Efficient Interpolation for the Theory of Arrays

Jochen Hoenicke and Tanja Schindler*

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 are inefficient for computing sequence and tree interpolants: the solver needs to run for every partitioning (A, B) of the interpolation problem to avoid creating AB-mixed terms. 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. The computed interpolants have worst-case exponential size for extensionality lemmas and worst-case quadratic size otherwise. We show that these bounds are strict in the sense that there are lemmas with no smaller interpolants. We implemented the algorithm and show that the produced interpolants are useful to prove memory safety for C programs.

1 Introduction

Several model-checkers [1,2,8,14,16,17,20,25,26] 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 reuses an existing unsatisfiability proof to compute interpolants efficiently.

Our method is based on the array solver presented in [10], which fits well into existing Nelson-Oppen frameworks. The solver generates lemmas, valid in the theory of arrays, that explain equalities between terms shared between different theories. The terms 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 [13], which produces interpolants from existing proofs that can contain propagated equalities between symbols from different parts of the interpolation problem.

The contribution of this paper is an algorithm to interpolate the lemmas produced by the solver of the theory of arrays without introducing quantifiers. The solver only generates two types of lemmas, namely a variant of the read-over-write axiom and a variant of the extensionality axiom. However, the lemmas

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

contain array store chains of arbitrary length which need to be handled by the interpolation procedure. The interpolants our algorithm produces summarize array store chains, e. g., they state that two shared arrays at the end of a subchain differ at most at m indices, each satisfying a subformula. Bruttomesso et al. [6] showed that adding a diff function to the theory of arrays makes the quantifier-free fragment closed under interpolation, i.e. it ensures the existence of quantifier-free interpolants for quantifier-free problems. We use the diff function to obtain the indices for store chains and give a more efficient algorithm that exploits the special shape of the lemmas provided by the solver.

Nevertheless, the lemma interpolants produced by our algorithm may be exponential in size (with respect to the size of the input lemma). We show that this is unavoidable as there are lemmas that have no small interpolants.

Related Work. The idea of computing interpolants from resolution proofs goes back to Krajíček and Pudlák [22,27]. McMillan [24] extended their work to SMT with a single theory. The theory of arrays can be added by including quantified axioms and can be interpolated using, e.g., the method by Christ and Hoenicke [9] for quantifier instantiation, or the method of Bonacina and Johansson [4] for superposition calculus. Brillout et al [5] apply a similar algorithm to compute interpolants from sequent calculus proofs. In contrast to our approach, using such a procedure generates quantified interpolants.

Equality interpolating theories [30,7] allow for the generation of quantifier-free interpolants in the combination of quantifier-free theories. A theory is equality interpolating if it can express an interpolating term for each equality using only the symbols occurring in both parts of the interpolation problem. The algorithm of Yorsh and Musuvathi [30] only supports convex theories and is not applicable to the theory of arrays. Bruttomesso et al. [7] 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 in [6]. However, their solver depends on the partitioning of the interpolation problem. 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 partitioning of the interpolation problem.

Totla and Wies [29] present an interpolation method for arrays based on complete instantiations. It combines the idea of [7] with local theory extension [28]. Given an interpolation problem A and B, they define two sets, each using only symbols from A resp. B, that contain the instantiations of the array axioms needed to prove unsatisfiability. Then an existing solver and interpolation procedure for uninterpreted functions can be used to compute the interpolant. The procedure causes 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. We give an example for this in Sect. 6.

The last two techniques require to know the partitioning at solving time. Thus, when computing sequence [24] or tree interpolants [19], they would require either an adapted interpolation procedure or the solver has to run multiple times. In contrast, our method can easily be extended to tree interpolation [11].

2 Basic Definitions

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. Its signature $\Sigma_\mathcal{A}$ 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 denote array terms, i, j, k index terms and v, w element terms. For 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 [23].

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

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

We consider the variant of the extensional theory of arrays proposed by Bruttomesso et al. [6] where the signature is extended by the function $\mathrm{diff}(\cdot,\cdot)$. For distinct arrays a and b, it returns an index where a and b differ, and an arbitrary index otherwise. The extensionality axiom then becomes

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

The authors of [6] 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 [29] for rewriting arrays. For $m \geq 0$, we define $a \overset{m}{\rightsquigarrow} b$ for two arrays a and b inductively as

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

Thus, $a \overset{m}{\rightsquigarrow} b$ changes the values in a at m indices to the values stored in b. The equality $a \overset{m}{\rightsquigarrow} b = b$ holds if and only if a and b differ at up to m indices. The indices where they differ are the diff terms occurring in $a \overset{m}{\rightsquigarrow} b$.

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 in the theory $\mathcal{T}$, (ii) I and B are $\mathcal{T}$-unsatisfiable and (iii) all non-theory symbols occurring in I are shared between A and B. Given an interpolation problem (A, B), the symbols shared between A and B are called *shared*, symbols only occurring in A are called *A-local* and symbols only occurring in B, *B-local*. A literal, e.g. $a = b$, that contains A-local and B-local symbols is called *mixed*.

3 Preliminaries

Our interpolation procedure operates on theory lemmas instantiated from particular variants of the read-over-write and extensionality axioms, and is designed to be used within the proof tree preserving interpolation framework. In the following, we give a short overview of this method and revisit the definitions and results about weakly equivalent arrays.

3.1 Proof Tree Preserving Interpolation

The proof tree preserving interpolation scheme presented by Christ et al. [13] allows to compute interpolants for an unsatisfiable formula using a resolution proof that is unaware of the interpolation problem.

For a partitioning (A, B) of the interpolation problem, two projections $\cdot \lfloor A$ and $\cdot \lfloor B$ project a literal to its A-part resp. B-part. For a literal ℓ occurring in A, we define $\ell \lfloor A \equiv \ell$. If ℓ is A-local, $\ell \lfloor B \equiv \mathbf{true}$. For ℓ in B, the projections are defined analogously. These projections are canonically extended to conjunctions of literals. A *partial interpolant* of a clause C occurring in the proof tree is defined as the interpolant of $A \wedge (\neg C) \lfloor A$ and $B \wedge (\neg C) \lfloor B$. 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 for the interpolation problem $((\neg C) \lfloor A, (\neg C) \lfloor B)$.

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

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

Thus, $a = b$ is equivalent to $\exists x_{ab}.(a = b) \lfloor A \wedge (a = b) \lfloor B$ and x_{ab} is a new shared variable that may occur in partial interpolants. For disequalities we introduce an uninterpreted predicate EQ and define the projections for $a \neq b$ as

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

For an interpolation problem $(A \wedge (\neg C) \lfloor A, B \wedge (\neg C) \lfloor B)$ where $\neg C$ contains $a \neq b$, we require as additional symbol condition that x_{ab} only occurs as first parameter of an EQ predicate which occurs positively in the interpolant, i.e., the interpolant has the form $I[\mathrm{EQ}(x_{ab}, s_1)] \ldots [\mathrm{EQ}(x_{ab}, s_n)]$[1]. For a resolution step on the mixed pivot literal $a = b$, the following rule combines the partial interpolants of the input clauses to a partial interpolant of the resolvent.

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

3.2 Weakly Equivalent Arrays

Proof tree preserving interpolation can handle mixed literals, but it cannot deal with mixed *terms* which can be produced when instantiating (read-over-write) on an A-local store term and a B-local index. The lemmas produced in the decision procedure for the theory of arrays presented by Christ and Hoenicke [10] avoid such mixed terms by exploiting *weak equivalences* between arrays.

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 each store

[1] One can show that such an interpolant exists for every equality interpolating theory in the sense of Definition 4.1 in [7]. The terms s_i are the terms $\underline{\mathrm{v}}$ in that definition.

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*. This is denoted by $a \overset{P}{\Leftrightarrow} b$. 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 they are connected by a path P such that $k \not\sim i$ holds for each $k \in \text{Stores}\,(P)$. Two arrays a and b are called *weakly congruent on* i, $a \sim_i b$, if they are weakly equivalent on i, or if there exist $a'[j], b'[k] \in V$ with $a'[j] \sim b'[k]$ and $j \sim k \sim i$ and $a' \approx_i a$, $b' \approx_i b$. If a and b are weakly congruent on i, they must store the same value at i. For example, if $a\langle i+1 \lhd v\rangle \sim b$ and $b[i] \sim c[i]$, arrays a and b are weakly equivalent on i while a and c are only weakly congruent on 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:

$$\frac{a \approx_i b \qquad i \sim j \qquad a[i], b[j] \in V}{\text{Cond}(a \approx_i b) \wedge i = j \rightarrow a[i] = b[j]} \tag{roweq}$$

$$\frac{a \overset{P}{\Leftrightarrow} b \qquad \forall i \in \text{Stores}\,(P)\,.\ a \sim_i b \qquad a, b \in V}{\text{Cond}(a \overset{P}{\Leftrightarrow} b) \wedge \bigwedge_{i \in \text{Stores}(P)} \text{Cond}(a \sim_i b) \rightarrow a = b} \tag{weq-ext}$$

The first rule, based on (read-over-write), propagates equalities between select terms and the second, based on extensionality, propagates equalities on array terms. These rules are complete for the quantifier-free theory of arrays [10]. In the following, we describe how to derive partial interpolants for these lemmas.

4 Interpolants for Read-Over-Weakeq Lemmas

A lemma generated by (roweq) explains the conflict (negation of the lemma)

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

The weak equivalence $a \approx_i b$ ensures that a and b are equal at $i = j$ which contradicts $a[i] \neq b[j]$ (see Fig. 1).

The general idea for computing an interpolant for this conflict, similar to [15], is to summarize maximal paths induced by literals of the same part (A or B), relying on the fact that the terms at the ends of these paths are shared. If a shared term is equal to the index i, we can express that the shared arrays at the path ends coincide or must differ at the index. There is a *shared term for* $i = j$ if i or j are shared or if $i = j$ is mixed. If there is no shared term for $i = j$, the

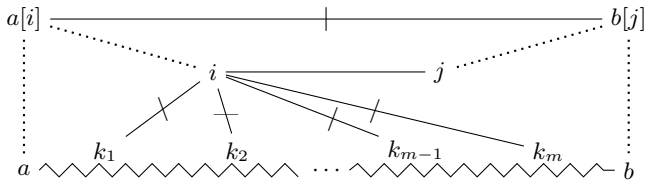

Fig. 1. A read-over-weakeq conflict. Solid lines represent strong (dis-)equalities, dotted lines function-argument relations, and zigzag lines represent weak paths consisting of store steps and array equalities.

interpolant can be expressed using diff chains to capture the index. We identify four basic cases: (i) there is a shared term for $i = j$ and $a[i] = b[j]$ is in B or mixed, (ii) there is a shared term for $i = j$ and $a[i] = b[j]$ is A-local, (iii) both i and j are B-local, and (iv) both i and j are A-local.

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

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

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

$$\begin{aligned} &a = s_1 \wedge s_1\langle k_1 \lhd v_1\rangle = s_2 \wedge s_2\langle k_2 \lhd v_2\rangle = s_3 \wedge s_3 = b \\ \wedge\, &i \neq k_1 \wedge i \neq k_2 \wedge i = j \wedge a[i] \neq b[j] \end{aligned}$$

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 Sect. 3.1 yields the interpolation problem

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

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

Algorithm. The first step is to subdivide the weak path $P : a \approx_i b$ into A- and B-paths. An equality edge $\leftrightarrow$ is assigned to either an A- or B-path depending on whether the corresponding equality is in A or B. A mixed equality $a' = b'$ is split into the A-local equality $a' = x_{a'b'}$ and the B-local equality $x_{a'b'} = b'$. Store edges $\overset{i}{\leftrightarrow}$ are assigned depending on which part contains the store term. If an equality or store term is shared between both parts, the algorithm can assign it to A or B arbitrarily. The whole path from a to b is then an alternation of A- and B-paths, which meet at shared boundary terms.

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 $\pi : s_1 \approx_i s_2$ of P starts and ends with a shared term. The summary is $s_1[x] = s_2[x]$. For a store edge on π with index k, add the disjunct $x = k$ if the corresponding disequality $i \neq k$ is B-local, and the disjunct $\mathrm{EQ}(x_{ik}, x)$ if the disequality is mixed. 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

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

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

Lemma 1. *If x is a shared term for $i = j$ and $a[i] = b[j]$ is in B or mixed, a partial interpolant of the lemma* $\mathrm{Cond}(a \approx_i b) \wedge i = j \rightarrow a[i] = b[j]$ *is*

$$I \equiv \bigwedge_{\pi \in A\text{-paths}} I_\pi \quad \wedge \quad \bigwedge_{\pi \in B\text{-paths}} F^B_\pi(x) \quad [\wedge i = j] \ .$$

Proof. The interpolant only contains the shared boundary arrays, the shared term x for $i = j$, auxiliary variables for mixed disequalities under an EQ predicate, and shared store indices k where the store term is in a different part than the corresponding index disequality.

$\neg C \downharpoonright A$ implies I: For a B-path π, we show that $F^B_\pi(x)$ follows from the A-part. If i is B-local, there are no A-local or mixed index disequalities and $F^B_\pi(x)$ holds trivially. Otherwise $i = x$ follows from A, since either i is shared and x is i, $i = j$ is A-local and x is j, or $i = x$ is the A-projection of the mixed equality $i = j$. Then $F^B_\pi(x)$ follows by replacing i by x in A-local disequalities and A-projections of mixed disequalities on π. For an A-path π, if $F^A_\pi(x)$ does

not hold, we get $s_1[x] = s_2[x]$ by applying rule (roweq). Note that $x \neq k$ follows from $i = x$ if $i \neq k$ is A-local, and from $\mathrm{EQ}(x_{ik}, k)$ and $\neg F^A_\pi(x)$ in the mixed case. For the outer A-path in case (ii), $a[x] = s_1[x]$ is combined with the A-projection of the mixed disequality $a[i] \neq b[j]$ using $i = x$, which yields the EQ term. Analogously we get the EQ term for (iii), but to derive $j = x$ in the case where both i and j are shared but $i = j$ is B-local, we need to exclude $i \neq j$.

$\neg C \lfloor B \wedge I$ is unsat: Again if i is in B then $i = x$ follows from B by the choice of x. For a B-path π, we can conclude $s_1[x] = s_2[x]$ by applying rule (roweq) and using the index disequalities in $\neg C \lfloor B$ and $F^B_\pi(x)$. For an A-path π, $s_1[x] = s_2[x]$ (or, in cases (ii) and (iii), $\mathrm{EQ}(x_{a[i]b[j]}, s[x])$) follows from I_π using the B-local index disequalities and $i = x$ to show that $F^A_\pi(x)$ cannot hold. Transitivity and the B-projection of $a[i] \neq b[j]$ lead to a contradiction. If $i = j$ is A-local, i is the shared term, and $b[j]$ is in B, the conjunct $i = j$ in I is needed here. □

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

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

Lemma 2. *Using the definitions of F^A_π and F^B_π from the previous section, if x is a shared term for $i = j$ and $a[i] = b[j]$ is A-local, then a partial interpolant of the lemma* $\mathrm{Cond}(a \approx_i b) \wedge i = j \to a[i] = b[j]$ *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 (or both A-local), we may not find a shared term for the index where a and b should be equal. Instead we use the diff function to express all indices where a and b differ. For instance, if $a = b\langle i \lhd v\rangle\langle j \lhd w\rangle$ for arrays a, b with $a[j] \neq b[j]$, then $\mathrm{diff}(a, b) = j$ or $\mathrm{diff}(a \overset{1}{\rightsquigarrow} b, b) = j$ hold.

Example 2. Consider the following conflict:

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

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 Sect. 3.1 yields the interpolation problem

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

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

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

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

$$\begin{aligned} \mathrm{weq}(a, b, 0, F(\cdot)) &\equiv a = b \\ \mathrm{weq}(a, b, m+1, F(\cdot)) &\equiv (a = b \vee F(\mathrm{diff}(a, b))) \wedge \mathrm{weq}(a \overset{1}{\rightsquigarrow} b, b, m, F(\cdot)) \ . \end{aligned}$$

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

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

Lemma 3. *A partial interpolant of the lemma* $\mathrm{Cond}(a \approx_i b) \wedge i = j \rightarrow a[i] = b[j]$ *with B-local i and j is*

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

Proof. The symbol condition holds by the same argument as in Lemma 1.

$\neg C \downharpoonright A$ implies I: Let $\pi : s_1 \approx_i s_2$ be an A-path on P. The path π shows that s_1 and s_2 can differ at most at $|\pi|$ indices, hence $s_1 \overset{|\pi|}{\rightsquigarrow} s_2 = s_2$ follows from $\neg C \downharpoonright A$. If $s_1 \overset{m}{\rightsquigarrow} s_2 \neq s_2$ holds for $m < |\pi|$, then $\mathrm{diff}(s_1 \overset{m}{\rightsquigarrow} s_2, s_2) = k$ for some $k \in \mathrm{Stores}\,(\pi)$. If $i \neq k$ is A-local, then $k = k$ holds trivially, if $i \neq k$ is mixed, then $\mathrm{EQ}(x_{ik}, k)$ is part of $\neg C \downharpoonright A$. Hence, $s_1 \overset{m}{\rightsquigarrow} s_2 = s_2 \vee F^A_\pi(\mathrm{diff}(s_1 \overset{m}{\rightsquigarrow} s_2, s_2))$ holds for all $m < |\pi|$. This shows $\mathrm{weq}(s_1, s_2, |\pi|, F^A_\pi(\cdot))$.

$\neg C \downharpoonright B \wedge I$ is unsat: For every B-path $\pi : s_1 \approx_i s_2$ on P, we get $s_1[i] = s_2[i]$ with (roweq). For every A-path $\pi : s_1 \approx_i s_2$, I implies that s_1 and s_2 differ at finitely many indices which all satisfy $F^A_\pi(\cdot)$. The disequalities and B-projections in B imply that i does not satisfy $F^A_\pi(i)$, and therefore $s_1[i] = s_2[i]$. Then $a[i] = b[i]$ holds by transitivity, in contradiction to $a[i] \neq b[j]$ and $i = j$ in B. □

4.4 Both i and j are A-local

The interpolant is dual to the previous case and we define the dual of weq for arrays a, b, a number $m \geq 0$ and a formula F:

$$\begin{aligned} \mathrm{nweq}(a, b, 0, F(\cdot)) &\equiv a \neq b \\ \mathrm{nweq}(a, b, m+1, F(\cdot)) &\equiv (a \neq b \wedge F(\mathrm{diff}(a, b))) \vee \mathrm{nweq}(a \overset{1}{\rightsquigarrow} b, b, m, F(\cdot)) \ . \end{aligned}$$

The formula $\mathrm{nweq}(a, b, m, F(\cdot))$ 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. Like in Sect. 4.2, the lemma interpolant is dual to the one computed in Sect. 4.3.

Lemma 4. *A partial interpolant of the lemma* $\mathrm{Cond}(a \approx_i b) \wedge i = j \rightarrow a[i] = b[j]$ *with A-local i and j is* $I \equiv \bigvee_{(\pi : s_1 \approx_i s_2) \in B\text{-}paths} \mathrm{nweq}(s_1, s_2, |\pi|, F^B_\pi(\cdot))$.

Theorem 1. *For all instantiations of the rule* (roweq)*, quantifier-free interpolants can be computed as described in Sects. 4.1–4.4.*

5 Interpolants for Weakeq-Ext Lemmas

A conflict corresponding to a lemma of type (weq-ext) is of the form

$$\mathrm{Cond}(a \overset{P}{\Leftrightarrow} b) \wedge \bigwedge_{i \in \mathrm{Stores}(P)} \mathrm{Cond}(a \sim_i b) \wedge a \neq b \ .$$

The main path P shows that a and b differ at most at the indices in $\mathrm{Stores}\,(P)$, and $a \sim_i b$ (called i-path as of now) shows that a and b do not differ at index i.

To compute an interpolant, we summarize the main path by weq (or nweq) terms to capture the indices where a and b can differ, and include summaries for the i-paths that are similar to the interpolants in Sect. 4. The i-paths can contain a select edge $a' \overset{k_1, k_2}{\leftarrow\!\!\rightarrow} b'$ where $a'[k_1] \sim b'[k_2]$, $i \sim k_1$, and $i \sim k_2$. In the B-local case 4.3, B-local select edges make no difference for the construction, as the weq formulas are built over A-paths, and analogously for the A-local case 4.4. However, if there are A-local select terms $a'[k]$ 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} and proceed as in the cases where there is a shared term.

We have to adapt the interpolation procedures in Sects. 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. 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 i-path index i. For B-paths we add to $F^B_\pi(x)$ a conjunct $x = k$ for each A-local $i = k$ and $x = x_{ik}$ for each mixed $i = k$. Moreover, if there is a mixed select equality $a'[k_1] = b'[k_2]$ on the i-path, the auxiliary variable $x_{a'[k_1]b'[k_2]}$ is used in the summary for the subpath instead of $s[x]$, i.e., we get a term of the form $s_1[x] = x_{a'[k_1]b'[k_2]}$ in 4.1, and analogously for 4.2.

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

5.1 $a = b$ is in B

If the literal $a = b$ 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 Sects. 4.1 and 4.3.

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

Lemma 5. *The lemma* $\mathrm{Cond}(a \overset{P}{\Leftrightarrow} b) \wedge \bigwedge_{i \in \mathrm{Stores}(P)} \mathrm{Cond}(a \sim_i b) \rightarrow a = b$ *where* $a = b$ *is in* B *has the partial interpolant*

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

Proof. The path summaries I_i fulfill the symbol conditions, and the boundary terms s_1, s_2 used in the weq formulas are guaranteed to be shared.

$\neg C \restriction A$ implies I: By Sects. 4.1 and 4.3, $\mathrm{Cond}(a \sim_i b) \restriction A$ implies I_i for $i \in \mathrm{Stores}(\pi)$ where π is a B-path on P. For an A-path $s_1 \overset{\pi}{\Leftrightarrow} s_2$ on P, we know that s_1 and s_2 differ at most at $|\pi|$ positions, namely at the indices $i \in \mathrm{Stores}(\pi)$. Each index satisfies the corresponding I_i by Sect. 4.1. Hence, $\mathrm{weq}(s_1, s_2, |\pi|, \bigvee_{i \in \mathrm{Stores}(\pi)} I_i(\cdot)[\wedge\, i = \cdot])$ holds.

$\neg C \restriction B \wedge I$ is unsat: We first note that if a and b differ at some index i, there must be an A-path or a B-path $s_1 \overset{\pi}{\Leftrightarrow} s_2$ on the main path, such that s_1 and s_2 also differ at index i. We show that no such index exists. For a B-path $s_1 \overset{\pi}{\Leftrightarrow} s_2$, s_1 and s_2 can only differ at $i \in \mathrm{Stores}(\pi)$. But for every $i \in \mathrm{Stores}(\pi)$, we get $a[i] = b[i]$ from I_i as in Lemma 1 resp. 3. For an A-path $s_1 \overset{\pi}{\Leftrightarrow} s_2$, the interpolant contains $\mathrm{weq}(s_1, s_2, |\pi|, \bigvee_{i \in \mathrm{Stores}(\pi)}(I_i(\cdot)[\wedge\, i = \cdot]))$. Thus, if s_1 and s_2 differ at some index i', the interpolant implies $I_i(i')$ for some index $i \in \mathrm{Stores}(\pi)$ and additionally $i = i'$ if i is shared. Together with $\mathrm{Cond}(a \sim_i b) \restriction B$ this implies $a[i'] = b[i']$ as in the proof of Lemma 1. This shows that there is no index where a and b differ, but this contradicts $a \neq b$ in $\neg C \restriction B$. □

5.2 $a = b$ is A-local

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

Lemma 6. *The lemma* $\mathrm{Cond}(a \overset{P}{\Leftrightarrow} b) \wedge \bigwedge_{i \in \mathrm{Stores}(P)} \mathrm{Cond}(a \sim_i b) \rightarrow a = b$ *where* $a = b$ *is* A*-local has the partial interpolant*

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

5.3 $a = b$ is mixed

If $a = b$ 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:

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

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.

Our algorithm below computes the following interpolant for the conflict.

$$I \equiv I_0(s) \vee \mathrm{nweq}\left(s, s_1, 2, I_0(s\langle \cdot \lhd s_1[\cdot]\rangle) \wedge \mathrm{EQ}(x_{i_1k_1}, \cdot)\right)$$
$$\text{where } I_0(\tilde{s}) = \mathrm{EQ}(x_{ab}, \tilde{s}) \wedge \mathrm{weq}(\tilde{s}, s_2, 1, \mathrm{EQ}(x_{i_2k_2}, \cdot))$$

Algorithm. Identify in the main path P the first A-path $a \overset{\pi_0}{\Leftrightarrow} s_1$ and its store indices $\mathrm{Stores}(\pi_0) = \{i_1, \dots i_{|\pi_0|}\}$. 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 $m \leq |\pi_0|$. This formula is an interpolant if $\tilde{s}$ is a shared array that differs from a only at the indices $i_1, \dots, i_m$.

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 using the algorithm in Sect. 5.1, but drop the weq formula for the path $\tilde{s} \leftrightarrow a \overset{\pi_0}{\Leftrightarrow} s_1$. This yields $I_{5.1}(\tilde{s})$. We define

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

For the induction step we assume that $\tilde{s}$ only differs from a at $i_1, \dots, i_m, i_{m+1}$. 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 Sect. 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 at 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 at 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 at 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

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

Lemma 7. *The lemma* $\mathrm{Cond}(a \overset{P}{\Leftrightarrow} b) \wedge \bigwedge_{i \in \mathrm{Stores}(P)} \mathrm{Cond}(a \sim_i b) \rightarrow a = b$ *where* $a = b$ *is mixed has the partial interpolant* $I \equiv I_{|\pi_0|}(s_1)$.

A proof by induction over the length of the path π_0 can be found in [21].

Theorem 2. *Sects. 5.1–5.3 give interpolants for all cases of the rule* (weq-ext).

6 Complexity

Expanding the definition of an array rewrite term $a \overset{k}{\rightsquigarrow} b$ naïvely already yields a term exponential in k. This is avoided by using let expressions for common subterms. With this optimization the interpolants for read-over-weakeq lemmas are quadratic in the worst case. The interpolants of Sects. 4.1 and 4.2 contain at most one literal for every literal in the lemma, so the interpolant is linear in the size of the lemma. The interpolants of Sects. 4.3 and 4.4 are quadratic, since expanding the definition of weq will copy the formula $F^A_\pi(\cdot)$ resp. $F^B_\pi(\cdot)$, for each local store edge and instantiate it with a different shared term.

Example 4. The following interpolation problem has only quadratic interpolants.

$$\begin{aligned}
&A : b = a\langle i_1 \lhd v_1\rangle \cdots \langle i_n \lhd v_n\rangle \wedge p_1(i_1) \wedge \cdots \wedge p_n(i_n)\\
&B : a[j] \neq b[j] \wedge \neg p_1(j) \wedge \ldots \neg p_n(j)\\
&I \equiv \mathsf{let}\, a_0 = a\, \mathsf{let}\, d_1 = \mathrm{diff}(a_0, b)\, \mathsf{let}\, a_1 = a_0\langle d_1 \lhd b[d_1]\rangle\\
&\quad \ldots \mathsf{let}\, d_n = \mathrm{diff}(a_{n-1}, b)\, \mathsf{let}\, a_n = a_{n-1}\langle d_n \lhd b[d_n]\rangle\\
&\quad (p_1(d_1) \vee \cdots \vee p_n(d_1) \vee a_0 = b) \wedge \cdots\\
&\quad (p_1(d_n) \vee \cdots \vee p_n(d_n) \vee a_{n-1} = b) \wedge a_n = b
\end{aligned}$$

There is no interpolant that is not quadratic in n. The interpolant has to imply that $p_k(i_k)$ is true for every k. There are no shared index-valued terms in the lemma. Hence, the only way to express the i_k values using shared terms is by applying the diff operator on a and b and constructing diff chains as in the interpolant I. The diff operator returns one of the $i_1, \ldots, i_n$ in every step, but it is not determined which one. Consequently, every combination $p_k(d_l)$ is needed.

The algorithms in Sects. 5.1 and 5.2 produce a worst-case quadratic interpolant as they nest the linear interpolants of 4.1 and 4.2 in a weq resp. nweq formula, which expands this term a linear number of times. However, the algorithm in 5.3 is worst-case exponential in the size of the extensionality lemma.

The following example explains why this bound is strict. This example also shows that the method of Totla and Wies [29] is not complete. In particular, for $n = 1$ their preprocessing algorithm produces a satisfiable formula from the original interpolation problem.

Example 5. The following interpolation problem of size $O(n^2)$ has only interpolants of exponential size in n.

$$\begin{aligned}
A : a = s\langle i_1^A \lhd v_1^A\rangle \cdots \langle i_n^A \lhd v_n^A\rangle \wedge p(a) \wedge\\
\bigwedge_{j=1}^{n} p_j(i_j^A) \;\wedge\; \bigwedge_{j=1}^{n} a[i_j^A] = s_j[i_j^A] \wedge\\
\bigwedge_{j=1}^{n} \bigwedge_{l=0, l\neq j}^{n} q_j(i_l^A) \;\wedge\; \bigwedge_{j=1}^{n} t_j = a\langle i_0^A \lhd w_{j0}^A\rangle \ldots \cancel{\langle i_j^A \lhd w_{jj}^A\rangle} \ldots \langle i_n^A \lhd w_{jn}^A\rangle
\end{aligned}$$

$$B : b = s\langle i_1^B \lhd v_1^B\rangle \cdots \langle i_n^B \lhd v_n^B\rangle \wedge \neg p(b) \wedge$$
$$\bigwedge_{j=1}^{n} \bigwedge_{l=0, l\neq j}^{n} \neg p_j(i_l^B) \;\wedge\; \bigwedge_{j=1}^{n} s_j = b\langle i_0^B \lhd w_{j0}^B\rangle \ldots \cancel{\langle i_j^B \lhd w_{jj}^B\rangle} \ldots \langle i_n^B \lhd w_{jn}^B\rangle \wedge$$
$$\bigwedge_{j=1}^{n} \neg q_j(i_j^B) \;\wedge\; \bigwedge_{j=1}^{n} b[i_j^B] = t_j[i_j^B]$$

The first line of A and the first line of B ensure that there is a store-chain from a over s to b of length $2n$ and $p(a)$ and $\neg p(b)$ are used to derive the contradiction from the extensionality axiom. To prove that a and b are equal, the formulas show that they are equal at the indices i_j^A, $j = 1, \ldots, n$ (second line of A and B). Here p_j is used to ensure that i_j^A is distinct from all i_l^B, $l \neq j$. Analogously the last line of A and B shows that a and b are equal at the indices i_j^B, $j = 1, \ldots, n$.

Since $p(a) \wedge \neg p(b)$ is essential to prove unsatisfiability, the interpolant needs to contain the term $p(\cdot)$ for some shared array term that is equal to a and b. This can only be expressed by store terms of size n, e. g., $p(s\langle i_1 \lhd \cdot\rangle \cdots \langle i_n \lhd \cdot\rangle)$ (alternatively some store term starting on s_j or t_j can be used). As in the previous example, the store indices i_j can only be expressed using diff chains between shared arrays. For each index there is only one shared array that is guaranteed to contain the right value. The diff function returns the indices in arbitrary order. Therefore, the interpolant needs a case for every combination of diff term and value, as it is done by the interpolant computed in Section 5.3. This means the interpolant contains exponentially many $p(\cdot)$ terms.

7 Evaluation

We implemented the presented algorithms into SMTInterpol [12], an SMT solver computing sequence and tree interpolants. Our implementation verifies at run-time that the returned interpolants are correct. To evaluate the interpolation algorithm we used the Ultimate Automizer software model-checker [17] on the memory safety track of the SV-COMP 2018 [2] benchmarks. This track was chosen because Ultimate uses arrays to model memory access. We ran our experiments using the open-source benchmarking software `benchexec` [3] on a machine with a 3.4 GHz Intel i7-4770 CPU and set a 900 s time and a 6 GB memory limit. As comparison, we ran Ultimate with Z3 [3] and SMTInterpol without array interpolation using Ultimate's built-in theory-independent interpolation scheme based on unsatisfiable cores and predicate transformers [18].

Table 1 shows the result. From the 326 benchmarks we removed 50 benchmarks which Ultimate could not parse. The unknown results come from nonlinear arithmetic (SMTInterpol), quantifiers (due to incomplete elimination in the setting SMTInterpol-NoArrayInterpol), or incomplete interpolation engine (Z3). Our new algorithm solves 12.6 % more problems, and both helps to verify safety and guide the counterexample generation for unsafe benchmarks.

[2] `https://sv-comp.sosy-lab.org/2018/`

[3] `https://github.com/Z3Prover/z3` in version 4.6.0 (2abc759d0)

Table 1. Evaluation of Ultimate Automizer on the SV-COMP benchmarks for memsafety running with our new interpolation engine, without array interpolation, and Z3.

Setting	Tasks	Safe	Unsafe	Timeout	Unknown
SMTInterpol-ArrayInterpol	276	101	96	66	13
SMTInterpol-NoArrayInterpol	276	92	83	75	26
Z3	276	32	44	13	187

8 Conclusion

We presented an interpolation algorithm for the quantifier-free fragment of the theory of arrays. Due to the technique of proof tree preserving interpolation, our algorithm also works for the combination with other theories. Our algorithm operates on lemmas produced by an efficient array solver based on weak equivalence on arrays. The interpolants are built by simply iterating over the weak equivalence and weak congruence paths found by the solver. We showed that the complexity bound on the size of the produced interpolants is optimal.

In contrast to most existing interpolation algorithms for arrays, the solver does not depend on the partitioning of the interpolation problem. Thus, our technique allows for efficient interpolation especially when several interpolants for different partitionings of the same unsatisfiable formula need to be computed. Although it remains to prove formally that the algorithm produces tree interpolants, during the evaluation all returned tree interpolants were correct.

Acknowledgement. We would like to thank Daniel Dietsch for running the experiments.

References

1. P. Andrianov, K. Friedberger, M. Mandrykin, V. Mutilin, and A. Volkov. CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions. In *TACAS (2)*, pages 355–359, 2017.
2. D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast. *STTT*, 9(5-6):505–525, 2007.
3. D. Beyer, S. Löwe, and P. Wendler. Benchmarking and resource measurement. In *SPIN*, pages 160–178, 2015.
4. M. Bonacina and M. Johansson. On interpolation in automated theorem proving. *J. Autom. Reasoning*, 54(1):69–97, 2015.
5. A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Program verification via Craig interpolation for Presburger arithmetic with arrays. In *VERIFY@IJCAR*, pages 31–46. EasyChair, 2010.
6. R. Bruttomesso, S. Ghilardi, and S. Ranise. Quantifier-free interpolation of a theory of arrays. *Logical Methods in Computer Science*, 8(2), 2012.
7. R. Bruttomesso, S. Ghilardi, and S. Ranise. Quantifier-free interpolation in combinations of equality interpolating theories. *ACM Trans. Comput. Log.*, 15(1):5:1–5:34, 2014.

8. F. Cassez, A. Sloane, M. Roberts, M. Pigram, P. Suvanpong, and P. González de Aledo. Skink: Static analysis of programs in LLVM intermediate representation. In *TACAS (2)*, pages 380–384, 2017.
9. J. Christ and J. Hoenicke. Instantiation-based interpolation for quantified formulae. In *Decision Procedures in Software, Hardware and Bioware*, volume 10161 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl, Germany, 2010.
10. J. Christ and J. Hoenicke. Weakly equivalent arrays. In *FroCos*, pages 119–134. Springer, 2015.
11. J. Christ and J. Hoenicke. Proof tree preserving tree interpolation. *J. Autom. Reasoning*, 57(1):67–95, 2016.
12. J. Christ, J. Hoenicke, and A. Nutz. SMTInterpol: An interpolating SMT solver. In *SPIN*, pages 248–254. Springer, 2012.
13. J. Christ, J. Hoenicke, and A. Nutz. Proof tree preserving interpolation. In *TACAS*, pages 124–138. Springer, 2013.
14. M. Dangl, S. Löwe, and P. Wendler. CPAchecker with support for recursive programs and floating-point arithmetic. In *TACAS*, pages 423–425. Springer, 2015.
15. A. Fuchs, A. Goel, J. Grundy, S. Krstic, and C. Tinelli. Ground interpolation for the theory of equality. *Logical Methods in Computer Science*, 8(1), 2012.
16. M. Greitschus, D. Dietsch, M. Heizmann, A. Nutz, C. Schätzle, C. Schilling, F. Schüssele, and A. Podelski. Ultimate Taipan: Trace abstraction and abstract interpretation. In *TACAS (2)*, pages 399–403, 2017.
17. M. Heizmann, Y. Chen, D. Dietsch, M. Greitschus, A. Nutz, B. Musa, C. Schätzle, C. Schilling, F. Schüssele, and A. Podelski. Ultimate Automizer with an on-demand construction of Floyd-Hoare automata. In *TACAS (2)*, pages 394–398, 2017.
18. M. Heizmann, D. Dietsch, J. Leike, B. Musa, and A. Podelski. Ultimate automizer with array interpolation. In *TACAS*, pages 455–457, 2015.
19. M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
20. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL*, pages 58–70. ACM, 2002.
21. J. Hoenicke and T. Schindler. Efficient interpolation for the theory of arrays. *CoRR*, abs/1804.07173, 2018.
22. J. Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.
23. J. McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.
24. K. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
25. K. McMillan. Lazy abstraction with interpolants. In *CAV*, pages 123–136. Springer, 2006.
26. A. Nutz, D. Dietsch, M. Mohamed, and A. Podelski. Ultimate Kojak with memory safety checks. In *TACAS*, pages 458–460. Springer, 2015.
27. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
28. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, pages 219–234. Springer, 2005.
29. N. Totla and T. Wies. Complete instantiation-based interpolation. *J. Autom. Reasoning*, 57(1):37–65, 2016.
30. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *CADE*, pages 353–368. Springer, 2005.