows by applying (read-over-weakeq) on $s_3 = s_2\langle k_2 \lhd v_2\rangle$ and using $x_{ij} = i \neq k_2$. Finally, $\mathrm{EQ}(x_{ik_1}, i)$ and $i = x_{ij}$ yield $\mathrm{EQ}(x_{ik_1}, x_{ij})$.
B contradicts I. By $\neg\,\mathrm{EQ}(x_{ik_1}, k_1)$ and $\mathrm{EQ}(x_{ik_1}, x_{ij})$ we get $x_{ij} \neq k_1$. With $s_2 = s_1\langle k_1 \lhd v_1\rangle$ and (read-over-weakeq) we get $s_1[x_{ij}] = s_2[x_{ij}]$. From $s_3 = b$ we get $s_3[x_{ij}] = b[x_{ij}]$. Transitivity, $\mathrm{EQ}(x_{a[i]b[j]}, s_1[x_{ij}])$ and $s_2[x_{ij}] = s_3[x_{ij}]$ and the above select equalities yield $\mathrm{EQ}(x_{a[i]b[j]}, b[x_{ij}])$. With $x_{ij} = j$, we get a contradiction to $\neg\,\mathrm{EQ}(x_{a[i]b[j]}, b[j])$.
Symbol condition for I. I contains only shared symbols and auxiliary variables. All auxiliary variables introduced for disequalities appear in positive EQ terms.

Algorithm. Assume that the weak path $P : a \approx_i b$ is subdivided into A- and B-paths where shared paths are added to B-paths. Let x be the shared term for $i = j$, i.e. x stands for i if i is shared, for j if i is not shared but j is, and for the auxiliary variable x_{ij} if $i = j$ is mixed.
(i) An inner A-path π of P starts and ends with a shared term: $\pi : s_1 \approx_i s_2$ (these shared array terms can also be auxiliary variables introduced for a mixed array equality). The summary is $s_1[x] = s_2[x]$. For every B-local index disequality $i \neq k$ on π add the disjunct $x = k$ and for every mixed index disequality add the disjunct $\mathrm{EQ}(x_{ik}, x)$. The interpolant of the subpath is

$$I_\pi \equiv s_1[x] = s_2[x] \vee F^A_\pi(x) \qquad \text{where } F^A_\pi(x) :\equiv \bigvee_{\substack{k\in\mathrm{Stores}(\pi)\\ i\neq k\ B\text{-local}}} x = k \ \vee \bigvee_{\substack{k\in\mathrm{Stores}(\pi)\\ i\neq k\ \text{mixed}}} \mathrm{EQ}(x_{ik}, x)\ \ .$$

(ii) If $a[i] \neq b[j]$ is mixed and $a[i]$ is A-local, the first A-path on P starts with a or a is shared, i.e. $\pi : a \approx_i s_1$ (where s_1 can be a). For the path π, build the term $\mathrm{EQ}(x_{a[i]b[j]}, s_1[x])$ and add $F^A_\pi(x)$ as in case (i).

$$I_\pi \equiv \mathrm{EQ}(x_{a[i]b[j]}, s_1[x]) \vee F^A_\pi(x)$$

(iii) Similarly in the case where $a[i] \neq b[j]$ is mixed and $b[j]$ is A-local, the last A-path on P ends with b or b is shared, $\pi : s_n \approx_i b$. In this case the disjunct $i \neq j$ needs to be added if $i = j$ is B-local and i, j are both shared.

$$I_\pi \equiv \mathrm{EQ}(x_{a[i]b[j]}, s_n[x]) \vee F^A_\pi(x) \; [\vee\, i \neq j]$$

(iv) For every B-path π, add the conjunct $x \neq k$ for each A-local index disequality $i \neq k$, and the conjunct $\mathrm{EQ}(x_{ik}, x)$ for each mixed index disequality $i \neq k$ on π. We define

$$I_\pi \equiv F^B_\pi(x) \qquad \text{where } F^B_\pi(x) :\equiv \bigwedge_{\substack{k \in \mathrm{Stores}(\pi)\\ i \neq k \;A\text{-local}}} x \neq k \;\wedge \bigwedge_{\substack{k \in \mathrm{Stores}(\pi)\\ i \neq k \text{ mixed}}} \mathrm{EQ}(x_{ik}, x) \;.$$

The lemma interpolant is the conjunction of the above path interpolants. If i, j are shared and $i = j$ is A-local, add the conjunct $i = j$.

$$I \equiv \bigwedge_{\pi \in A\text{-paths}} I_\pi \quad \wedge \bigwedge_{\pi \in B\text{-paths}} I_\pi \quad [\wedge\, i = j] \;.$$

4.2 There is a Shared Term for $i = j$ and $a[i] \neq b[j]$ is A-local

If there exists a shared index for $i = j$ and $a[i] \neq b[j]$ is A-local, we build disequalities for the B-paths instead of equalities for the A-paths. This corresponds roughly to obtaining the interpolant of the inverse problem (B, A) by Section 4.1 and negating the resulting formula. Only the EQ terms are not negated because of the asymmetry of the projection.

Using the definitions of F^A_π and F^B_π from the previous section, the lemma interpolant is

$$I \equiv \bigvee_{(\pi : s_1 \approx_i s_2) \in B\text{-paths}} (s_1[x] \neq s_2[x] \wedge F^B_\pi(x)) \quad \vee \bigvee_{\pi \in A\text{-paths}} F^A_\pi(x) \quad [\vee\, i \neq j] \;.$$

4.3 Both i and j are B-local

When both i and j are B-local, we have no term x representing the weak path index i in the interpolant I. Hence, summarizing A- (resp. B-) paths by terms of the form $s_1[x] = s_2[x] \vee F^A_\pi$ (resp. $s_1[x] \neq s_2[x] \wedge F^B_\pi$) as above is not possible. Instead we use the diff function to make statements about the desired index. For instance, if $a = b\langle i \lhd v\rangle\langle j \lhd w\rangle$ for arrays a, b with $b[j] \neq w$, then $a \overset{2}{\rightsquigarrow} b = b$ and $\mathrm{diff}(a, b) = j$ or $\mathrm{diff}(a \overset{1}{\rightsquigarrow} b, b) = j$ hold.

Example 2. Consider the following conflict:

$$i = j \wedge a = s_1 \wedge s_1\langle k \lhd v\rangle = s_2 \wedge s_2 = b \wedge i \neq k \wedge a[i] \neq b[j]$$

where a, b, i, j are B-local, k, v are A-local, and s_1, s_2 are shared. Splitting the mixed disequality $i \neq k$ as described in Section 3.1 results in the interpolation problem

$$\begin{aligned} &A : s_1\langle k \lhd v\rangle = s_2 \wedge \mathrm{EQ}(x_{ik}, k)\\ &B : i = j \wedge a = s_1 \wedge s_2 = b \wedge \neg\,\mathrm{EQ}(x_{ik}, i) \wedge a[i] \neq b[j] \;. \end{aligned}$$

An interpolant should reflect the information that s_1 and s_2 can differ at most at one index satisfying the EQ term. Using diff, we can express the interpolant

$$I : (s_1 = s_2 \vee \mathrm{EQ}(x_{ik}, \mathrm{diff}(s_1, s_2))) \wedge s_1 \overset{1}{\rightsquigarrow} s_2 = s_2 \;.$$

A implies I. The first literal in A implies that s_1 and s_2 differ at most at index k, yielding $s_1 \overset{1}{\rightsquigarrow} s_2 = s_2$. If $s_1 \neq s_2$, then $k = \text{diff}(s_1, s_2)$ and $\text{EQ}(x_{ik}, \text{diff}(s_1, s_2))$ follows from $\text{EQ}(x_{ik}, k)$.
B contradicts I. B implies that $s_1[i] \neq s_2[i]$ holds. Thus, $s_1 = s_2$ cannot hold and I states that $\text{EQ}(x_{ik}, \text{diff}(s_1, s_2))$ holds. With $\neg\text{EQ}(x_{ik}, i)$ in B, this implies $\text{diff}(s_1, s_2) \neq i$. Hence, s_1 and s_2 must differ at least at two indices. This contradicts $s_1 \overset{1}{\rightsquigarrow} s_2 = s_2$ in I.
Symbol condition for I. I contains only shared symbols and the auxiliary variable for $i \neq k$ appears in a positive EQ term only.

To generalize this idea, we define inductively for arrays a and b, a number $m \geq 0$ and a formula $F(\cdot)$ with one free parameter:

$$\begin{aligned} \text{weq}(a, b, 0, F(\cdot)) &:\equiv \quad a = b \\ \text{weq}(a, b, m+1, F(\cdot)) &:\equiv \quad (a = b \lor F(\text{diff}(a, b))) \;\land\; \text{weq}(a \overset{1}{\rightsquigarrow} b, b, m, F(\cdot)) \;. \end{aligned} \tag{weq}$$

The term $\text{weq}(a, b, m, F(\cdot))$ states that arrays a and b differ at most at m indices and that each index i on which they differ satisfies the formula $F(i)$.

Algorithm. For any A-path $\pi : s_1 \approx_i s_2$, we count the number of stores $|\pi| := |\,\text{Stores}\,(\pi)\,|$. Each index i where s_1 and s_2 differ needs to satisfy $F^A_\pi(i)$ as defined in Section 4.1. There is nothing to do for B-paths. The lemma interpolant is

$$I \equiv \bigwedge_{(\pi : s_1 \approx_i s_2) \in \text{A-paths}} \text{weq}(s_1, s_2, |\pi|, F^A_\pi(\cdot)) \;.$$

4.4 Both i and j are A-local

In the case where both i and j are A-local, we summarize the B-paths, as all B-path ends are shared terms. Analogously to weq, we define for arrays a, b, a number $m \geq 0$ and a formula F:

$$\begin{aligned} \text{nweq}(a, b, 0, F(\cdot)) &:\equiv \quad a \neq b \\ \text{nweq}(a, b, m+1, F(\cdot)) &:\equiv \quad (a \neq b \land F(\text{diff}(a, b))) \;\lor\; \text{nweq}(a \overset{1}{\rightsquigarrow} b, b, m, F(\cdot)) \;. \end{aligned} \tag{nweq}$$

The term $\text{nweq}(a, b, m, F(\cdot))$ expresses that either one of the first m indices i found by stepwise rewriting a to b satisfies the formula $F(i)$, or a and b differ at more than m indices.

Analogously to the last section, the lemma interpolant can be expressed as

$$I \equiv \bigvee_{(\pi : s_1 \approx_i s_2) \in \text{B-paths}} \text{nweq}(s_1, s_2, |\pi|, F^B_\pi(\cdot)) \;.$$

Note that this is almost the negation of the interpolant of (B, A) computed as in Section 4.3. Only the EQ terms are not negated because of the asymmetry of the projection.

5 Interpolants for Weakeq-Ext Lemmas

In this section, we describe how to compute interpolants for array lemmas of type (weakeq-ext). Figure 2 displays a visualization of the corresponding conflict

$$\text{Cond}(a \overset{P}{\Leftrightarrow} b) \land \bigwedge_{i \in \text{Stores}(P)} \text{Cond}(a \sim_i b) \land a \neq b \;.$$

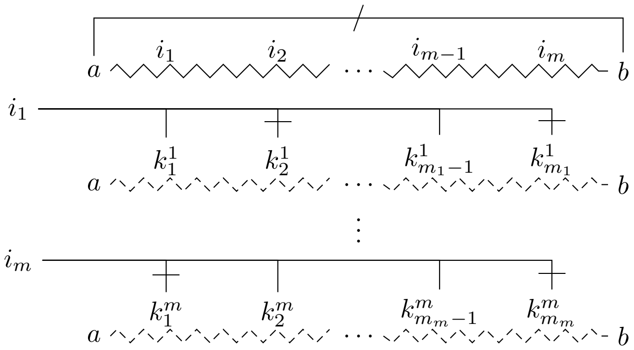

Figure 2: Visualization of a weakeq-ext lemma. Solid lines represent strong (dis-)equalities, zigzag lines store paths and dashed zigzag lines weak paths which can contain select edges.

The main path P shows that a and b differ at most at the indices in Stores (P) and $a \sim_i b$ shows that a and b do not differ at index i.

A weak path labelled with i (short: i-path) represents weak congruence on i, i.e. it can contain select edges $a' \overset{j\ \ k}{\leftarrow\!\!-\!\!\rightarrow} b'$ where $i \sim j$ and $i \sim k$ and $a'[j] \sim b'[k]$. We modify the methods in Section 4 to summarize the i-paths. In the B-local case 4.3, B-local select edges make no difference, as the weq terms are built over A-paths, and analogously for the A-local case 4.4. However, if there are A-local select edges in the B-local case or vice versa, then k is shared or the index equality $i = k$ is mixed and we can use k or the auxiliary variable x_{ik} to proceed as in the shared cases 4.1 and 4.2.

We have to adapt the interpolation procedures in Sections 4.1 and 4.2 by adding the index equalities that pertain to a select edge, analogously to the index disequality for a store edge before. More specifically, we add to $F^A_\pi(x)$ a disjunct $x \neq k$ for each B-local $i = k$ on an A-path, and $x \neq x_{ik}$ for each mixed $i = k$. Here, x is the shared term for the main index equality $i = j$ as before. For B-paths we add to $F^B_\pi(x)$ the conjunct $x = k$ for each A-local $i = k$ and $x = x_{ik}$ for each mixed $i = k$. Furthermore, if there is a mixed select equality $a'[j] = b'[k]$ on the weak path, the auxiliary variable $x_{a'[j]b'[k]}$ is used in the summary for the A-path instead of $s[x]$, i.e., we get a term of the form $x_{a'[j]b'[k]} = s_2[x]$ in 4.1, and analogously for 4.2.

For (weakeq-ext) lemmas, we distinguish three cases: (i) $a \neq b$ is in B, (ii) $a \neq b$ is A-local, or (iii) $a \neq b$ is mixed.

5.1 $a \neq b$ is in B

If the disequality $a \neq b$ is in B, the A-paths both on the main store path and on the weak paths have only shared path ends. Hence, we summarize A-paths similarly to Sections 4.1 and 4.3.

Algorithm. Divide the main path $a \overset{P}{\Leftrightarrow} b$ into A-paths and B-paths. For each $i \in$ Stores (P) on a B-path, summarize the corresponding i-path as in Sections 4.1 or 4.3. The resulting summary is denoted by I_i. For an A-path $s_1 \overset{\pi}{\Leftrightarrow} s_2$ use a weq term to state that each index where s_1 and s_2 differ satisfies $I_i(\cdot)$ for some $i \in$ Stores (π) where I_i is computed as in 4.1 with the shared term $\cdot$ for $i = j$. The lemma interpolant is

$$I \equiv \bigwedge_{\substack{i \in \text{Stores}(\pi) \\ \pi \in B\text{-paths}}} I_i \quad \wedge \bigwedge_{(s_1 \overset{\pi}{\Leftrightarrow} s_2) \in A\text{-paths}} \text{weq}(s_1, s_2, |\pi|, \bigvee_{i \in \text{Stores}(\pi)} I_i(\cdot)) \ .$$

5.2 $a \neq b$ is A-local

The case where $a \neq b$ is A-local is similar with the roles of A and B swapped. For each $i \in \text{Stores}\,(\pi)$ on an A-path π on P, interpolate the corresponding weak path as in Sections 4.2 or 4.4 and obtain I_i. For each $i \in \text{Stores}\,(\pi)$ on a B-path π on P, interpolate the corresponding weak path as in Section 4.2 using $\cdot$ as shared term and obtain $I_i(\cdot)$. The lemma interpolant is

$$I \equiv \bigvee_{\substack{i\in\text{Stores}(\pi)\\ \pi\in A\text{-paths}}} I_i \quad \vee \bigvee_{(s_1 \overset{\pi}{\Leftrightarrow} s_2)\in B\text{-paths}} \text{nweq}(s_1, s_2, |\pi|, \bigwedge_{i\in\text{Stores}(\pi)} I_i(\cdot)) \enspace .$$

5.3 $a \neq b$ is mixed

If $a \neq b$ is mixed, where w.l.o.g. a is A-local, the outer A- and B-paths end with A-local or B-local terms respectively. The auxiliary variable x_{ab} may not be used in store or select terms, thus we first need to find a shared term representing a before we can summarize A-paths.

Example 3. Consider the following conflict:

$$\begin{aligned} a = s\langle i_1 \lhd v_1\rangle \wedge b = s\langle i_2 \lhd v_2\rangle \wedge a \neq b \qquad & \text{(main path)}\\ \wedge\, a[i_1] = s_1[i_1] \wedge b = s_1\langle k_1 \lhd w_1\rangle \wedge i_1 \neq k_1 \qquad & (i_1\text{-path})\\ \wedge\, a = s_2\langle k_2 \lhd w_2\rangle \wedge i_2 \neq k_2 \wedge b[i_2] = s_2[i_2] \qquad & (i_2\text{-path}) \end{aligned}$$

where a, i_1, v_1, k_2, w_2 are A-local, b, i_2, v_2, k_1, w_1 are B-local and s, s_1, s_2 are shared. An interpolant for the conflict is

$$\begin{aligned} I \equiv\, & \text{EQ}(x_{ab}, s) \wedge \text{weq}(s, s_2, 1, \text{EQ}(x_{i_2k_2}, \cdot))\\ & \vee \text{nweq}\Big(s, s_1, 2, \text{EQ}(x_{ab}, s\langle \cdot \lhd s_1[\cdot]\rangle) \wedge \text{weq}(s\langle \cdot \lhd s_1[\cdot]\rangle, s_2, 1, \text{EQ}(x_{i_2k_2}, :)) \wedge \text{EQ}(x_{i_1k_1}, \cdot)\Big) \enspace , \end{aligned}$$

where in the second line the symbol $\cdot$ refers to the diff term of the outer nweq term and the symbol $:$ to the diff term of the inner weq term.

A implies I. Because of $a = s\langle i_1 \lhd v_1\rangle$, the arrays a and s can differ only at i_1. If $a = s$, we get $\text{EQ}(x_{ab}, s)$ by replacing a in the A-projection of $a \neq b$ and we get $\text{weq}(s, s_2, 1, \text{EQ}(x_{i_2k_2}, \cdot))$ from the i_2-path as in Section 4.3. Otherwise, $s[i_1] \neq s_1[i_1]$, as $a[i_1] = s_1[i_1]$ holds in A by the i_1-path. By correcting s at i_1 with $s_1[i_1]$, we get a, and therefore $\text{EQ}(x_{ab}, s\langle i_1 \lhd s_1[i_1]\rangle)$ holds. Again, we get $\text{weq}(s\langle i_1 \lhd s_1[i_1]\rangle, s_2, 1, \text{EQ}(x_{i_2k_2}, :))$ as in Section 4.3. Finally, for i_1, we have the literal $\text{EQ}(x_{i_1k_1}, i_1)$ by projection. We know that i_1 is among the diff terms between s and s_1. Thus, the nweq term holds since i_1 satisfies the nested formula.

I contradicts B. Assume that the first disjunct of I holds. By $\text{EQ}(x_{ab}, s)$ in I and $\neg\,\text{EQ}(x_{ab}, b)$ in B, we get $s \neq b$. The only potential difference is at i_2 because of B. Since B contains $s_2[i_2] = b[i_2]$ this implies that s and s_2 also differ at i_2. However, with the weq term in I and $\neg\,\text{EQ}(x_{i_2k_2}, i_2)$ in B, we get a contradiction. Assume now that the second disjunct of I holds. Then either s and s_1 must differ at some index $\cdot$ which satisfies the formula inside the nweq term, or at more than 2 positions. We know from B that s and s_1 can only differ at i_2 and k_1. For k_1, there is the term $\neg\,\text{EQ}(x_{i_1k_1}, k_1)$ in B. Hence, $\cdot$ can only be i_2. Because of $\text{EQ}(x_{ab}, s\langle \cdot \lhd s_1[\cdot]\rangle)$ in I and $\neg\,\text{EQ}(x_{ab}, b)$ in B, we get $s\langle \cdot \lhd s_1[\cdot]\rangle \neq b$. Since s and b only differ at i_2, the difference can only be at $i_2 = \cdot$ and $s\langle \cdot \lhd s_1[\cdot]\rangle[i_2] \neq b[i_2] = s_2[i_2]$. By the inner weq term in I, i_2 has to satisfy $\text{EQ}(x_{i_2k_2}, i_2)$, which is a contradiction to B.

Algorithm. Identify in the main path P the first A-path $a \overset{\pi_0}{\Leftrightarrow} s_1$ and its store indices $\text{Stores}(\pi_0) = \{i_1, \ldots i_{|\pi_0|}\}$. To build an interpolant, we rewrite s_1 by storing at each index i_m the value $a[i_m]$. We use $\tilde{s}$ to denote the intermediate arrays. We build a formula $I_m(\tilde{s})$ inductively over $m \le |\pi_0|$. This formula is an interpolant if $\tilde{s}$ is a shared array that differs from a only at the indices $i_1, \ldots, i_m$.

For $m = 0$, i.e., $a = \tilde{s}$, we modify the lemma by adding the strong edge $\tilde{s} \leftrightarrow a$ in front of all paths and summarize it as if it was B-local using the algorithm in Section 5.1, but drop the weq term for the path $\tilde{s} \leftrightarrow a \overset{\pi_0}{\Leftrightarrow} s_1$. This yields $I_{5.1}(\tilde{s})$. We define

$$I_0(\tilde{s}) \equiv \text{EQ}(x_{ab}, \tilde{s}) \wedge I_{5.1}(\tilde{s}) \ .$$

For the induction step to $m+1$ we assume that $\tilde{s}$ only differs from a at $i_1, \ldots, i_m, i_{m+1}$. Our goal is to find a shared index term x for i_{m+1} and a shared value v for $a[x]$. We use the i_{m+1}-path to conclude that $\tilde{s}\langle x \lhd v\rangle$ is equal to a at i_{m+1}. Then we can include $I_m(\tilde{s}\langle x \lhd v\rangle)$ computed using the induction hypothesis.

(i) If there is a select edge on a B-subpath of the i_{m+1}-path or if i_{m+1} is itself shared, we immediately get a shared term x for i_{m+1}. If the last B-path π^{m+1} on the i_{m+1}-path starts with a mixed select equality, then the corresponding auxiliary variable is the shared value v. Otherwise, π^{m+1} starts with a shared array s^{m+1} and $v := s^{m+1}[x]$. We summarize the i_{m+1}-path from a to the start of π^{m+1} as in Section 4.2 and get $I_{4.2}(x)$. Finally, we set

$$I_{m+1}(\tilde{s}) \equiv I_{4.2}(x) \vee (I_m(\tilde{s}\langle x \lhd v\rangle) \wedge F^B_{\pi^{m+1}}(x)) \ .$$

(ii) Otherwise, we split the i_{m+1}-path into $a \sim_{i_{m+1}} s^{m+1}$ and $s^{m+1} \overset{\pi^{m+1}}{\Leftrightarrow} b$, where π^{m+1} is the last B-subpath of the i_{m+1}-path. If s_1 and a are equal on i_{m+1} then also $\tilde{s}$ and a are equal and the interpolant is simply $I_m(\tilde{s})$. If a and s^{m+1} differ on i_{m+1}, we build an interpolant from $a \sim_{i_{m+1}} s^{m+1}$ as in 4.4 and obtain $I_{4.4}$. Otherwise, s_1 and s^{m+1} differ on i_{m+1}. We build the store path $s_1 \overset{P'}{\Leftrightarrow} s^{m+1}$ by concatenating P and π^{m+1}. Using nweq on the subpaths $s \overset{\pi}{\Leftrightarrow} s'$ of P' we find the shared term x for i_{m+1}. If π is in A we need to add the conjunct $s \overset{|\pi|}{\rightsquigarrow} s' = s'$ to obtain an interpolant. We get

$$\begin{aligned} I_{m+1}(\tilde{s}) \equiv I_m(\tilde{s}) \ \vee\ I_{4.4} \quad [\text{for } a \sim_{i_{m+1}} s^{m+1}] \\ \vee \bigvee_{s \overset{\pi}{\Leftrightarrow} s' \text{ in } P'} \text{nweq}\left(s, s', |\pi|, I_m(\tilde{s}\langle \cdot \lhd s^{m+1}[\cdot]\rangle) \wedge F^B_{\pi^{m+1}}(\cdot)\right) [\wedge\, s \overset{|\pi|}{\rightsquigarrow} s' = s'] \ . \end{aligned}$$

The lemma interpolant for the mixed extensionality lemma is $I \equiv I_{|\pi_0|}(s_1)$.

6 Conclusion and Future Work

We presented an interpolation algorithm for the quantifier-free fragment of the theory of arrays. The algorithm uses an efficient array solver based on weak equivalence on arrays. In contrast to most existing interpolation algorithms for arrays, the solver does not depend on the partioning of the interpolation problem. Thus, our technique allows for efficient interpolation especially in the context of sequence interpolants and tree interpolants where interpolants for different partitions of the same unsatisfiable formula need to be computed.

Because we use the framework of proof tree preserving interpolation we only need to provide algorithms to interpolate the lemmas. These algorithms build the formulas by simply iterating over the weak equivalence and weak congruence paths found by the array solver.

Implementation of the algorithm in SMTInterpol [8] is ongoing work. The algorithm for read-over-weakeq lemmas is already implemented and supports sequence and tree interpolation.

References

[1] Pavel Andrianov, Karlheinz Friedberger, Mikhail U. Mandrykin, Vadim S. Mutilin, and Anton Volkov. CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions - (competition contribution). In *TACAS (2)*, volume 10206 of *Lecture Notes in Computer Science*, pages 355–359, 2017.

[2] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker blast. *STTT*, 9(5-6):505–525, 2007.

[3] Angelo Brillout, Daniel Kroening, Philipp Rümmer, and Thomas Wahl. Program verification via craig interpolation for presburger arithmetic with arrays. In *VERIFY@IJCAR*, volume 3 of *EPiC Series in Computing*, pages 31–46. EasyChair, 2010.

[4] Roberto Bruttomesso, Silvio Ghilardi, and Silvio Ranise. Quantifier-free interpolation in combinations of equality interpolating theories. *ACM Trans. Comput. Log.*, 15(1):5:1–5:34, 2014.

[5] Franck Cassez, Anthony M. Sloane, Matthew Roberts, Matthew Pigram, Pongsak Suvanpong, and Pablo González de Aledo Marugán. Skink: Static analysis of programs in LLVM intermediate representation - (competition contribution). In *TACAS (2)*, volume 10206 of *Lecture Notes in Computer Science*, pages 380–384, 2017.

[6] Jürgen Christ and Jochen Hoenicke. Weakly equivalent arrays. In *FroCos*, volume 9322 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2015.

[7] Jürgen Christ and Jochen Hoenicke. Proof tree preserving tree interpolation. *J. Autom. Reasoning*, 57(1):67–95, 2016.

[8] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Smtinterpol: An interpolating SMT solver. In *SPIN*, volume 7385 of *Lecture Notes in Computer Science*, pages 248–254. Springer, 2012.

[9] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2013.

[10] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. *CoRR*, abs/1705.05309, 2017. Improved and simplified version. http://arxiv.org/abs/1705.05309.

[11] Matthias Dangl, Stefan Löwe, and Philipp Wendler. Cpachecker with support for recursive programs and floating-point arithmetic - (competition contribution). In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 423–425. Springer, 2015.

[12] Marius Greitschus, Daniel Dietsch, Matthias Heizmann, Alexander Nutz, Claus Schätzle, Christian Schilling, Frank Schüssele, and Andreas Podelski. Ultimate taipan: Trace abstraction and abstract interpretation - (competition contribution). In *TACAS (2)*, volume 10206 of *Lecture Notes in Computer Science*, pages 399–403, 2017.

[13] Matthias Heizmann, Yu-Wen Chen, Daniel Dietsch, Marius Greitschus, Alexander Nutz, Betim Musa, Claus Schätzle, Christian Schilling, Frank Schüssele, and Andreas Podelski. Ultimate automizer with an on-demand construction of floyd-hoare automata - (competition contribution). In *TACAS (2)*, volume 10206 of *Lecture Notes in Computer Science*, pages 394–398, 2017.

[14] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

[15] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. Lazy abstraction. In *POPL*, pages 58–70. ACM, 2002.

[16] John McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.

[17] Kenneth L. McMillan. Lazy abstraction with interpolants. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

[18] Alexander Nutz, Daniel Dietsch, Mostafa Mahmoud Mohamed, and Andreas Podelski. ULTIMATE KOJAK with memory safety checks - (competition contribution). In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 458–460. Springer, 2015.

[19] Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, volume

3632 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2005.

[20] Nishant Totla and Thomas Wies. Complete instantiation-based interpolation. *J. Autom. Reasoning*, 57(1):37–65, 2016.

[21] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.