Logical Methods in Computer Science
Vol. 8 (2:04) 2012, pp. 1–39
www.lmcs-online.org

Submitted Oct. 26, 2011
Published Apr. 27, 2012

QUANTIFIER-FREE INTERPOLATION OF A THEORY OF ARRAYS

ROBERTO BRUTTOMESSO [a], SILVIO GHILARDI [b], AND SILVIO RANISE [c]

[a] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano (Italy)
e-mail address: bruttomesso@dsi.unimi.it

[b] Dipartimento di Matematica, Università degli Studi di Milano (Italy)
e-mail address: ghilardi@dsi.unimi.it

[c] FBK-Irst, Trento (Italy)
e-mail address: ranise@fbk.eu

Abstract. The use of interpolants in model checking is becoming an enabling technology to allow fast and robust verification of hardware and software. The application of encodings based on the theory of arrays, however, is limited by the impossibility of deriving quantifier-free interpolants in general.

In this paper, we show that it is possible to obtain quantifier-free interpolants for a Skolemized version of the extensional theory of arrays. We prove this in two ways:

(1) non-constructively, by using the model theoretic notion of amalgamation, which is known to be equivalent to admit quantifier-free interpolation for universal theories; and
(2) constructively, by designing an interpolating procedure, based on solving equations between array updates. (Interestingly, rewriting techniques are used in the key steps of the solver and its proof of correctness.)

To the best of our knowledge, this is the first successful attempt of computing quantifier-free interpolants for a variant of the theory of arrays with extensionality.

1998 ACM Subject Classification: F.4.1, F.3.1.
Key words and phrases: Amalgamation, Modular constraints, Interpolating metarules.
This paper is a substantially extended version of [17].

Contents

1. Introduction

Craig's interpolation theorem [24] applies to first order logic formulæ and states that whenever the sequent $A \wedge B \Rightarrow \bot$ is valid, then it is possible to derive a formula I such that (i) $A \Rightarrow I$ is valid , (ii) $I \wedge B \Rightarrow \bot$ is valid, and (iii) I is defined over the common symbols of A and B.[1] After the seminal work of McMillan (see, e.g., [48]), Craig's interpolation has become an important technique in verification. Intuitively, the interpolant I can be seen as an over-approximation of A with respect to B. This observation is crucial for several applications of interpolation in verification. For example, the importance of computing *quantifier-free* interpolants (as several symbolic verification procedures represent sets of states and transitions as quantifier-free formulae) to over-approximate the set of reachable states for model checking has been observed. Unfortunately, Craig's interpolation theorem does not guarantee that it is always possible to compute quantifier-free interpolants. Even worse, for certain first-order theories, it is known that quantifiers must occur in interpolants of quantifier-free formulae [37]. As a consequence, several papers [11, 21, 22, 34, 37, 40, 42, 46, 50, 52, 54, 56] focused on the efficient computation of quantifier-free interpolants for first-order theories which are relevant for verification such as uninterpreted functions, (fragments of) Presburger arithmetic, theories of some data-structures, and their combination. Despite the ongoing efforts, so far, only the negative result in [37] is available for the computation of interpolants in the theory of arrays with extensionality, axiomatized by the following three sentences:

$$\begin{aligned}
&\forall y, i, e. rd(wr(y, i, e), i) = e \\
&\forall y, i, j, e. i \neq j \Rightarrow rd(wr(y, i, e), j) = rd(y, j) \\
&\forall x, y. x \neq y \Rightarrow (\exists i.\ rd(x, i) \neq rd(y, i))
\end{aligned}$$

where rd and wr are the usual operations for reading or updating arrays, respectively. For instance, there is no quantifier-free interpolant for the pair of quantifier-free formulae

$$\begin{aligned}
&A \equiv x = wr(y, i, e) \\
&B \equiv rd(x, j) \neq rd(y, j) \wedge rd(x, k) \neq rd(y, k) \wedge j \neq k.
\end{aligned}$$

This theory is important for both hardware and software verification, and a procedure for computing quantifier-free interpolants "*would extend the utility of interpolant extraction as a tool in the verifier's toolkit*" [48]. Indeed, the endeavour of designing such a procedure would be bound to fail (according to [37]) if we restrict ourselves to the original theory. To circumvent the problem, we add the (binary) function `diff` to rd and wr. Intuitively, $\texttt{diff}(a, b)$ is an index at which the elements stored in the arrays a and b are different ($\texttt{diff}(a, b)$ is defined arbitrarily in case a and b coincide). Formally, this is characterized by Skolemizing the third axiom above (also called the extensionality axiom) to obtain

$$\forall x, y. x \neq y \Rightarrow rd(x, \texttt{diff}(x, y)) \neq rd(y, \texttt{diff}(x, y))).$$

This axiom is sufficient to ensure that the theory of arrays with `diff` admits quantifier-free interpolants for quantifier-free formulae or, equivalently, that the quantifier-free fragment of the theory is closed under interpolation. For example, a quantifier-free interpolant for A

[1] To be precise, the original formulation of [24] is slightly different, and it states that whenever $A \Rightarrow B$ is valid, then it is possible to derive an I such that $A \Rightarrow I \Rightarrow B$ are valid, and I is over the common symbols of A and B. Clearly, the two formulations are equivalent.

and B above is

$$I \equiv x = wr(y, \mathtt{diff}(x, y), rd(x, \mathtt{diff}(x, y))).$$

Notice how `diff` permits to represent indexes in the quantifier-free interpolant I by mentioning only the array constants a and b that are common to A and B. As we will see in the rest of the paper, this is crucial to compute quantifier-free interpolants. One may wonder how useful it is to be able to compute quantifier-free interpolants in the Skolemized variant of the theory of arrays with extensionality considered here. The answer lies in the observation that this variant is sufficient whenever there is a need to check the unsatisfiability of formulae as it is the case of many applications; one of the most important is in model checking procedures for infinite state systems (see, e.g., [34]).

1.1. **Contributions.**

The paper presents two main contributions, that are strictly related but completely independent.

First, we prove non-constructively that given two quantifier-free formulae in the the theory of arrays with `diff`, it is possible to compute a quantifier-free interpolant. We do this by using the notion of *amalgamation* [20, 32]. Intuitively, a first-order theory has the amalgamation property if any two structures in its class of models sharing a common sub-model can be regarded as sub-structures of a larger model. A well-known result (see, e.g., [7]) states that if the class of models of a universal theory T (namely, a theory axiomatized by sentences obtained by prefixing a quantifier-free formula with a block of universal quantifiers) have the amalgamation property, then T admits quantifier-free interpolants for quantifier-free formulae in the theory and *vice versa*. Since the theory of arrays with `diff` is universal, we consider the problem of showing that its class of models has the amalgamation property. We provide a first, non-constructive, proof of this result by using model-theoretic notions only.

The second contribution of the paper is an algorithm for the generation of quantifier-free interpolants from finite sets (intended conjunctively) of literals in the theory of arrays with `diff`. Our algorithm uses as a sub-module a satisfiability procedure for sets of literals of the theory. Such a module is based on a sequence of syntactic manipulations organized in groups of syntactic transformations. The most important group of transformations is a Knuth-Bendix completion procedure (see, e.g., [4]) extended in such a way to solve an equation $a = wr(b, i, e)$ for b when this is required by the ordering defined on terms. (We call Gaussian completion this extended procedure because of its similarity with the techniques to handle Gaussian theories [3].) The goal of these transformations is to produce what we call a "modular" constraint for which it is trivial to establish satisfiability. Given two sets A and B of literals, the satisfiability procedure is invoked on A and B. While running, the two instances of the procedure exchange literals on the common signature of A and B (similarly to the Nelson and Oppen combination method, see, e.g., [51]) and perform some additional actions. At the end of the computation, the execution trace is examined and the desired interpolant is built by simple rules whose goal is to produce a set of literals on the common signature of A and B. In fact, the problem during the execution of Gaussian completion is to avoid the generation of equalities containing terms built out of non-shared symbols. Notice that our approach seems to be quite different from the standard method of extracting interpolants from an unsatisfiability proof of A and B in a given calculus (e.g., [11, 46]). Theoretically, it is not difficult to refine our proof of termination to show that the proposed algorithm is in NP, which is optimal since the satisfiability problem of quantifier-free formulae in the theory of arrays with extensionality is NP-complete (see, e.g., [10]).

1.2. **Plan of the paper.** In Section 2, we recall some background notions about theories, model-theoretic notions, and rewriting. In Section 3, we define the theory of arrays with `diff`, characterize its models, and show non-constructively that it admits quantifier-free interpolation. The rest of the paper is devoted to prove the same result constructively. In Section 4, we introduce modular constraints (which will be manipulated by the interpolation procedure) and state (and prove) their key properties. In Section 5, we describe the satisfiability solver for the theory of arrays with `diff` based on syntactic transformations of modular constraints. Then, in Section 6, we extend such as solver to produce quantifier-free interpolants by using a carefully designed set of meta-rules for interpolation. Finally, in Section 7, we extensively discuss the related work and conclude.

The appendix contains a proof of the result in [7] to make the paper self-contained.

2. Formal preliminaries

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, formula, and sentence) and semantic (e.g., structure, truth, satisfiability, and validity) notions of first-order logic. The equality symbol "=" is included in all signatures considered below. For clarity, we shall use "$\equiv$" in the meta-theory to express the syntactic identity between two symbols or two strings of symbols, or to introduce a new definition.

2.1. **Theories, constraints, interpolants.** A *theory* T is a pair (Σ, Ax_T), where Σ is a signature and Ax_T is a set of Σ-sentences, called the axioms of T (we shall sometimes write directly T for Ax_T). The Σ-structures in which all sentences from Ax_T are true are the *models* of T. A *universal* (resp. *existential*) sentence is obtained by prefixing a string of universal (resp. existential) quantifiers to a quantifier-free formula. A theory T is *universal* iff Ax_T consists of universal sentences. A Σ-formula ϕ is *T-satisfiable* if there exists a model $\mathcal{M}$ of T such that ϕ is true in $\mathcal{M}$ under a suitable assignment $\mathtt{a}$ to the free variables of ϕ (in symbols, $(\mathcal{M}, \mathtt{a}) \models \phi$); it is *$T$-valid* (in symbols, $T \vdash \varphi$) if its negation is T-unsatisfiable or, equivalently, iff φ is provable from the axioms of T in a complete calculus for first-order logic. A formula φ_1 *T-entails* a formula φ_2 if $\varphi_1 \to \varphi_2$ is *T-valid*; the notation used for such T-entailment is $\varphi_1 \vdash_T \varphi_2$ or simply $\varphi_1 \vdash \varphi_2$, if T is clear from the context. The *satisfiability modulo the theory T ($SMT(T)$) problem* amounts to establishing the T-satisfiability of quantifier-free Σ-formulae.

Let T be a theory in a signature Σ; a *T-constraint* (or, simply, a constraint) A is a set of ground literals in a signature Σ' obtained from Σ by adding a set of free constants. Taking conjunction, we can consider a finite constraint A as a single formula; thus, when we say that a constraint A is *T-satisfiable* (or just "satisfiable" if T is clear from the context), we mean that the associated formula (also called A) is satisfiable in a Σ'-structure which is a model of T. Let $a_1, \dots, a_n$ be the tuple of free constants occurring in a sentence A and $x_1, \dots, x_n$ be a tuple of fresh distinct individual variables, the formula $A^\exists$ is obtained from A by replacing each a_i with x_i (for $i = 1, ..., n$) and then existentially quantifying $x_1, \dots, x_n$, i.e. $A^\exists$ denotes the formula $\exists x_1 \cdots \exists x_n A(x_1/a_1, \dots, x_n/a_n)$. We have two notions of equivalence between constraints, which are summarized in the next definition.

Definition 2.1. Let A and B be finite constraints (or, more generally, first order sentences) in an expanded signature. We say that A and B are *logically equivalent* (modulo T) iff $T \vdash A \leftrightarrow B$; on the other hand, we say that they are $\exists$-*equivalent* (modulo T) iff $T \vdash A^{\exists} \leftrightarrow B^{\exists}$.

Logical equivalence means that the constraints have the same semantic content (modulo T); $\exists$-equivalence is also useful because we are mainly interested in T-satisfiability of constraints and it is trivial to see that $\exists$-equivalence implies equisatisfiability (again, modulo T). As an example, if we take a constraint A, we replace all occurrences of a certain term t in it by a fresh constant a and add the equality $a = t$, called the *(explicit) definition (of t)*, the constraint A' we obtain in this way is $\exists$-equivalent to A. As another example, suppose that $A \vdash_T a = t$, that a does not occur in t, and that A' is obtained from A by replacing a by t everywhere; then the following four constraints are $\exists$-equivalent

$$A, \quad A \cup \{a = t\}, \quad A' \cup \{a = t\}, \quad A'$$

(the first three are also pairwise logically equivalent). The above examples show how explicit definitions can be introduced and removed from constraints while preserving $\exists$-equivalence.

A theory T is said to *admit quantifier-free interpolation* (or, equivalently, to *have quantifier-free interpolants*) iff for every pair of quantifier free formulae ϕ, ψ such that $\psi \wedge \phi$ is not T satisfiable, there exists a quantifier free formula θ, called an *interpolant*, such that: (i) ψ T-entails θ; (ii) $\theta \wedge \phi$ is not T-satisfiable: (iii) only variables occurring both in ψ and in ϕ occur in θ.

2.2. **Some model theoretic concepts and results.**

We recall some basic model-theoretic notions that will be used in the paper (for more details, the interested reader is pointed to standard textbooks in model theory, such as [20]).

If Σ is a signature, we use the notation $\mathcal{M} = (M, \mathcal{I})$ for a Σ-structure, meaning that M is the support of $\mathcal{M}$ and $\mathcal{I}$ is the related interpretation function for Σ-symbols (in a many-sorted framework, the support is the disjoint union of the interpretations of the sorts symbols of Σ).

Roughly, an embedding is a homomorphism that preserves and reflects relations and operations. Formally, a Σ-*embedding* (or, simply, an embedding) between two Σ-structures $\mathcal{M} = (M, \mathcal{I})$ and $\mathcal{N} = (N, \mathcal{J})$ is any mapping $\mu : M \longrightarrow N$ among the corresponding support sets satisfying the following three conditions: (a) μ is a (sort-preserving) injective function; (b) μ is an algebraic homomorphism, that is for every n-ary function symbol f and for every $a_1, \dots, a_n \in M$, we have $f^{\mathcal{N}}(\mu(a_1), \dots, \mu(a_n)) = \mu(f^{\mathcal{M}}(a_1, \dots, a_n))$; (c) μ preserve and reflects interpreted predicates, i.e. for every n-ary predicate symbol P, we have $(a_1, \dots, a_n) \in P^{\mathcal{M}}$ iff $(\mu(a_1), \dots, \mu(a_n)) \in P^{\mathcal{N}}$. By using simple set-theory, it possible to show that every embedding can be factored in an isomorphism and an inclusion. This means that if μ is an embedding from $\mathcal{M}$ to $\mathcal{N}$, it is possible to assume that —up to an isomorphism—$\mathcal{M}$ is a substructure of $\mathcal{N}$, in the sense defined below.

If $M \subseteq N$ and the embedding $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ is just the identity inclusion $M \subseteq N$, we say that $\mathcal{M}$ is a *substructure* of $\mathcal{N}$ or that $\mathcal{N}$ is an *superstructure* of $\mathcal{M}$. Notice that a substructure of $\mathcal{N}$ is nothing but a subset of the carrier set of $\mathcal{N}$ which is closed under the Σ-operations and whose Σ-structure is inherited from $\mathcal{N}$ by restriction. In fact, given $\mathcal{N} = (N, \mathcal{J})$ and $G \subseteq N$, there exists the smallest substructure of $\mathcal{N}$ containing G in its carrier set. This is called the substructure *generated by* G and its carrier set can be characterized as the set of the elements $b \in N$ such that $t^{\mathcal{N}}(\underline{a}) = b$ for some Σ-term t and some finite tuple $\underline{a}$ from G (when we write $t^{\mathcal{N}}(\underline{a}) = b$, we mean that $(\mathcal{N}, \mathtt{a}) \models t(\underline{x}) = y$ for an assignment $\mathtt{a}$ mapping the $\underline{a}$ to the $\underline{x}$ and b to y). An easy—but fundamental—fact is that the truth of a universal (resp. existential) sentence is preserved through substructures (resp. through superstructures).

Let $\mathcal{M} = (M, \mathcal{I})$ be a Σ-structure which is generated by $G \subseteq M$. Let us expand Σ with a set of fresh free constants in such a way that in the expanded signature Σ_G there is a fresh free constant c_g for every $g \in G$ (write c_g directly with g for simplicity). Let $\mathcal{M}^G$ be the Σ_G-structure obtained from $\mathcal{M}$ by interpreting each c_g as g. The *Σ_G-diagram $\delta_{\mathcal{M}}(G)$ of $\mathcal{M}$* is the set of all ground Σ_G-literals L such $\mathcal{M}^G \models L$. When we speak of the diagram of $\mathcal{M}$ *tout court*, we mean the Σ_M-diagram $\delta_{\mathcal{M}}(M)$.

The following celebrated result [20] is simple, but nevertheless very powerful and it will be used in the rest of the paper.

Lemma 2.2 (Robinson Diagram Lemma)**.** *Let $\mathcal{M} = (M, \mathcal{I})$ be a Σ-structure which is generated by $G \subseteq M$ and $\mathcal{N} = (N, \mathcal{J})$ be another Σ-structure. Then, there is a bijective correspondence between Σ-embeddings $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ and Σ_G-expansions $\mathcal{N}^{(G)} = (N, \mathcal{J}^{(G)})$ of $\mathcal{N}$ such that $\mathcal{N}^{(G)} \models \delta_{\mathcal{M}}(G)$. The correspondence associates with μ the extension of $\mathcal{J}$ to Σ_G given by $\mathcal{J}^{(G)}(c_g) \equiv \mu(g)$.*

Notice that an embedding $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ is uniquely determined, in case it exists, by the image of the set of generators G: this is because the fact that G generates $\mathcal{M}$ implies (and is equivalent to) the fact that every $c \in M$ is of the kind $t^{\mathcal{M}}(\underline{g})$, for some term t and some $\underline{g}$ from G.

Intuitively, amalgamation is a property of collections of structures that guarantees that two structures in the collection can be glued into substructures of a larger one. Formally, a theory T is said to have the *amalgamation property* iff whenever we are given embeddings

$$\mu_1 : \mathcal{N} \longrightarrow \mathcal{M}_1, \qquad \mu_2 : \mathcal{N} \longrightarrow \mathcal{M}_2$$

among the models $\mathcal{N}, \mathcal{M}_1, \mathcal{M}_2$ of T, then there exists a further model $\mathcal{M}$ of T endowed with embeddings

$$\nu_1 : \mathcal{M}_1 \longrightarrow \mathcal{M}, \qquad \nu_2 : \mathcal{M}_2 \longrightarrow \mathcal{M}$$

such that $\nu_1 \circ \mu_1 = \nu_2 \circ \mu_2$. Notice that, up to isomorphism, we can limit ourselves in the above definition to the case in which μ_1, μ_2 are inclusions, i.e. to the case in which $\mathcal{N}$ is just a substructure of both $\mathcal{M}_1, \mathcal{M}_2$; in this case, $\mathcal{M}$ is said to be a T-amalgam of $\mathcal{M}_1$ and $\mathcal{M}_2$ over $\mathcal{N}$. (When the signature does not have ground terms of some sort, models $\mathcal{N}$ having empty domain(s) must be included in the definition of amalgamation property.)

Theorem 2.3 ([7])**.** *Let T be universal; then T admits quantifier free interpolants iff T has the amalgamation property.*

We emphasize that the hypothesis for T to be universal is necessary for the above result to hold. To make the paper self-contained, we include the proof of this result in Appendix A.

2.3. **Some term rewriting concepts and results.**

We shall need basic term rewriting system notions and results (see, e.g., [4]). In the following, we recall some of the most important ones for this paper.

The reflexive and transitive closure of a binary relation $\to$ is denoted with $\to^*$ and its transitive closure by $\to^+$. A binary relation $\to$ over a set E is *terminating* if there are no infinite sequence $e_0, e_1, ...$ of elements of E such that $(e_i, e_{i+1}) \in\rightarrow$, also written as $e_i \rightarrow e_{i+1}$, for every $i \geq 0$. The relation $\rightarrow\subseteq E \times E$ is *confluent* if there exists $v \in E$ such that $s \rightarrow^* v$ and $t \rightarrow^* v$ whenever $u \rightarrow^* s$ and $u \rightarrow^* t$, for $s, t, u \in E$. The relation $\rightarrow$ is *convergent* if it is both terminating and confluent.

A *rewrite rule* is an ordered pair of terms l and r, written as $l \rightarrow r$ (intuitively, the rule is used to replace instances of l with instances of r).[2] A *(term-)rewriting system* is a set R of rewrite rules, which induces a *rewrite relation* $\rightarrow_R$ (or simply $\rightarrow$ when R is clear from the context) on terms as follows: $\rightarrow_R$ is the relation that contains the pairs of terms (t, t') such that (for some $l \rightarrow r$ in R) the term t has a sub-term of the form $l\sigma$ for some substitution σ (in symbols $t \equiv t[l\sigma]$), and t' is obtained by replacing that subterm $l\sigma$ by $r\sigma$ in t (in symbols $t' \equiv t[r\sigma]$). Let s and t terms; we say that s and t are *joinable* w.r.t. a rewrite relation $\rightarrow$ (in symbols, $s \downarrow t$) when there exists a term u such that $s \rightarrow^* u$ and $t \rightarrow^* u$. A term t is *reducible* w.r.t. a rewrite relation $\rightarrow$ if there exists a term u such that $t \rightarrow u$; otherwise, t is *irreducible.* A term u is a *normal form* of t w.r.t. a rewrite relation $\rightarrow$ if $t \rightarrow^* u$ and u is irreducible. A rewrite relation is *ground convergent* when it is convergent once restricted to the set of ground terms. Convergent rewrite relations are interesting because they have unique normal forms. *KnuthBendix completion* is a procedure, based on superposition of critical pairs, for transforming a rewrite system into a confluent one (see, e.g., [4] for details). Termination of rewrite systems is undecidable.

A *quasi-ordering* is a reflexive and transitive relation. The *lexicographic path ordering* $\succ$ on a set of terms induced by a quasi-ordering $>$, called *precedence relation*, on the set of constant and function symbols on which the terms are built is defined as follows: $s = f(s_l, \ldots, s_m) \succ g(t_l, \ldots, t_n) = t$ iff
(1) $s_k \succ t$ or $s_k \equiv t$ for some $k \in \{1, \ldots, m\}$, or
(2) $f > g$ and $s \succ t_l$ for each $l \in \{1, \ldots, n\}$, or
(3) $f \equiv g$, $s_1 \equiv t_1$, ..., $s_{j-1} \equiv t_{j-1}$, $s_j \succ t_j$, $s \succ t_{j+1}$, ..., $s \succ t_n$ for some $j \in \{1, \ldots, n\}$.
If the precedence relation $>$ is also total, then so is $\succ$ once restricted to ground terms.

3. Theories of Arrays and Quantifier-free Interpolation

The McCarthy *theory of arrays* $\mathcal{AX}$ [43] has three sorts $\texttt{ARRAY}, \texttt{ELEM}, \texttt{INDEX}$ (called "array", "element", and "index" sort, respectively) and two function symbols rd and wr of appropriate arities; its axioms are:

$$\forall y, i, e. \qquad rd(wr(y, i, e), i) = e \tag{3.1}$$

$$\forall y, i, j, e. \qquad i \neq j \Rightarrow rd(wr(y, i, e), j) = rd(y, j). \tag{3.2}$$

The theory of *arrays with extensionality* $\mathcal{AX}_{\text{ext}}$ has the further axiom

$$\forall x, y. x \neq y \Rightarrow (\exists i.\ rd(x, i) \neq rd(y, i)),$$

called the 'extensionality' axiom. In this paper, we consider a variant of the McCarthy theory of arrays with extensionality, obtained by Skolemizing the axioms of extensionality. Formally, we define the *theory of arrays with* $\texttt{diff}$ $\mathcal{AX}_{\texttt{diff}}$ by adding the additional (Skolem) function $\texttt{diff}$ to the signature of $\mathcal{AX}_{\texttt{diff}}$ and replace the extensionality axiom by its Skolemization, namely

$$\forall x, y. \qquad x \neq y \Rightarrow rd(x, \texttt{diff}(x, y)) \neq rd(y, \texttt{diff}(x, y)). \tag{3.3}$$

[2]To avoid pathological cases, it is assumed that all variables occurring in r occur also in l.

The new symbol diff is binary and takes two arguments of sort ARRAY and returns an element of sort INDEX. The new axiom (3.3) constrains diff to return an index at which the two arrays in input store different values, whereas it returns an arbitrary value when input arrays are equal.

3.1. **A semantic argument for quantifier-free interpolation.** Here, we show that $\mathcal{AX}_{\texttt{diff}}$ does admit quantifier-free interpolation, contrary to $\mathcal{AX}_{\text{ext}}$ [37]. We do so by using a model-theoretic argument based on the equivalence between amalgamation of the models and admitting quantifier-free interpolation for universal theories (recall Theorem 2.3 in Section 2.2). Notice that $\mathcal{AX}_{\texttt{diff}}$ is universal whereas $\mathcal{AX}_{\text{ext}}$ is not.

Since amalgamation is a property of the models of a theory, we preliminarily discuss the class of models of $\mathcal{AX}_{\texttt{diff}}$. A model of $\mathcal{AX}_{\text{ext}}$ or $\mathcal{AX}_{\texttt{diff}}$ is *standard* when ARRAY is interpreted as the set of all functions from indexes to elements. In a standard model of $\mathcal{AX}_{\text{ext}}$ or $\mathcal{AX}_{\texttt{diff}}$, arrays are interpreted as functions, rd as function application, and wr as the point-wise update operation (i.e. the interpretation of $wr(a, i, e)$ returns the same values of the interpretation of a, except at the interpretation of index i where it returns the interpretation of e). Indeed, the class of models of $\mathcal{AX}_{\text{ext}}$ or $\mathcal{AX}_{\texttt{diff}}$ contains also non-standard models. This is because the axioms of both $\mathcal{AX}_{\text{ext}}$ and $\mathcal{AX}_{\texttt{diff}}$, being first-order formulae, do not constrain the interpretation of the sort ARRAY to contain all mappings from indexes to elements. (This is similar to the interpretation of function variables according to the Henkin semantics of second order logic; see, e.g., [27].) Fortunately, because of the extensionality axiom, it is easy to show (see below) that every model of such theories embeds into a standard one (recall the definition of embedding in Section 2.2). This means that any model is isomorphic to a sub-structure of a standard model in which arrays are interpreted as functions, although it might happen that not all functions are part of the interpretation of ARRAY in the model. As a consequence, whenever we want to test the validity of universal formulae or the satisfiability of constraints, we can—w.l.o.g.—consider only standard models. (This fact will be used in the proofs of some results in later sections, such as the proof of Lemma 4.3 where a standard model is built to show the satisfiability of a certain class of constraints of $\mathcal{AX}_{\texttt{diff}}$.)

We show that the universal theory $\mathcal{AX}_{\texttt{diff}}$ has the amalgamation properties so that, by Theorem 2.3, we are entitled to conclude that it admits quantifier-free interpolation. Recall from Section 2.2 that a universal theory has the amalgamation property if two of its models can be glued as substructures of a third model. Thus, we need to consider arbitrary models of $\mathcal{AX}_{\texttt{diff}}$, not only the standard ones. This is why we need more insight into arbitrary models of our theories and their relationship to standard ones.

Let us choose an arbitrary model $\mathcal{M}$ of $\mathcal{AX}_{\text{ext}}$. We can build the standard model $std(\mathcal{M})$ such that $\texttt{INDEX}^{std(\mathcal{M})} = \texttt{INDEX}^{\mathcal{M}}$ and $\texttt{ELEM}^{std(\mathcal{M})} = \texttt{ELEM}^{\mathcal{M}}$. To embed $\mathcal{M}$ into $std(\mathcal{M})$ is sufficient to associate with every $a \in \texttt{ARRAY}^{\mathcal{M}}$ the function mapping i to $rd^{\mathcal{M}}(a, i)$ (this is an embedding because of the extensionality axiom). In this way, we can identify $\texttt{ARRAY}^{\mathcal{M}}$ with a subset of the set of all functions $\texttt{ARRAY}^{std(\mathcal{M})}$. If we call *functional* a model $\mathcal{M}$ in which $\texttt{ARRAY}^{\mathcal{M}}$ is a subset of the set of functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$ (and in which $rd^{\mathcal{M}}, wr^{\mathcal{M}}$ have the standard meaning), we have just shown that *every model is isomorphic to a functional one.* (The argument extends to models of $\mathcal{AX}_{\texttt{diff}}$ although—in a standard model—the interpretation of diff is not fixed as the interpretations of rd and wr.) In this respect, the crucial question is the following: which subsets of the set $\texttt{ARRAY}^{\bar{\mathcal{M}}}$

in a standard model $\bar{\mathcal{M}}$ can be in the support $\mathtt{ARRAY}^{\mathcal{M}}$ of a functional model $\mathcal{M}$ (with $\mathtt{INDEX}^{\mathcal{M}} = \mathtt{INDEX}^{\bar{\mathcal{M}}}$, $\mathtt{ELEM}^{\mathcal{M}} = \mathtt{ELEM}^{\bar{\mathcal{M}}}$) that is a substructure of $\bar{\mathcal{M}}$? We shall answer the question by using the notion of "closure under cardinality dependence," that we formally define next.

Let a, b be elements of $\mathtt{ARRAY}^{\mathcal{M}}$ in a model $\mathcal{M}$ of $\mathcal{AX}_{\mathtt{diff}}$. We say that *$a$ and b are cardinality dependent* (in symbols, $\mathcal{M} \models |a-b| < \omega$) iff $\{i \in \mathtt{INDEX}^{\mathcal{M}} \mid \mathcal{M} \models rd(a,i) \neq rd(b,i)\}$ is finite. Cardinality dependency is obviously an equivalence relation.

Lemma 3.1. *Let $\mathcal{N}$, $\mathcal{M}$ be models of $\mathcal{AX}_{\mathtt{diff}}$ such that $\mathcal{M}$ is a substructure of $\mathcal{N}$. For every $a, b \in \mathtt{ARRAY}^{\mathcal{M}}$, we have that*

$$\mathcal{M} \models |a-b| < \omega \quad \text{iff} \quad \mathcal{N} \models |a-b| < \omega.$$

Proof. The right-to-left side is trivial because if $\mathcal{M} \models |a-b| < \omega$ then $\mathcal{M} \models a = wr(b, I, E)$, where $I \equiv i_1, \ldots, i_n$ is a list of terms of sort $\mathtt{INDEX}$, $E \equiv e_1, \ldots, e_n$ is a list of terms of sort $\mathtt{ELEM}$, and $wr(b, I, E)$ abbreviates the term $wr(wr(\cdots wr(a, i_1, e_1)\cdots), i_n, e_n)$ (this and similar notations will be discussed in more details in Section 4). Thus, also $\mathcal{N} \models a = wr(b, I, E)$ because $\mathcal{M}$ is a substructure of $\mathcal{N}$. Vice versa, suppose that $\mathcal{M} \not\models |a-b| < \omega$. This means that there are infinitely many $i \in \mathtt{INDEX}^{\mathcal{M}}$ such that $rd^{\mathcal{M}}(a,i) \neq rd^{\mathcal{M}}(b,i)$. Since $\mathcal{M}$ is a substructure of $\mathcal{N}$, there are also infinitely many $i \in \mathtt{INDEX}^{\mathcal{N}}$ such that $rd^{\mathcal{N}}(a,i) \neq rd^{\mathcal{N}}(b,i)$, i.e. $\mathcal{N} \not\models |a-b| < \omega$. □

We are now in the position to show how any functional model $\mathcal{M}$ of $\mathcal{AX}_{\mathtt{diff}}$ (i.e. up to isomorphism, any model whatsoever) can be obtained from a standard one. In order to produce any such $\mathcal{M}$, it is sufficient to take a standard model $\bar{\mathcal{M}}$, to let $\mathtt{INDEX}^{\mathcal{M}} \equiv \mathtt{INDEX}^{\bar{\mathcal{M}}}$, $\mathtt{ELEM}^{\mathcal{M}} \equiv \mathtt{ELEM}^{\bar{\mathcal{M}}}$, and to let $\mathtt{ARRAY}^{\mathcal{M}}$ to be equal to any subset of $\mathtt{ARRAY}^{\bar{\mathcal{M}}}$ that is *closed under cardinality dependence*, i.e. such that if $a \in \mathtt{ARRAY}^{\mathcal{M}}$ and $\bar{\mathcal{M}} \models |a-b| < \omega$, then b is also in $\mathtt{ARRAY}^{\mathcal{M}}$. In other words, functional substructures $\mathcal{M}$ of $\bar{\mathcal{M}}$ with $\mathtt{INDEX}^{\mathcal{M}} = \mathtt{INDEX}^{\bar{\mathcal{M}}}$ and $\mathtt{ELEM}^{\mathcal{M}} = \mathtt{ELEM}^{\bar{\mathcal{M}}}$ are in bijective correspondence with subsets of $\mathtt{ARRAY}^{\bar{\mathcal{M}}}$ closed under cardinality dependence.

A similar remark holds for embeddings. Suppose that $\mu : \mathcal{N} \longrightarrow \mathcal{M}$ is an embedding that restricts to an inclusion $\mathtt{INDEX}^{\mathcal{N}} \subseteq \mathtt{INDEX}^{\mathcal{M}}$, $\mathtt{ELEM}^{\mathcal{N}} \subseteq \mathtt{ELEM}^{\mathcal{M}}$ for $\mathcal{M}$ and $\mathcal{N}$ functional models of $\mathcal{AX}_{\mathtt{diff}}$. The action of the embedding μ on $\mathtt{ARRAY}^{\mathcal{N}}$ can be characterized as follows: take an element a for each cardinality dependence equivalence class, extend arbitrarily a to the set $\mathtt{INDEX}^{\mathcal{M}} \setminus \mathtt{INDEX}^{\mathcal{N}}$ to produce $\mu(a)$ and then define $\mu(b)$ for non representative b in the only possible way for wr to be preserved; i.e. if $\mathcal{N} \models b = wr(a, I, E)$ for a representative a, let $\mu(b)$ be $wr^{\mathcal{M}}(\mu(a), I, E)$.

By using the observation above, we are ready to show that $\mathcal{AX}_{\mathtt{diff}}$ has the amalgamation property.

Theorem 3.2. *The theory $\mathcal{AX}_{\mathtt{diff}}$ has the amalgamation property.*

Proof. Take two embeddings $\mu_0 : \mathcal{N} \longrightarrow \mathcal{M}_0$ and $\mu_1 : \mathcal{N} \longrightarrow \mathcal{M}_1$. As observed above, we can suppose—w.l.o.g.—that $\mathcal{N}, \mathcal{M}_0, \mathcal{M}_1$ are functional models, that μ_0, μ_1 restricts to inclusions for the sorts $\mathtt{INDEX}$ and $\mathtt{ELEM}$, and that $(\mathtt{ELEM}^{\mathcal{M}_0} \setminus \mathtt{ELEM}^{\mathcal{N}}) \cap (\mathtt{ELEM}^{\mathcal{M}_1} \setminus \mathtt{ELEM}^{\mathcal{N}}) = \emptyset$, $(\mathtt{INDEX}^{\mathcal{M}_0} \setminus \mathtt{INDEX}^{\mathcal{N}}) \cap (\mathtt{INDEX}^{\mathcal{M}_1} \setminus \mathtt{INDEX}^{\mathcal{N}}) = \emptyset$. To simplify our task, we can also suppose—again w.l.o.g.—that there exists some $e_i \in (\mathtt{ELEM}^{\mathcal{M}_i} \setminus \mathtt{ELEM}^{\mathcal{N}})$ and some $j_i \in (\mathtt{INDEX}^{\mathcal{M}_i} \setminus \mathtt{INDEX}^{\mathcal{N}})$ (i.e. that these sets are not empty), for $i = 0, 1$. (If this additional condition is not satisfied, it is sufficient to enlarge $\mathcal{M}_1, \mathcal{M}_2$ so that they satisfy it.) The amalgamated model $\mathcal{M}$ will be the standard model over $\mathtt{INDEX}^{\mathcal{M}_0} \cup \mathtt{INDEX}^{\mathcal{M}_1}$ and $\mathtt{ELEM}^{\mathcal{M}_0} \cup \mathtt{ELEM}^{\mathcal{M}_1}$. We need to define $\nu_i : \mathcal{M}_i \longrightarrow \mathcal{M}$ $(i = 0, 1)$ in such a way that $\nu_0 \circ \mu_0 = \nu_1 \circ \mu_1$. The only relevant point is the action of ν_i on $\mathtt{ARRAY}^{\mathcal{M}_i}$: as observed above, in order to define it, it is sufficient to extend any $a \in \mathtt{ARRAY}^{\mathcal{M}_i}$ to the indexes $k \in (\mathtt{INDEX}^{\mathcal{M}_{1-i}} \setminus \mathtt{INDEX}^{\mathcal{N}})$:

(I) we let the value $\nu_i(a)(k)$ be e_i in case there is no c such that $\mathcal{M}_i \models |a - \mu_i(c)| < \omega$;

(II) otherwise, we can do the following: take any such c such that $\mathcal{M}_i \models |a - \mu_i(c)| < \omega$ and put $\nu_i(a)(k) \equiv \mu_{1-i}(c)(k)$.

Because of Lemma 3.1 the choice of c in (II) above is immaterial. In fact, any other c' differs from c only w.r.t. a finite set of indices in $\mathcal{M}_i$. This also holds in $\mathcal{N}$ (by Lemma 3.1) and thus we have $\mathcal{N} \models c' = wr(c, I, E)$ for some $I \subseteq \mathtt{INDEX}^{\mathcal{N}}$. The latter implies that $\mu_{1-i}(c)$ and $\mu_{1-i}(c')$ cannot differ at any $k \in (\mathtt{ELEM}^{\mathcal{M}_{1-i}} \setminus \mathtt{ELEM}^{\mathcal{N}})$. This guarantees that $\nu_1 \circ \mu_1 = \nu_2 \circ \mu_2$.

In order to define $\mathtt{diff}^{\mathcal{M}}$ we can simply extend $\mathtt{diff}^{\mathcal{M}_1} \cup \mathtt{diff}^{\mathcal{M}_2}$ in such a way that axiom 3.3 holds. More precisely, we define $\mathtt{diff}^{\mathcal{M}}(a, b)$ as follows: (i) if for some $i = 0, 1$, we have that $a = \nu_i(a')$ and $b = \nu_i(b')$, then $\mathtt{diff}^{\mathcal{M}}(a, b)$ is taken to be $\mathtt{diff}^{\mathcal{M}_i}(a', b')$; (ii) otherwise it is defined to be any i such that $a(i) \neq b(i)$ (it is arbitrary whenever $a = b$). For this definition of $\mathtt{diff}^{\mathcal{M}}$ to be correct, it is sufficient to show that

Claim: *if $a = \nu_0(a_0) = \nu_1(a_1)$, then there exists c such that $a_0 = \mu_0(c)$ and $a_1 = \mu_1(c)$.*

To prove the claim, suppose that $a = \nu_0(a_0) = \nu_1(a_1)$. Then $\nu_0(a_0)$ and $\nu_1(a_1)$ must have been defined as in (II) above (otherwise they cannot coincide with each other at indexes j_0, j_1),[3] which means that there exists c_i such that for $i = 0, 1$ we have $\mathcal{M}_i \models |a_i - \mu_i(c_i)| < \omega$. Since $\nu_0(a_0) = a = \nu_1(a_1)$, this means that $\nu_0(\mu_0(c_0)) = \nu_1(\mu_1(c_0))$ and a differ only at finitely many indexes; the same is true for $\nu_1(\mu_1(c_1))$ and a, which in turns implies that $\nu_1(\mu_1(c_0))$ and $\nu_1(\mu_1(c_1))$ differ only at finitely many indexes too. The same consequently holds for c_0, c_1 in $\mathcal{N}$ too, for $\mu_0(c_0)$ and $\mu_0(c_1)$ in $\mathcal{M}_0$ and for $\mu_1(c_0)$ and $\mu_1(c_1)$ in $\mathcal{M}_1$. Thus, since the choice of c in (II) is immaterial, we can suppose—w.l.o.g.—that $c_0 = c_1$ (let us use just c to name it). Then, by (II) applied to the definition of $\nu_1(a_1)$, we have that $\nu_0(\mu_0(c)) = \nu_1(\mu_1(c))$ and $a = \nu_1(a_1)$ cannot differ at any $k \in (\mathtt{ELEM}^{\mathcal{M}_0} \setminus \mathtt{ELEM}^{\mathcal{N}})$. Similarly, $\nu_0(\mu_0(c)) = \nu_1(\mu_1(c))$ and a cannot differ at any $k \in (\mathtt{ELEM}^{\mathcal{M}_1} \setminus \mathtt{ELEM}^{\mathcal{N}})$. Thus a and $\nu_0(\mu_0(c)) = \nu_1(\mu_1(c))$ possibly differ only for $k \in \mathtt{INDEX}^{\mathcal{N}}$ and actually only for finitely many such k. But $a = \nu_0(a_0) = \nu_1(a_1)$, so the values of a at any $k \in \mathtt{INDEX}^{\mathcal{N}}$ belongs $\mathtt{ELEM}^{\mathcal{M}_0} \cap \mathtt{ELEM}^{\mathcal{M}_1} = \mathtt{ELEM}^{\mathcal{N}}$, which means that a is equal to $wr^{\mathcal{M}}(\nu_0(\mu_0(c)), I, E) = \nu_0(\mu_0(wr^{\mathcal{N}}(c, I, E)))$ for $I \subseteq \mathtt{INDEX}^{\mathcal{N}}$ and $E \subseteq \mathtt{ELEM}^{\mathcal{N}}$. In conclusion, we have that a is of the kind $\nu_0(\mu_0(\tilde{c})) = \nu_1(\mu_1(\tilde{c}))$ and from $a = \nu_0(a_0) = \nu_1(a_1)$, we get $a_0 = \mu_0(\tilde{c})$ and $a_1 = \mu_1(\tilde{c})$ because ν_0, ν_1 are injective. □

Before stating the main result of the paper which immediately follows from Theorems 2.3 and 3.2, it is interesting to observe the following about the Claim used in the proof of Theorem 3.2. The property mentioned in the Claim is known as *strong amalgamability property* in Universal Algebra and is key to derive quantifier-free interpolation in combination of theories [19]. The fact that $\mathcal{AX}_{\mathtt{diff}}$ enjoys strong amalgamability is crucial to transfer quantifier-free interpolation to combinations of $\mathcal{AX}_{\mathtt{diff}}$ with other important theories, like equality with uninterpreted symbols, difference logic, real arithmetic, appropriate variants of integer linear arithmetic, etc. We refer the reader to [19] for details.

Theorem 3.3. *The theory $\mathcal{AX}_{\mathtt{diff}}$ admits quantifier-free interpolation.*

[3]The Claim might be false in case $\mathtt{INDEX}^{\mathcal{M}_1} = \mathtt{INDEX}^{\mathcal{N}} = \mathtt{INDEX}^{\mathcal{M}_2}$, this is the reason why we enlarged $\mathtt{INDEX}^{\mathcal{M}_1}, \mathtt{INDEX}^{\mathcal{M}_2}$ by adding the extra indexes j_0, j_1.

We conclude this section with some observations concerning the theories $\mathcal{AX}_{\rm ext}$ and $\mathcal{AX}$. Lemma 3.1 holds also for the theory $\mathcal{AX}_{\rm ext}$ and the proof of Theorem 3.2 goes through also for $\mathcal{AX}_{\rm ext}$. However, according to Theorem 2.3 in Section 2.2, amalgamation alone is not sufficient for establishing quantifier-free interpolation for theories like $\mathcal{AX}_{\rm ext}$ which are not universal (for non universal theories one needs sub-amalgamability, not just amalgamability, see [19]). Indeed, $\mathcal{AX}_{\rm ext}$ is amalgamable but does not admit quantifier-free interpolation.

Despite being universal, $\mathcal{AX}$ is not amalgamable and thus it does not admit quantifier-free interpolation. Indeed, the left-to-right implication of Lemma 3.1 does not hold for $\mathcal{AX}$ as the arguments in the proof of Theorem 3.2. To get a formal counterexample to the amalgamability of $\mathcal{AX}$, consider the following situation. Let $\mathcal{N}$ be the $\mathcal{AX}$-model in which $\mathtt{ELEM}^{\mathcal{N}}$ and $\mathtt{INDEX}^{\mathcal{N}}$ are empty and $\mathtt{ARRAY}^{\mathcal{M}}$ contains two distinct elements, say a and b. As already observed, empty supports must be taken into account when showing the amalgamation property and, for $\mathcal{AX}$, the axiom of extensionality needs not be satisfied. Extend $\mathcal{N}$ to two standard models $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathtt{ELEM}^{\mathcal{M}_1} = \{e, e'\}, \mathtt{INDEX}^{\mathcal{M}_1} = \{i\}$ and $\mathtt{ELEM}^{\mathcal{M}_2} = \{d_1, d_2\}, \mathtt{INDEX}^{\mathcal{M}_2} = \{j_1, j_2\}$. Then, embed $\mathcal{N}$ into $\mathcal{M}_1$ by letting a, b differ at i (thus, e.g., $\mathcal{M}_1 \models a = wr(b, i, e) \wedge rd(b, i) = e'$) and embed $\mathcal{N}$ into $\mathcal{M}_2$ by letting a, b differ at both j_1 and j_2. Now, observe that amalgamation fails because we should have

$$\mathcal{M} \models a = wr(b, i, e) \wedge rd(a, j_1) \neq rd(b, j_1) \wedge rd(a, j_2) \neq rd(b, j_2) \wedge j_1 \neq j_2$$

in any amalgamated model $\mathcal{M}$ and this is in contradiction with the two axioms of $\mathcal{AX}$.

4. Modular constraints for Arrays with `diff` and their combinations

Theorem 3.3 is proved by semantic arguments, hence it does not give an interpolation algorithm; it only guarantees that, by enumerating quantifier free formulae, one can find sooner or later the desired interpolant. In the rest of the paper, we develop (*independently* of the results of Section 3) techniques based on rewriting and constraint solving to construct an algorithm computing quantifier-free interpolants for conjunctions of ground literals in $\mathcal{AX}_{\mathtt{diff}}$. Here, we introduce the notion of "modular constraint," which is the main data structure manipulated by the quantifier-free interpolation procedure and we prove two key properties. First, we show that the satisfiability of modular constraints can be easily detected (Lemma 4.3). Second, we prove that they can be combined in a modular way (Proposition 4.5).

Preliminarily, we introduce some notational conventions which are specific for constraints in the theory $\mathcal{AX}_{\mathtt{diff}}$. We use $a, b, \ldots$ to denote free constants of sort `ARRAY`, $i, j, \ldots$ for free constants of sort `INDEX`, and $d, e, \ldots$ for free constants of sort `ELEM`; $\alpha, \beta, \ldots$ stand for free constants of any sort. Below, we shall introduce non-ground rewriting rules involving (universally quantified) variables of sort `ARRAY`: for these variables, we shall use the symbols $x, y, z, \ldots$. We make use of the following abbreviations.

- [Nested write terms] By $wr(a, I, E)$ we indicate a nested write on the array variable a, where indexes are represented by the free constants list $I \equiv i_1, \ldots, i_n$ and elements by the free constants list $E \equiv e_1, \ldots, e_n$; more precisely, $wr(a, I, E)$ abbreviates the term $wr(wr(\cdots wr(a, i_1, e_1) \cdots), i_n, e_n)$. Notice that, whenever the notation $wr(a, I, E)$ is used, the lists I and E must have the same length; for empty I, E, the term $wr(a, I, E)$ conventionally stands for a.

Refl	$wr(a,I,E) = a \leftrightarrow rd(a,I) = E$ *Proviso*: $Distinct(I)$
Symm	$(wr(a,I,E) = b \wedge rd(a,I) = D) \leftrightarrow (wr(b,I,D) = a \wedge rd(b,I) = E)$ *Proviso*: $Distinct(I)$
Trans	$(a = wr(b,I,E) \wedge b = wr(c,J,D)) \leftrightarrow (a = wr(c, J \cdot I, D \cdot E) \wedge b = wr(c,J,D))$
Confl	$b = wr(a, I \cdot J, E \cdot D) \wedge b = wr(a, I \cdot H, E' \cdot F) \leftrightarrow$ $\leftrightarrow (b = wr(a,I,E) \wedge E = E' \wedge rd(a,J) = D \wedge rd(a,H) = F)$ *Proviso*: $Distinct(I \cdot J \cdot H)$
Red	$(a = wr(b,I,E) \wedge rd(b,i_k) = e_k) \leftrightarrow (a = wr(b, I{-}k, E{-}k) \wedge rd(b,i_k) = e_k)$ *Proviso*: $Distinct(I)$

Legenda: a and b are constants of sort `ARRAY`; $I \equiv i_1, \dots, i_n$, $J \equiv j_1, \dots, j_m$ and $H \equiv h_1, \dots, h_l$ are lists of constants of sort `INDEX`; $E \equiv e_1, \dots, e_n$, $E' \equiv e'_1, \dots, e'_n$, $D \equiv d_1, \dots, d_m$, and $F \equiv f_1, \dots, f_l$ are lists of constants of sort `ELEM`.

Figure 1: Key properties of write terms

- [Multiple read literals] Let a be a constant of sort `ARRAY`, $I \equiv i_1, \dots, i_n$ and $E \equiv e_1, \dots, e_n$ be lists of free constants of sort `INDEX` and `ELEM`, respectively; $rd(a,I) = E$ abbreviates the formula $rd(a,i_1) = e_1 \wedge \cdots \wedge rd(a,i_n) = e_n$.
- [Multiple equalities] If $L \equiv \alpha_1, \dots, \alpha_n$ and $L' \equiv \alpha'_1, \dots, \alpha'_n$ are lists of constants of the same sort, by $L = L'$ we indicate the formula $\bigwedge_{i=1}^n \alpha_i = \alpha'_i$.
- [Multiple distinctions] If $L \equiv \alpha_1, \dots, \alpha_n$ is a list of constants of the same sort, by $Distinct(L)$ we abbreviate the formula $\bigwedge_{i \neq j} \alpha_i \neq \alpha_j$.
- [Juxtaposition and subtraction] If $L \equiv \alpha_1, \dots, \alpha_n$ and $L' \equiv \alpha'_1, \dots, \alpha'_m$ are lists of constants, by $L \cdot L'$ we indicate the list $\alpha_1, \dots, \alpha_n, \alpha'_1, \dots, \alpha'_m$; for $1 \leq k \leq n$, the list $L - k$ is the list $\alpha_1, \dots, \alpha_{k-1}, \alpha_{k+1}, \dots, \alpha_n$.

Some key properties of equalities involving write terms are stated in the following lemma (see also Figure 1).

Lemma 4.1 (Key properties of write terms)**.** *The formulae in Figure 1 are all $\mathcal{AX}_{\texttt{diff}}$-valid under the assumption that their provisoes - if any - hold (when we say that a formula ϕ is $\mathcal{AX}_{\texttt{diff}}$-valid under the proviso π, we just mean that $\pi \vdash_{\mathcal{AX}_{\texttt{diff}}} \phi$).*

Proof. The properties in Figure 1 are all straightforward to derive. Here, we just sketch the proof of **Trans**itivity, as an example: one side is by replacement of equals; for the-right-to-left side, notice that the equalities $a = wr(c, J \cdot I, D \cdot E)$ and $b = wr(c,J,D)$ can be used as rewrite rules to rewrite both members of $a = wr(b,I,E)$ to the same term. $\square$

4.1. Modular constraints in $\mathcal{AX}_{\texttt{diff}}$.

A (ground) *flat* literal is a literal of the form $a = wr(b,I,E), rd(a,i) = e, \texttt{diff}(a,b) = i, \alpha = \beta, \alpha \neq \beta$. Notice that replacing a sub-term t with a fresh constant α in a constraint A and adding the corresponding defining equation $\alpha = t$ to A always produces an $\exists$-equivalent constraint; by repeatedly applying this method, one can show that every constraint is $\exists$-equivalent to a *flat* constraint, i.e., to one containing only flat literals. We split a flat constraint A into two parts, the *index* part A_I and the *main* part A_M: A_I contains the literals of the form $i = j, i \neq j, \texttt{diff}(a,b) = i$, whereas A_M contains the remaining literals, i.e., those of the form $a = wr(b,I,E), a \neq b, rd(a,i) = e, e = d, e \neq d$ (atoms $a = b$ are identified with literals $a = wr(b, \emptyset, \emptyset)$). We write $A =< A_I, A_M >$

to indicate the two parts of the constraint A. In the main part of a constraint, positive literals will be treated as rewrite rules; to get a suitable orientation, we use a *lexicographic path ordering* with a total precedence $>$ such that $a > wr > rd > \mathtt{diff} > i > e$, for all a, i, e of the corresponding sorts. This choice orients equalities $a = wr(b, I, E)$ *from left to right* when $a > b$; equalities like $a = wr(b, I, E)$ for $a < b$ or $a \equiv b$ will be called *badly orientable* equalities.

Definition 4.2. A constraint $A =< A_I, A_M >$ is said to be *modular* iff it is flat and the following conditions are satisfied (we let $\tilde{I}, \tilde{E}$ be the sets of free constants of sort `INDEX` and `ELEM` occurring in A):

(o) no positive index literal $i = j$ occurs in A_I;
(i) no negative array literal $a \neq b$ occurs in A_M;
(ii) A_M does not contain badly orientable equalities;
(iii) the rewriting system A_R given by the oriented positive literals of A_M joined with the rewriting rules

$$rd(wr(x,i,e),j) \to rd(x,j) \qquad \text{for } i,j \in \tilde{I},\ e \in \tilde{E},\ i \not\equiv j \qquad (4.1)$$

$$rd(wr(x,i,e),i) \to e \qquad \text{for } i \in \tilde{I},\ e \in \tilde{E} \qquad (4.2)$$

$$wr(wr(x,i,e),j,d) \to wr(wr(x,j,d),i,e) \qquad \text{for } i,j \in \tilde{I},\ e,d \in \tilde{E},\ i > j \qquad (4.3)$$

$$wr(wr(x,i,e),i,d) \to wr(x,i,d). \qquad \text{for } i \in \tilde{I},\ e,d \in \tilde{E} \qquad (4.4)$$

is confluent and ground irreducible;[4]
(iv) if $a = wr(b, I, E) \in A_M$ and i, e are in the same position in the lists I, E, respectively, then $rd(b,i) \not\downarrow_{A_R} e$;
(v) $\{\mathtt{diff}(a,b) = i, \mathtt{diff}(a',b') = i'\} \subseteq A_I$ and $a \downarrow_{A_R} a'$ and $b \downarrow_{A_R} b'$ imply $i \equiv i'$;
(vi) $\mathtt{diff}(a,b) = i \in A_I$ and $rd(a,i) \downarrow_{A_R} rd(b,i)$ imply $a \downarrow_{A_R} b$.

Condition (o) means that the index constants occurring in a modular constraint are implicitly assumed to denote distinct objects. This is supported also by the statement of Lemma 4.3 below, from which, it is evident that the addition of all the negative literals $i \neq j$ (for $i, j \in \tilde{I}$, with $i \not\equiv j$) does not compromise the satisfiability of a modular constraint, precisely because such negative literals are implicitly (already) part of the constraint. In Condition (i), negative array literals $a \neq b$ are not allowed because they can be replaced by suitable literals involving fresh constants and the `diff` operation (see axiom (3.3)). Rules (4.1) and (4.2) mentioned in condition (iii) reduce read-over-writes and rules (4.3) and (4.4) sort indexes in flat terms $wr(a, I, E)$ in ascending order. In addition, condition (iv) prevents further redundancies in our rules. Finally, conditions (v) and (vi) deal with `diff`. In particular, (v) says that `diff` is "well defined" and (vi) is a "conditional" translation of the contraposition of axiom (3.3).

The non-ground rules from Definition 4.2(iii) form a convergent rewrite system (critical pairs are confluent): this can be checked manually (and can be confirmed also by tools like

[4]The latter means that no rule can be used to reduce the left-hand or the right-hand side of another ground rule. Notice that ground rules from A_R are precisely the rules obtained by orienting an equality from A_M (rules (4.1)-(4.4) are not ground as they contain one *variable*, namely the array variable x).

SPASS or MAUDE). Ground rules from A_R are of the form

$$a \to wr(b, I, E), \tag{4.5}$$

$$rd(a, i) \to e, \tag{4.6}$$

$$e \to d. \tag{4.7}$$

Only rules of the form (4.7) can overlap with the non-ground rules (4.1)-(4.4), but the resulting critical pairs are trivially confluent. Thus, in order to check confluence of A_M, *only overlaps between ground rules* (4.5)-(4.7) *need to be considered* (this is the main advantage of our choice to orient equalities $a = wr(b, I, E)$ from left to right instead of right to left).

Lemma 4.3. *Suppose that A is modular. Then A is $\mathcal{AX}_{\texttt{diff}}$-satisfiable iff there is no element inequality $e \neq d$ in A_M such that $e \downarrow_{A_R} d$. Moreover, A is $\mathcal{AX}_{\texttt{diff}}$-satisfiable iff*

$$A \cup \{i \neq j \mid i, j \in \tilde{I}, i \not\equiv j\} \cup \{\alpha \neq \beta\}_{\alpha,\beta}$$

(varying α, β among the different pairs of element and array constants in normal form occurring in A) is $\mathcal{AX}_{\texttt{diff}}$-satisfiable.

Proof. Clearly, the satisfiability of A implies that for no negative index literal $e \neq d$ from A_M, we have that $e \downarrow_{A_R} d$. Assume conversely that this is the case: our aim is to build a model for $A \cup \{\alpha \neq \beta\}_{\alpha,\beta} \cup \{i \neq j\}_{i,j}$ (varying α, β and i, j as indicated in the statement of the Lemma). We can freely make the following *further assumption*: if a, i occur in A and a is in normal form, there is some e such that $rd(a, i) = e$ belongs to A (in fact, if this does not hold, it is sufficient to add a further equality $rd(a, i) = e$ - with fresh e - without destroying the modular property of the constraint).

Let I^* be the set of constants of sort `INDEX` occurring in A and let E^* be the set of constants of sort `ELEM` in normal form occurring in A (we have $I^* = \tilde{I}$ and $E^* \subseteq \tilde{E}$). Finally, we let X be the set of free constants of sort `ARRAY` occurring in A which are in normal form.

We build a model $\mathcal{M}$ as follows (the symbol $+$ denotes disjoint union):

- $\texttt{INDEX}^{\mathcal{M}} \equiv I^* + \{*\}$;
- $\texttt{ELEM}^{\mathcal{M}} \equiv E^* + X$;
- $\texttt{ARRAY}^{\mathcal{M}}$ is the set of total functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$, $rd^{\mathcal{M}}$ and $wr^{\mathcal{M}}$ are the standard read and write operations (i.e. $rd^{\mathcal{M}}$ is function application and $wr^{\mathcal{M}}$ is the operation of modifying the first argument function by giving it the third argument as a value for the second argument input);[5]
- for a constant i of sort `INDEX`, $i^{\mathcal{M}} \equiv i$ for all $i \in I^*$;
- for a constant e of sort `ELEM`, $e^{\mathcal{M}}$ is the normal form of e;
- for a constant a of sort `ARRAY` in normal form and $i \in I^*$, we put $a^{\mathcal{M}}(i)$ to be equal to the normal form of $rd(a, i)$ (this is some $e \in \texttt{ELEM}^{\mathcal{M}}$ by our further assumption above); we also put $a^{\mathcal{M}}(*) \equiv a$ (notice that $\texttt{ELEM}^{\mathcal{M}} \equiv E^* + X$, hence $a \in \texttt{ELEM}^{\mathcal{M}}$).
- for a constant a of sort `ARRAY` not in normal form, let $wr(c, I, E)$ be the normal form of a: we let $a^{\mathcal{M}}$ to be equal to $wr^{\mathcal{M}}(c^{\mathcal{M}}, I^{\mathcal{M}}, E^{\mathcal{M}})$ (This definition is correct because a and c cannot coincide; in fact, since $a < wr(a, I, E)$, the term $wr(a, I, E)$ cannot be the normal form of a.)
- we shall define $\texttt{diff}^{\mathcal{M}}$ later on.

[5]In the terminology used in Section 3.1, this means that $\mathcal{M}$ is a standard model.

It is clear that in this way we have that all constants α of sort `ELEM` or `ARRAY` are interpreted in such a way that, if $\hat{\alpha}$ is the normal form of α, then

$$\alpha^{\mathcal{M}} = \hat{\alpha}^{\mathcal{M}}. \tag{4.8}$$

Also notice that, by the definition of $a^{\mathcal{M}}$, if e is the normal form of $rd(a,i)$, then we have

$$rd(a,i)^{\mathcal{M}} = e^{\mathcal{M}} \tag{4.9}$$

in any case (whether a is in normal form or not). Finally, if $wr(c,I,E)$ is the normal form of a, then

$$a^{\mathcal{M}} = c^{\mathcal{M}} \quad \Rightarrow \quad (I = \emptyset \text{ and } E = \emptyset); \tag{4.10}$$

this is because the only rule that can reduce a must have a as left-hand side and $wr(c,I,E)$ as right-hand side (rules are ground irreducible), thus in the rule $a \to wr(c,I,E) \in A_M$ we must have $I = \emptyset, E = \emptyset$ in case $a^{\mathcal{M}} = c^{\mathcal{M}}$ (recall Definition 4.2(iv)). In more details, suppose that I and E are not empty and take $i \in I$ and $e \in E$ in corresponding positions. We have that $rd(c,i)^{\mathcal{M}} = rd^{\mathcal{M}}(c^{\mathcal{M}}, i^{\mathcal{M}}) = c^{\mathcal{M}}(i^{\mathcal{M}}) = a^{\mathcal{M}}(i^{\mathcal{M}}) = rd^{\mathcal{M}}(a^{\mathcal{M}}, i^{\mathcal{M}}) = rd(a,i)^{\mathcal{M}}$ (we used the definition of interpretation of a ground term, the fact that $rd^{\mathcal{M}}$ is interpreted as functional application and that $a^{\mathcal{M}} = c^{\mathcal{M}}$). Now, since $rd(a,i)$ normalizes to e, applying (4.9), we get that $rd(c,i)^{\mathcal{M}} = e^{\mathcal{M}}$, which means, again by (4.9), that $rd(c,i)$ normalizes to e too (e is in normal form, thus if $\tilde{e}$ is the normal form of $rd(c,i)$, we have that $\tilde{e}^{\mathcal{M}} = e^{\mathcal{M}}$ implies $e \equiv \tilde{e}$). This is contrary to Definition 4.2(iv).

Since A is modular, literals in A are flat. It is clear that all negative literals from A are true: in fact, a modular constraint does not contain inequalities between array constants, inequalities between index constants are true by construction and inequalities between element constants are true by the hypothesis of the Lemma. Also, if α, β are either element or array constants in normal form, we have $\alpha^{\mathcal{M}} \neq \beta^{\mathcal{M}}$ by construction (in particular, the interpretation of different array constants both in normal form differ at index $*$). Let us now consider positive literals in A: those from A_M are equalities of terms of sort `ELEM` or `ARRAY` and consequently are of the kind

$$e = d, \qquad a = wr(c,I,E), \quad rd(a,i) = e.$$

Since ground rules are irreducible, d is the normal form of e and $wr(c,I,E)$ is the normal form of a, hence we have $e^{\mathcal{M}} = d^{\mathcal{M}}$ and $a^{\mathcal{M}} = wr(c,I,E)^{\mathcal{M}}$ by (4.8) above. For the same reason a and e are in normal form in $rd(a,i) = e$, hence $rd(a,i)^{\mathcal{M}} = e^{\mathcal{M}}$ follows by construction.

It remains to define $\mathtt{diff}^{\mathcal{M}}$ in such a way that flat literals $\mathtt{diff}(a,b) = i$ from A_I are true and the axiom (3.3) is satisfied. Before doing that, let us observe that for all free constants a, b occurring in A, *we have that* $a^{\mathcal{M}} = b^{\mathcal{M}}$ *is equivalent to* $a \downarrow_{A_R} b$. In fact, one side is by (4.8); for the other side, suppose that $a^{\mathcal{M}} = b^{\mathcal{M}}$ and that $wr(c,I,E)$, $wr(c',I',E')$ are the normal forms of a and b, respectively. Then c must be equal to c', otherwise $a^{\mathcal{M}}$ and $b^{\mathcal{M}}$ would differ at index $*$. If either a or b is equal to c, trivially $a \downarrow_{A_R} b$ follows from (4.10). Otherwise, a and b are both reducible in A_R and since ground rules are irreducible and the only rules that can reduce an array constant have the left-hand side equal to that array constant, we have that $a \to wr(c,I,E)$ and $b \to wr(c,I',E')$ are both rules in A_R: as such, they are subject to Condition (iv) from Definition 4.2. First observe that we must have that $I \equiv I'$: otherwise, if there is $i \in I \setminus I'$, we could infer the following: (i) by (4.8), $b^{\mathcal{M}}(i) = c^{\mathcal{M}}(i)$; (ii) $c^{\mathcal{M}}(i)$ is the normal form of $rd(c,i)$ by construction; (iii) by $a^{\mathcal{M}} = b^{\mathcal{M}}$, $c^{\mathcal{M}}(i)$ is also equal to the normal form of the e having in the list E the same position as i in the list I, contrary to Condition (iv) from Definition 4.2. Since terms are normalized with respect to rule (4.3), I and I' coincide not only as sets, but also as lists; this means that the lists E and E' coincide too (the terms $wr(c, I, E)$, $wr(c, I, E')$ are in normal form and we have $wr(c, I, E)^{\mathcal{M}} = wr(c, I, E')^{\mathcal{M}}$). In more details, let $i, e, \tilde{e}$ be in the k-th positions in the lists I, E, E', respectively. From $wr(c, I, E)^{\mathcal{M}} = wr(c, I, E')^{\mathcal{M}}$, applying $rd^{\mathcal{M}}(-, i^{\mathcal{M}})$, we get $e^{\mathcal{M}} = \tilde{e}^{\mathcal{M}}$, i.e. $e \downarrow_{A_R} \tilde{e}$, which means $e \equiv \tilde{e}$ because $wr(c, I, E)$, $wr(c, I, E')$ are in normal form (in particular, their sub-terms $e, \tilde{e}$ are not reducible). In conclusion, $a \downarrow_{A_R} b$ holds.

Among the elements of $\texttt{ARRAY}^{\mathcal{M}}$, some of them are of the kind $a^{\mathcal{M}}$ for some free constant a of sort ARRAY occurring in A and some are not of this kind: we call the former 'definable' arrays. In principle, it could be that $a^{\mathcal{M}} = b^{\mathcal{M}}$ for different a, b, but we have shown that this is possible only when a and b have the same normal form.

We are ready to define $\texttt{diff}^{\mathcal{M}}$: we must assign a value $\texttt{diff}^{\mathcal{M}}(\mathrm{a}, \mathrm{b})$ to all pairs of arrays $\mathrm{a}, \mathrm{b} \in \texttt{ARRAY}^{\mathcal{M}}$. If a or b is not definable or if there are no a, b defining them such that $\texttt{diff}(a, b)$ occurs in A_I, we can easily find $\texttt{diff}^{\mathcal{M}}(\mathrm{a}, \mathrm{b})$ so that axiom (3.3) is true for a, b: one picks an index where they differ if they are not identical, otherwise the definition can be arbitrary. So let us concentrate into the case in which a, b are defined by constants a, b such that the literal $\texttt{diff}(a, b) = i$ occurs in A_I: in this case, we define $\texttt{diff}^{\mathcal{M}}(a^{\mathcal{M}}, b^{\mathcal{M}})$ to be i: Condition (v) from Definition 4.2 (together with the above observation that two constants defining the same array in $\mathcal{M}$ must have an identical normal form) ensures that the definition is correct and that all literals $\texttt{diff}(a, b) = i \in A_I$ becomes true. Finally, axiom (3.3) is satisfied by Condition (vi) from Definition 4.2 and the fact that $rd(a, i)^{\mathcal{M}} = rd(b, i)^{\mathcal{M}}$ is equivalent to $rd(a, i) \downarrow_{A_R} rd(b, i)$ (to see the latter, just recall (4.9)). ☐

Remark 4.4. As we said, the importance of Definition 4.2 lies in Lemma 4.3 and in Proposition 4.5 below. On the other hand, it is not true that if A is modular, then A entails (modulo $\mathcal{AX}_{\texttt{diff}}$) a positive literal $t = v$ iff $t \downarrow_{A_R} v$, even in case t, v are ground flat terms. As a counterexample, consider $A = \{rd(a, i) \to e\}$; we have $A \vdash_{\mathcal{AX}_{\texttt{diff}}} a = wr(a, i, e)$ but $a \not\downarrow_{A_R} wr(a, i, e)$. However, the proof of Lemma 4.3 shows that the following weaker—but still important—property holds: if A is modular and t, v are terms of the same sort *occurring in* A, then $A \vdash_{\mathcal{AX}_{\texttt{diff}}} t = v$ iff $t \downarrow_{A_R} v$. This may look unusual, however recall that our aim is not to decide equality by normalization but to have algorithms for satisfiability and interpolation.

4.2. **Combining modular constraints.** Let A, B be two constraints in the signatures Σ^A, Σ^B obtained from the signature Σ by adding some free constants and let $\Sigma^C \equiv \Sigma^A \cap \Sigma^B$. Given a term, a literal or a formula φ we call it:

- AB-*common* iff it is defined over Σ^C;
- A-*local* (resp. B-*local*) if it is defined over Σ^A (resp. Σ^B);
- A-*strict* (resp. B-*strict*) iff it is A-local (resp. B-local) but not AB-common;
- AB-*mixed* if it contains symbols in both $(\Sigma^A \setminus \Sigma^C)$ and $(\Sigma^B \setminus \Sigma^C)$;
- AB-*pure* if it does not contain symbols in both $(\Sigma^A \setminus \Sigma^C)$ and $(\Sigma^B \setminus \Sigma^C)$.

(Notice that, sometimes in the literature about interpolation, "A-local" and "B-local" are used to denote what we call here "A-strict" and "B-strict"). The following modularity result is crucial to justify our interpolation algorithm for $\mathcal{AX}_{\texttt{diff}}$.

Proposition 4.5. *Let $A = \langle A_I, A_M \rangle$ and $B = \langle B_I, B_M \rangle$ be constraints in expanded signatures Σ^A, Σ^B as above (here Σ is the signature of $\mathcal{AX}_{\mathtt{diff}}$); let A, B be both consistent and modular. Then $A \cup B$ is consistent and modular, in case all the following conditions hold:*

(O) *an AB-common literal belongs to A iff it belongs to B;*

(I) *every rewrite rule in $A_M \cup B_M$ whose left-hand side is AB-common has also an AB-common right-hand side;*

(II) *if a, b are both AB-common and $\mathtt{diff}(a, b) = i \in A_I \cup B_I$, then i is AB-common too;*

(III) *if a rewrite rule of the kind $a \to wr(c, I, E)$ is in $A_M \cup B_M$ and the term $wr(c, I, E)$ is AB-common, so is the constant a.*

Proof. Since we cannot rewrite AB-common terms to terms which are not, it is easy to see that $A_M \cup B_M$ is still convergent and ground irreducible; the other conditions from Definition 4.2 are trivial, except condition (v). The latter is guaranteed by the hypotheses (II)-(III) as follows: the relevant case is when, say $\mathtt{diff}(a, b) = i \in A_I$ is A-local and $\mathtt{diff}(a', b') = i' \in B_I$ is B-local. If $a \downarrow a'$, since A_M and B_M are ground irreducible, we have that a single rewrite step reduces both a and a' to their normal form, that is we have

$$a \to wr(c, I, E) \leftarrow a'.$$

Now $wr(c, I, E)$ is AB-common, because the rules $a \to wr(c, I, E), a' \to wr(c, I, E)$ are in A_M and in B_M, respectively. By hypothesis (III), we have that a and a' are AB-common too; the same applies to b, b' and hence to i, i' by (II). Thus $\mathtt{diff}(a', b') = i'$ is AB-common and belongs to A_I, hence $i \equiv i'$ because A is modular.

Since all conditions from Definition 4.2 are satisfied, $A \cup B$ is modular. Lemma 4.3 applies, thus yielding consistency. □

The above proof is so easy mainly because ground rewrite rules cannot superpose with the non ground rewrite rules (4.1)-(4.4) (with the exception of the rewrite rules $e \to d$, that may superpose but with trivially confluent critical pairs): this is the main benefit of our choice of orienting equalities $a = wr(b, I, E)$ from left-to-right (and not from right-to-left).

We conclude this section with a remark about the combination of modular constraints in $\mathcal{AX}_{\mathtt{diff}}$ with constraints in other theories. The theory $\mathcal{AX}_{\mathtt{diff}}$ is stably infinite (in all its sorts) but non-convex: this means that it is suitable for Nelson-Oppen combination, but that disjunctions of equalities (not just equalities) need to be propagated from an $\mathcal{AX}_{\mathtt{diff}}$-constraint, in case it is involved in a combined problem. Actually, this does not happen for modular constraints, as it is shown by the statement of Lemma 4.3. In other words, no disjunction of equalities needs to be propagated from a modular constraint A and only equalities that can be syntactically extracted from A need to be propagated.

5. A Solver for Arrays with `diff`

The first step towards the quantifier-free interpolation procedure for $\mathcal{AX}_{\mathtt{diff}}$ is the design of a satisfiability solver. Although a solver for this theory can be easily derived from existing solvers for $\mathcal{AX}$ or $\mathcal{AX}_{\text{ext}}$, we need a specific algorithm from which interpolants can be extracted. To do this, Lemma 4.3 will play an important role by allowing for the design of $\exists$-equivalence preserving transformations that, once successively applied to a given constraint A, will bring it to a consistent modular constraint (if possible). Failure of applying these transformations implies that A is unsatisfiable. In other words, the $\exists$-equivalence preserving transformations will determine whether a finite constraint A is satisfiable or not by transforming it into a modular $\exists$-equivalent constraint.

One of the key design choice underlying our transformations is to separate the "index" part, that will be handled by guessing, of a constraint from the "array" and "elem" parts, that will be subject to rewriting. Another important design decision is to distinguish a *preprocessing* and a *completion* phase. In the preprocessing phase, besides flattening (see, e.g., [2]) and similar operations, a complete guessing of equalities/inequalities among index constants will be performed. Indeed, this guessing will be realized by backtracking: if the completion phase will terminate in a failure, another guessing has to be tried and unsatisfiability can only be declared when all guessing fail. The completion phase will guarantee the confluence of the current rewriting system A_R, recall Definition 4.2. The confluence of A_R is the main requirement for a constraint to be modular.

5.1. **Preprocessing.** The preprocessing phase consists of the following sequential steps applied to our initial constraint A:

$\boxed{\text{Step 1}}$ Flatten A, by replacing sub-terms with fresh constants and by adding the related defining equalities.

$\boxed{\text{Step 2}}$ Replace array inequalities $a \neq b$ by the following literals (i, e, d are fresh)

$$\texttt{diff}(a, b) = i, \quad rd(b, i) = e, \quad rd(a, i) = d, \quad d \neq e.$$

$\boxed{\text{Step 3}}$ Guess a partition of index constants, i.e., for any pair of indexes i, j add either $i = j$ or $i \neq j$ (but not both of them); then remove the positive literals $i = j$ by replacing i by j everywhere (if $i > j$ according to the symbol precedence, otherwise replace j by i); if an inconsistent literal $i \neq i$ is produced, try with another guess (and if all guesses fail, report `unsat`).

$\boxed{\text{Step 4}}$ For all a, i such that $rd(a, i) = e$ does not occur in the constraint, add such a literal $rd(a, i) = e$ with fresh e.

At the end of the preprocessing phase, we get a finite set of flat constraints; *the disjunction of these constraints is $\exists$-equivalent to the original constraint.* For each of these constraints, go to the completion phase: *if the transformations below can be exhaustively applied (without failure) to at least one of the constraints, report* `sat`, *otherwise report* `unsat`. Failure can be caused by instructions (V) below.

The reason for inserting Step 4 above is just to simplify Orientation and Gaussian completion below. Notice that, even if rules $rd(a, i) \to e$ can be removed during completion, the following **invariant** is maintained: *terms $rd(a, i)$ always reduce to constants of sort* `ELEM`.

5.2. **Completion.** The completion phase consists in various transformations that should be non-deterministically executed until no rule or a failure instruction applies. For clarity, we divide the transformations into five groups.

(I) Orientation. This group contains a single instruction: get rid of badly orientable equalities, by using the equivalences **Refl**exivity and **Symm**etry of Figure 1; a badly orientable equality $a = wr(b, I, E)$ (with $a < b$), after normalization of the term $wr(b, I, E)$ with respect to the non-grund rules (4.3) – (4.4), is replaced by an equality of the form $b = wr(a, I, D)$ and by the equalities $rd(a, I) = E$ (all "read literals" required by the left-hand side of **Symm** comes from the above invariant). A badly orientable equality $a = wr(a, I, E)$ is removed and replaced by read literals only (or by nothing if I, E are empty).

(II) Gaussian completion. We now take care of the confluence of A_R (i.e., point (iii) of Definition 4.2). To this end, we consider all the critical pairs that may arise among our rewriting rules (4.5)-(4.7) (recall that there is no need to examine overlaps involving the non ground rules (4.1)-(4.4)). To treat the relevant critical pairs, we combine standard Knuth-Bendix completion for congruence closure with a specific method ("Gaussian completion") based on equivalences **Symm**etry, **Trans**itivity and **Confl**ict of Figure 1. The critical pairs are listed below. Two preliminary observations are in order. First, we normalize a critical pair by using $\rightarrow_*$ before recovering convergence by adding a suitably oriented equality and removing the parent equalities (the symbol $\rightarrow_*$ denotes the reflexive and transitive closure of the rewrite relation $\rightarrow$ induced by the rewrite rules $A_R \cup \{(4.1) - (4.4)\}$). Second, the provisos of all the equivalences in Figure 1 used below (i.e., **Symm**, **Trans**, and **Confl**) are satisfied because of the pre-processing Step 3 above.

(C1): $\boxed{wr(b_1, I_1, E_1) \;{}_*\!\!\leftarrow wr(b'_1, I'_1, E'_1) \leftarrow a \rightarrow wr(b'_2, I'_2, E'_2) \rightarrow_* wr(b_2, I_2, E_2)}$

with $b_1 > b_2$. We proceed in two steps. First, we use **Symm** (from right to left) to replace the parent rule $a \rightarrow wr(b'_1, I'_1, E'_1)$ with

$$wr(a, I_1, F) = b_1 \wedge rd(a, I_1) = E_1$$

for a suitable list F of constants of sort `ELEM` (notice that the equalities $rd(b_1, I_1) = F$, which are required to apply **Symm**, are already available because terms of the form $rd(b_1, j)$ for j in I_1 always reduce to constants of sort `ELEM` by the invariant resulting from the application of Step 4 in the pre-processing phase). Then, we apply **Trans** to the previously derived equality $b_1 = wr(a, I_1, F)$ and to the normalized second equality of the critical pair (i.e., $a = wr(b_2, I_2, E_2)$) and we derive

$$b_1 = wr(b_2, I_2 \cdot I_1, E_2 \cdot F) \wedge a = wr(b_2, I_2, E_2). \tag{5.1}$$

Hence, we are entitled to replace $b_1 = wr(a, I_1, F)$ with the rule $b_1 \rightarrow wr(b_2, J, D)$, where J and D are lists obtained by normalizing the right-hand-side of the first equality of (5.1) with respect to the non-ground rules (4.3) and (4.4). To summarize: the parent rules are removed and replaced by the rules

$$b_1 \rightarrow wr(b_2, J, D), \quad a \rightarrow wr(b_2, I_2, E_2)$$

and a bunch of new equalities of the form $rd(a, i) = e$, giving rise, in turn, to rules of the form $rd(b_2, i) \rightarrow e$ or to rewrite rules of the form (4.7) after normalization of their left members (normalization of terms $rd(a, i)$ is indeed needed for the termination argument of Theorem 5.1 below to work).

(C2): $\boxed{wr(b, I_1, E_1) \;{}_*\!\!\leftarrow wr(b'_1, I'_1, E'_1) \leftarrow a \rightarrow wr(b'_2, I'_2, E'_2) \rightarrow_* wr(b, I_2, E_2)}$

Since identities like $wr(c, H, G) = wr(c, \pi(H), \pi(G))$ are $\mathcal{AX}_{\mathtt{diff}}$-valid for every permutation π (under the proviso $Distinct(H)$), it is harmless to suppose that the set of index variables $I \equiv I_1 \cap I_2$ coincides with the common prefix of the lists I_1 and I_2; hence we have $I_1 \equiv I \cdot J$ and $I_2 \equiv I \cdot H$ for suitable disjoint lists J and H. Then, let E and E' be the prefixes of E_1 and E_2, respectively, of length equal to that of I; and let $E_1 \equiv E \cdot D$ and $E_2 \equiv E' \cdot F$ for suitable lists D and F. At this point, we can apply **Confl** to replace both parent rules forming the critical pair with

$$a = wr(b, I, E) \wedge E = E' \wedge rd(b, J) = D \wedge rd(b, H) = F,$$

where the first equality is oriented from left to right (i.e., $a \to wr(b, I, E)$).

(III) Knuth-Bendix completion. The remaining critical pairs are treated by standard completion methods for congruence closure.

(C3): $\boxed{d \;{}_*\!\!\leftarrow rd(wr(b, I, E), i) \leftarrow rd(a, i) \to e' \to_* e}$

Remove the parent rule $rd(a, i) \to e'$ and, depending on whether $d > e, e > d$, or $d \equiv e$, add the rule $d \to e$, $e \to d$, or do nothing. (Notice that terms of the form $rd(b, j)$ are always reducible because of the invariant of Step 4 in the pre-processing phase; hence, $rd(wr(b, I, E), i)$ always reduces to some constant of sort `ELEM`.)

(C4): $\boxed{e \;{}_*\!\!\leftarrow e' \leftarrow rd(a, i) \to d' \to_* d}$

Orient the critical pair (if e and d are not identical), add it as a new rule and remove one parent rule.

(C5): $\boxed{d \;{}_*\!\!\leftarrow d' \leftarrow e \to d'_1 \to_* d_1}$

Orient the critical pair (if d and d_1 are not identical), add it as a new rule and remove one parent rule.

(IV) Reduction. The instructions in this group simplify the current rewrite rules.

(R1): If the right-hand side of a current ground rewrite rule can be reduced, reduce it as much as possible, remove the old rule, and replace it with the newly obtained reduced rule. Redundant equalities like $t = t$ are also removed.

(R2): For every rule $a \to wr(b, I, E) \in A_M$, after normalization of the term $wr(b, I, E)$ with respect to the non-grund rules (4.3) – (4.4), exhaustively apply **Red**uction in Figure 1 from left to right (this amounts to do the following: if there are i, e in the same position k in the lists I, E such that $rd(b, i) \downarrow_{A_R} e$, replace $a = wr(b, I, E)$ with $a = wr(b, I\text{–}k, E\text{–}k)$).

(R3): If $\mathtt{diff}(a, b) = i \in A_I$, $rd(a, i) \downarrow_{A_R} rd(b, i)$ and $a > b$, add the rule $a \to b$; replace also $\mathtt{diff}(a, b) = i$ by $\mathtt{diff}(b, b) = i$ (this is needed for termination, it prevents the rule for being indefinitely applied).

(V) Failure. The instructions in this group aim at detecting inconsistency.

(U1): If for some negative literal $e \neq d \in A_M$ we have $e \downarrow_{A_R} d$, report `failure` and backtrack to Step 3 of the pre-processing phase.

(U2): If $\{\mathtt{diff}(a, b) = i, \mathtt{diff}(a', b') = i'\} \subseteq A_I$ and $a \downarrow_{A_R} a'$ and $b \downarrow_{A_R} b'$ for $i \not\equiv i'$, report `failure` and backtrack to Step 3 of the pre-processing phase.

Notice that the instructions in the last two groups may require a confluence test $\alpha \downarrow_{A_R} \beta$ that can be effectively performed in case the instructions from groups (II)-(III) have been exhaustively applied, because then all critical pairs have been examined and the rewrite system A_R is confluent. If this is not the case, one may pragmatically compute and compare any normal form of α and β, keeping in mind that the test has to be repeated when all completion instructions (II)-(III) have been exhaustively applied.

Theorem 5.1. *The above procedure decides constraint satisfiability in $\mathcal{AX}_{\mathtt{diff}}$.*

Proof. Correctness and completeness of the solver are clear: since all steps and instructions from Section 5 manipulate the constraint up to $\exists$-equivalence, it follows that if all guessings originated by Step 3 fail, the input constraint is unsatisfiable and, if one of them succeed, the exhaustive application of the completion instructions leads to a modular constraint which is satisfiable by Lemma 4.3.

We must only consider termination; to show that any sequence of our instructions terminates, we use a standard technique. With every positive literal $l = r$ we associate the multi-set of terms $\{l, r\}$; with every negative literal $l \neq r$, we associate the multi-set of terms $\{l, l, r, r\}$. Finally, with a constraint A we associate the multi-set $M(A)$ of the multi-sets associated with every literal from A. Now it is easy to see that such multi-set decreases after the application of any instruction. $\square$

The termination analysis in the proof of Theorem 5.1 can be refined so as to show that our algorithm is in NP, which is optimal because satisfiability of quantifier free formulae in $\mathcal{AX}_{\rm ext}$ is already NP-complete [10].

6. The Interpolation Algorithm for Arrays with `diff`

In the literature one can roughly distinguish two approaches to the problem of computing interpolants. In the former (see e.g. [11, 47]), an interpolating calculus is obtained from a standard calculus by adding decorations so as to enable the recursive construction of an interpolating formula from a proof; in the latter (see, e.g., [22, 28, 56]), the focus is on how to extend an available decision procedure to return interpolants. Our methodology is similar to the second approach, since we add the capability of computing interpolants to the satisfiability procedure in Section 5. However, we do this by designing a flexible and abstract framework, relying on the identification of *basic operations* that can be performed independently from the method used by the underlying satisfiability procedure to derive a refutation.

6.1. **Interpolating Metarules.** Let now A, B be constraints in signatures Σ^A, Σ^B expanded with free constants and $\Sigma^C \equiv \Sigma^A \cap \Sigma^B$; we shall refer to the definitions of AB-common, A-local, B-local, A-strict, B-strict, AB-mixed, AB-pure terms, literals and formulae given in Section 4. Our goal is to produce, in case $A \wedge B$ is $\mathcal{AX}_{\tt diff}$-unsatisfiable, a ground AB-common sentence ϕ such that $A \vdash_{\mathcal{AX}_{\tt diff}} \phi$ and $\phi \wedge B$ is $\mathcal{AX}_{\tt diff}$-unsatisfiable.

Let us examine some of the transformations to be applied to A, B. Suppose for instance that the literal ψ is AB-common and such that $A \vdash_{\mathcal{AX}_{\tt diff}} \psi$; then we can transform B into $B' \equiv B \cup \{\psi\}$. Suppose now that we got an interpolant ϕ for the pair A, B': clearly, we can derive an interpolant for the original pair A, B by taking $\phi \wedge \psi$. The idea is to collect some useful transformations of this kind. Notice that these transformations can also modify the signatures Σ^A, Σ^B, in the sense that the signature of the pair A', B' obtained after applying a single transformation to a pair A, B might be different from the signature of A, B (typically, the signature of A', B' may contain extra fresh constants). For instance, suppose that t is an AB-common term and that c is a fresh constant; then we can put $A' \equiv A \cup \{c = t\}$, $B' \equiv B \cup \{c = t\}$: in fact, if ϕ is an interpolant for A', B', then $\phi(t/c)$ is an interpolant for A, B. (Notice that the fresh constant c is now a shared symbol, because Σ^A is enlarged to $\Sigma^A \cup \{c\}$, Σ^B is enlarged to $\Sigma^B \cup \{c\}$ and hence $(\Sigma^A \cup \{c\}) \cap (\Sigma^B \cup \{c\}) = \Sigma^C \cup \{c\}$.) The transformations we need are called *metarules* and are listed in Table 1 below (in the Table and more generally in this Subsection, we use the notation $\phi \vdash \psi$ for $\phi \vdash_{\mathcal{AX}_{\tt diff}} \psi$).[6]

[6]Rules Redplus1, Redplus2 can be seen as instances of Rules Disjunction1, Disjunction2 (for $n = 1$), thus they are redundant. In Rule Propagate1, one can change the proviso to the weaker requirement '$\psi \in A$

An *interpolating metarules refutation* for A, B is a labelled tree having the following properties: (i) nodes are labelled by pairs of finite sets of constraints; (ii) the root is labelled by A, B; (iii) the leaves are labelled by a pair A, B such that $\bot \in A \cup B$; (iv) each non-leaf node is the conclusion of a rule from Table 1 and its successors are the premises of that rule. The crucial properties of the metarules are summarized in the following two Propositions.

Proposition 6.1. *The unary metarules* $\frac{A \mid B}{A' \mid B'}$ *from Table 1 have the property that* $A \wedge B$ *is* $\exists$*-equivalent to* $A' \wedge B'$*; similarly, the* n*-ary metarules* $\frac{A_1 \mid B_1 \cdots A_n \mid B_n}{A \mid B}$ *from Table 1 have the property that* $A \wedge B$ *is* $\exists$*-equivalent to* $\bigvee_{k=1}^{n}(A_k \wedge B_k)$.

Proposition 6.2. *If there exists an interpolating metarules refutation for* A, B *then there is a quantifier-free interpolant for* A, B *(namely there exists a quantifier-free* AB*-common sentence* ϕ *such that* $A \vdash \phi$ *and* $B \wedge \phi \vdash \bot$*). The interpolant* ϕ *is recursively computed applying the relevant interpolating instructions from Table 1.*

The proofs of both Propositions 6.1 and 6.2 are straightforward. The following observations are the basis of such proofs. The metarules are applied **bottom-up** whereas interpolants are computed (from an interpolating refutation) in a **top-down** manner. We should have labelled nodes in an interpolating metarules refutation by 4-tuples $(\Sigma^A, A, \Sigma^B, B)$, where Σ^A, Σ^B are signatures expanded with free constants, A is a Σ^A-constraint and B is a Σ^B-constraint. The *shared signature* of the node labelled $(\Sigma^A, A, \Sigma^B, B)$ (i.e. the signature where interpolants are recursively computed) is taken to be $\Sigma^C \equiv \Sigma^A \cap \Sigma^B$; the *root signature pair* is the pair of signatures comprising all symbols occurring in the original pair of constraints. We did not make all this explicit in order to avoid notation overhead. Notice that the only metarules that modify the signatures are (Define0), (Define1), (Define2) (which add a to $\Sigma^A \cap \Sigma^B, \Sigma^A, \Sigma^B$, respectively). Some other rules like (ConstElim0), (ConstElim1), (ConstElim2) could in principle restrict the signature, but signature restriction is not relevant for the computation of interpolants: there is no need that *all* AB-common symbols occur in the interpolants, but we certainly do not want *extra* symbols to occur in them, so only bottom-up signature expansion must be tracked.

6.2. **The Interpolating Solver.** The metarules are complete, i.e. if $A \wedge B$ is $\mathcal{AX}_{\mathtt{diff}}$-unsatisfiable, then (since we know that an interpolant exists) a single application of (Propagate1) and (Close2) gives an interpolating metarules refutation. This observation shows that metarules are by no means better than the brute force enumeration of formulae to find interpolants. However, metarules are useful to design an algorithm manipulating pairs of constraints based on transformation instructions. In fact, each of the transformation instructions can be *justified* by a metarule (or by a sequence of metarules): in this way, if our instructions form a complete and terminating algorithm, we can use Proposition 6.2 to get the desired interpolants. The main advantage of using metarules as justifications is that we just need to take care of the completeness and termination of the algorithm, and not about interpolants anymore. Here "completeness" means that our transformations should be able to bring a pair (A, B) of constraints into a pair (A', B') that either matches the requirements of Proposition 4.5 or is explicitly inconsistent, in the sense that $\bot \in A' \cup B'$.

and ψ is AB-common' (the case $A \vdash \psi$ could be obtained by applying Redplus1); a similar observation applies to Propagate2. We thank an anonymous referee for these remarks.

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

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

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

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

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

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

The latter is obviously the case whenever the original pair (A, B) is $\mathcal{AX}_{\mathtt{diff}}$-unsatisfiable and it is precisely the case leading to an interpolating metarules refutation.

The basic idea is that of invoking the algorithm of Section 5 on A and B separately and to propagate equalities involving AB-common terms. We shall assume *an ordering precedence making AB-common constants smaller than A-strict or B-strict constants of the same sort*. However, this is not sufficient to prevent the algorithm of Section 5 from generating literals and rules violating one or more of the hypotheses of Proposition 4.5: this is why the extra correcting instructions of group (γ) below are needed. Our interpolating algorithm has a pre-processing and a completion phase, like the algorithm from Section 5.

Pre-processing. In this phase the four Steps of Section 5.1 are performed on both A and B; to justify these steps we need metarules (Define0,1,2), (Redplus1,2), (Redminus1,2), (Disjunction1,2), (ConstElim0,1,2), and (Propagate1,2)—the latter because if i, j are AB-common, the guessing of $i = j$ versus $i \neq j$ in Step 3 can be done, say, in the A-component and then propagated to the B-component. At the end of the preprocessing phase, the following properties (to be maintained as invariants afterwards) hold:

(i1): A (resp. B) contains $i \neq j$ for all A-local (resp. B-local) constants i, j of sort `INDEX` occurring in A (resp. in B);

(i2): if a, i occur in A (resp. in B), then $rd(a, i)$ reduces to an A-local (resp. B-local) constant of sort `ELEM`.

Completion. Some groups of instructions to be executed non-deterministically constitute the completion phase. There is however an important difference here with respect to the completion phase of Section 5.2: it may happen that we need some *guessing* also inside the completion phase (only the instructions from group (γ) below may need such guessings). Each instruction can be easily justified by suitable metarules (we omit the straightforward details). The groups of instructions are the following:

(α) Apply to A or to B any instruction from the completion phase of Section 5.2.

(β) If there is an AB-common literal that belongs to A but not to B (or vice versa), copy it in B (resp. in A).

(γ) Replace *undesired literals*, i.e., those violating conditions (I)-(II)-(III) from Proposition 4.5.

To avoid trivial infinite loops with the (β) instructions, rules in (α) deleting an AB-common literal should be performed *simultaneously* in the A- and in the B-components (it can be easily checked - see the proof of Theorem 6.3 below - that this is always possible, *if rules in* (β) *and* (γ) *are given higher priority*).

Instructions (γ) need to be described in more details. Preliminarily, we introduce a technique that we call *Term Sharing*. Suppose that the A-component contains a literal $\alpha = t$, where the term t is AB-common but the free constant α is only A-local. Then it is possible to "make α AB-common" in the following way. First, introduce a fresh AB-common constant α' with the explicit definition $\alpha' = t$ (to be inserted both in A and in B, as justified by metarule (Define0)); then replace the literal $\alpha = t$ by $\alpha = \alpha'$ and replace α by α' everywhere else in A; finally, delete $\alpha = \alpha'$ too. The result is a pair (A, B) where basically nothing has changed but α has been renamed to an AB-common constant α'. Notice that the above transformations can be justified by metarules (Define0), (Redplus1), (Redminus1), (ConstElim1). We are now ready to explain instructions (γ) in details. First, consider undesired literals corresponding to the rewrite rules of the form

$$rd(c, i) \to d \tag{6.1}$$

in which the left-hand side is AB-common and the right-hand side is, say, A-strict. If we apply Term Sharing, we can solve the problem by renaming d to an AB-common fresh constant d'. We can apply a similar procedure to the rewrite rules

$$a \to wr(c, I, E) \tag{6.2}$$

in case the right-hand side is AB-common and the left-hand side is not; when we rename a to some fresh AB-common constant c', we must arrange the precedence so that $c' > c$ to orient the renamed literal as $c' \to wr(c, I, E)$. Then, consider the literals of the form

$$\mathtt{diff}(a, b) = k \tag{6.3}$$

in which the left-hand side is AB-common and the right-hand side is, say, A-strict. Again, we can rename k to some AB-common constant k' by Term Sharing. Notice that k' is AB-common, whereas k was only A-local: this implies that we might need to perform some guessing to maintain the invariant (i1). Basically, we need to repeat Step 3 from Section 5.1 till invariant (i1) is restored (k' must be compared for equality with the other B-local constants of sort `INDEX`). The last undesired literals to take care of are the rules of the form

$$c \to wr(c', I, E) \tag{6.4}$$

having an AB-common left-hand side but, say, only an A-local right-hand side (literals of the form $d = e$ are automatically oriented in the right way by our choice of the precedence). Notice that from the fact that c is AB-common, it follows (by our choice of the precedence) that c' is AB-common too. We can freely suppose that I and E are split into sub-lists I_1, I_2 and E_1, E_2, respectively, such that $I \equiv I_1 \cdot I_2$ and $E \equiv E_1 \cdot E_2$, where I_1, E_1 are AB-common, $I_2 \equiv i_1, \dots, i_n$, $E_2 \equiv e_1, \dots, e_n$ and for each $k = 1, \dots, n$ at least one from i_k, e_k is not AB-common. This n (measuring essentially the number of non AB-common symbols in (6.4)) is called the *degree* of the undesired literal (6.4): in the following, we shall see how to eliminate (6.4) or to replace it with a smaller degree literal. We first make a guess (see metarule (Disjunction1)) about the truth value of the literal $c = wr(c', I_1, E_1)$. In the first case, we add the positive literal to the current constraint; as a consequence, we get that the literal (6.4) is equivalent to $c = wr(c, I_2, E_2)$ and also to $rd(c, I_2) = E_2$ (see **Red** in Figure 1). In conclusion, in this case, the literal (6.4) is replaced by the AB-common rewrite rule $c \to wr(c', I_1, E_1)$ and by the literals $rd(c, I_2) = E_2$. In the second case, we guess that the negative literal $c \neq wr(c', I_1, E_1)$ holds; we introduce a fresh AB-common constant c'' together with the defining AB-common literal[7]

$$c'' \to wr(c', I_1, E_1) \tag{6.5}$$

(see metarule (Define0)). The literal (6.4) is replaced by the literal

$$c \to wr(c'', I_2, E_2). \tag{6.6}$$

We show how to make the degree of (6.6) smaller than n. In addition, we eliminate the negative literal $c \neq c''$ coming from our guessing (notice that, according to (6.5), c'' renames $wr(c', I_1, E_1)$). This is done as follows: we introduce fresh AB-common constants i, d, d'' together with the AB-common defining literals

$$\mathtt{diff}(c, c'') = i, \quad rd(c, i) \to d, \quad rd(c'', i) \to d'' \tag{6.7}$$

(see metarule (Define0)). Now it is possible to replace $c \neq c''$ by the literal $d \neq d''$ (see axiom (3.3)). Under the assumption $Distinct(I_2)$, the following statement is $\mathcal{AX}_{\mathtt{diff}}$ valid:

$$c = wr(c'', I_2, E_2) \wedge rd(c'', i) = d'' \wedge rd(c, i) = d \wedge d \neq d'' \to \bigvee_{k=1}^{n} (i = i_k \wedge d = e_k).$$

[7]We put $c > c'' > c'$ in the precedence. Notice that invariant (i2) is maintained, because all terms $rd(c'', h)$ normalize to an element constant. In case I_1 is empty, one can directly take c' as c''.

Thus, we get n alternatives (see metarule (Disjunction1)). In the k-th alternative, we can remove the constants i_k, e_k from the constraint, by replacing them with the AB-common terms i, d respectively (see metarules (Redplus1), (Redplus2), (Redminus1), (Redminus2),(ConstElim1),(ConstElim0)); notice that it might be necessary to complete the index partition. In this way, the degree of (6.6) is now smaller than n.

In conclusion, if we apply exhaustively Pre-Processing and Completion instructions above, starting from an initial pair of constraints (A, B), we can produce a tree, whose nodes are labelled by pairs of constraints (the successor nodes of a node labelled $(\tilde{A}, \tilde{B})$ are labelled by pairs of constraints that are obtained from $(\tilde{A}, \tilde{B})$ by applying an instruction). Notice that the branching in the tree is due to instructions that need guessing and that Pre-Processing instructions are applied only in the initial segment of a branch. We call such a tree an *interpolating tree* for (A, B). The following result shows that we obtained an interpolation algorithm for $\mathcal{AX}_{\mathtt{diff}}$.

Theorem 6.3. *Any interpolation tree for (A, B) is finite; moreover, it is an interpolating metarules refutation (from which an interpolant can be recursively computed according to Proposition 6.2) precisely iff $A \wedge B$ is $\mathcal{AX}_{\mathtt{diff}}$-unsatisfiable.*

Proof. Since all instructions can be justified by metarules and since our instructions bring any pair of constraints into constraints which are either manifestly inconsistent (i.e. contain $\perp$) or satisfy the requirements of Proposition 4.5, the second part of the claim is clear. We only have to show that all branches are finite (then König lemma applies).

A complication that we may face here is due to the fact that during instructions (γ), the signature is enlarged. However, notice that our instructions may introduce genuinely new AB-common array constants, however *they can only rename index constants, element constants and non AB-common array constants.* Moreover: (1) Term Sharing decreases the number of the constants which are not AB-common; (2) each call in the recursive procedure for the elimination of literals (6.4), *either* (2.i) renames to AB-common constants some constants which were not AB-common before, *or* (2.ii) just replaces a literal of the kind $c = wr(c', I_1 \cdot I_2, E_1 \cdot E_2)$ by the literals

$$c = wr(c', I_1, E_1), \qquad rd(c', I_2) = E_2$$

(see the first alternative following the guessing about truth of the literal $c = wr(c', I_1, E_1)$). Since there are only finitely many non AB-common constants at all, after finitely many steps neither Term Sharing nor (2.i) apply anymore. We finally show that instructions (α), (β) and (2.ii) (that do *not* enlarge the signature) cannot be executed infinitely many times either. To this aim, it is sufficient to associate with each pair of constraints $(\tilde{A}, \tilde{B})$ the complexity measure given by the multi-set of pairs (ordered lexicographically) $\langle m(L), N_L \rangle$ (varying $L \in \tilde{A} \cup \tilde{B}$), where $m(L)$ is the multi-set of terms associated with the literal L and N_L is 1 if $L \in \tilde{A} \setminus \tilde{B}$, 2 if $L \in \tilde{B} \setminus \tilde{A}$, and 0 if $L \in \tilde{A} \cap \tilde{B}$. In fact, the second component in the above pairs takes care of instructions (β), whereas the first component covers all the remaining instructions. Notice that it is important that, whenever an AB-common literal is deleted, the deletion happens simultaneously in both components (otherwise, the (β) instruction could re-introduce it, causing an infinite loop; our complexity measure does not decrease if an AB-common literal is replaced by smaller literals only in the A- or in the B-component): in fact, it can be shown (by inspecting the instructions from the completion phase of Subsection 5.2) that whenever an AB-common literal is deleted, the instruction that removes it involves only AB-common literals, if undesired literals are removed first.[8] Thus, if instructions in (β) and (γ) have priority (as required by our specifications above), AB-common literal deletions caused by (α) can be performed both in the A- and in the B-component (notice also that the instructions from (β) and (2ii) do not remove AB-common literals). $\square$

From the theorem above it immediately follows Theorem 3.3, that we have already proved in Section 3.1 by using model-theoretic notions (thus in a non-constructive way).

6.3. **An Example.** To illustrate our method, we describe the computation of an interpolant for the problem

$$\Pi \equiv (A_0,\ B_0)$$

where

$$\begin{array}{rcl} A_0 & \equiv & \{\ a = wr(b,i,d)\ \} \\ B_0 & \equiv & \{\ rd(a,j) \neq rd(b,j),\ rd(a,k) \neq rd(b,k),\ j \neq k\ \}. \end{array}$$

Notice that i, d are A-strict constants, j, k are B-strict constants, and a, b are AB-common constants with precedence $a > b$. The computation of the interpolant in our framework can be represented with a tree, growing upward from Π, in which each step can be identified with a set of appropriate metarules application.

To begin with we first apply Pre-Processing instructions to obtain

$$\begin{array}{rcl} A_1 & \equiv & \{\ a = wr(b,i,d),\ rd(a,i) = e_5,\ rd(b,i) = e_6\ \} \\ B_1 & \equiv & \{rd(a,j) = e_1,\ rd(b,j) = e_2,\ rd(a,k) = e_3,\ rd(b,k) = e_4,\ e_1 \neq e_2,\ e_3 \neq e_4,\ j \neq k\}. \end{array}$$

Since $a = wr(b,i,d)$ is an undesired literal of the kind (6.4), we generate the two subproblems

$$\begin{array}{rcl} \Pi_1 & \equiv & (A_1 \cup \{\ rd(b,i) = d,\ a = b\ \},\ B_1),\ \text{and} \\ \Pi_2 & \equiv & (A_1 \cup \{\ a \neq b\ \},\ B_1) \end{array}$$

(this is precisely the case in which there is no need of an extra AB-common constant c'').

Let us consider Π_1 first. Notice that $A \vdash a = b$, and $a = b$ is AB-common. Therefore we send $a = b$ to B_1, and we may derive the new equality $e_1 = e_2$ from the critical pair (C3) $e_1 \leftarrow rd(a,j) \rightarrow rd(b,j) \rightarrow e_2$, thus obtaining

$$\begin{array}{rcl} A_2 & \equiv & \{\ rd(b,i) = d,\ a = b,\ rd(a,i) = e_5,\ rd(b,i) = e_6\ \} \\ B_2 & \equiv & \{rd(b,j) = e_2,\ rd(a,k) = e_3,\ rd(b,k) = e_4,\ e_1 \neq e_2,\ e_3 \neq e_4,\ j \neq k,\ a = b,\ e_1 = e_2\}. \end{array}$$

Now B is inconsistent (as it contains both $e_1 \neq e_2$ and $e_1 = e_2$). The interpolant for Π_1 can be computed with the *interpolating instructions* of the metarules (Close2, Redplus2, Redmius2, Propagate1) resulting in

$$\varphi_1 \equiv a = b$$

as shown in Figure 2.

[8] Let us see an example by considering instruction (C3). This instruction removes a literal $rd(a,i) \rightarrow e'$ using a literal $a \rightarrow wr(b,I,E)$ (and possibly rewrite rules $rd(b,i) \rightarrow d'$ as well as rewrite rules that might reduce some of the e', d', E). Now, if $rd(a,i) \rightarrow e'$ is AB-common and all the other involved rules are not undesired literals, the instruction as a whole manipulates AB-common literals. As such, if (β) has been conveniently applied, the instruction can be performed simultaneously in the A- and in the B-component and our specification is precisely to do that.

$$
\begin{array}{rcc}
\text{Close2} & \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} & \top \\
 & \ldots \qquad\qquad \mid B_1' \cup \{\, a = b,\, e_1 = e_2 \,\} & \\
\text{Redminus2} & \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} & \top \\
 & \ldots \qquad\qquad \mid B_1 \cup \{\, a = b,\, e_1 = e_2 \,\} & \\
\text{Redplus2} & \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} & \top \\
 & \ldots \qquad\qquad \mid B_1 \cup \{\, a = b \,\} & \\
\text{Propagate1} & \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} & a = b \\
 & A_1 \cup \{\, rd(b,i) = d,\ a = b \,\} \mid B_1 &
\end{array}
$$

where
$B_1' \equiv B_1 \setminus \{rd(b,j) = e_2\}$

Figure 2: Interpolant derivation for Π_1 using metarules. The derivation is to be read bottom-up. The labels for the rules are shown on the left, while the partial interpolants, computed top-down, are shown on the right.

Then, let us consider branch Π_2. Recall that this branch originates from the attempt of removing the undesired rule $a \to wr(b,i,d)$. We introduce, in both A and B, the AB-common defining literals $\mathtt{diff}(a,b) = l, rd(a,l) = f_1, rd(b,l) = f_2$. In order to remove $a \neq b$, we introduce $f_1 \neq f_2$ in A, which is propagated to B, thus obtaining:

$$
\begin{aligned}
A_3 \ \equiv \ \{ \ & a = wr(b,i,d), \\
& \mathtt{diff}(a,b) = l,\ rd(a,l) = f_1,\ rd(b,l) = f_2,\ f_1 \neq f_2 \ \} \\
B_3 \ \equiv \ \{ \ & rd(a,j) = e_1,\ rd(b,j) = e_2,\ rd(a,k) = e_3,\ rd(b,k) = e_4, \\
& e_1 \neq e_2,\ e_3 \neq e_4,\ j \neq k, \\
& \mathtt{diff}(a,b) = l,\ rd(a,l) = f_1,\ rd(b,l) = f_2,\ f_1 \neq f_2 \ \}.
\end{aligned}
$$

Since $a = wr(b,i,d)$ contains only the index i, we do not have a real case split. Therefore we replace i with l, and d with f_1. At last, we propagate the AB-common literal $a = wr(b,l,f_1)$ to B. After all these steps we obtain:

$$
\begin{aligned}
A_4 \ \equiv \ \{ \ & a = wr(b,l,f_1), \\
& \mathtt{diff}(a,b) = l,\ rd(a,l) = f_1,\ rd(b,l) = f_2,\ f_1 \neq f_2 \ \} \\
B_4 \ \equiv \ \{ \ & rd(a,j) = e_1,\ rd(b,j) = e_2,\ rd(a,k) = e_3,\ rd(b,k) = e_4, \\
& e_1 \neq e_2,\ e_3 \neq e_4,\ j \neq k, \\
& \mathtt{diff}(a,b) = l,\ rd(a,l) = f_1,\ rd(b,l) = f_2,\ f_1 \neq f_2, \\
& a = wr(b,l,f_1) \ \}.
\end{aligned}
$$

Since we have one more AB-common index constant l, we complete the current index constant partition, namely $\{k\}$ and $\{j\}$: we have three alternatives, to let l stay alone in a new class, or to add l to one of the two existing classes. In the first alternative, because of the following critical pair (C3) $e_1 \leftarrow rd(a,j) \to rd(wr(b,l,f_1),j) \to e_2$, we add $e_1 = e_2$ to B, which becomes trivially unsatisfiable. The other two alternatives yield similar outcomes. For each sub-problem the interpolant is $\top$. The partial interpolant for Π_2 has to be reconstructed by the reverse application of the interpolanting instructions of (Define0) and (Propagate1), as shown in Figure 3, which yield

$$\varphi_2 \equiv (a = wr(b, \mathtt{diff}(a,b), rd(a, \mathtt{diff}(a,b))) \wedge rd(a, \mathtt{diff}(a,b)) \neq rd(b, \mathtt{diff}(a,b))).$$

$$\begin{array}{rcl}
\text{Close2} & \overline{\phantom{\dots \mid B_1'' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\}}} & \top \\
& \dots \mid B_1'' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\} & \\
\text{Redminus2} & \overline{\phantom{\dots \mid B_1'' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\}}} & \top \\
& \dots \mid B_1' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\} & \\
\text{Redplus2} & \overline{\phantom{\dots \mid B_1'' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\}}} & \top \qquad \vdots \qquad \vdots \\
& \dots \mid B_1' \cup C \cup \{l \neq k,\, l \neq j\} & \\
\text{Disjunction2} & \overline{\phantom{\dots \mid B_1'' \cup C \cup \{l \neq k,\, l \neq j,\, e_1 = e_2\} \qquad\qquad}} & \top \\
& \dots \mid B_1' \cup C & \\
\text{Propagate1} & \overline{\phantom{A_1' \cup \{f_1 \neq f_2,\, a = wr(b,l,f_1)\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\}}} & a = wr(b,l,f_1) \\
& A_1' \cup \{f_1 \neq f_2,\, a = wr(b,l,f_1)\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\} & \\
\text{Redminus1} & \overline{\phantom{A_1' \cup \{f_1 \neq f_2,\, a = wr(b,l,f_1)\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\}}} & a = wr(b,l,f_1) \\
& A_1 \cup \{f_1 \neq f_2,\, a = wr(b,l,f_1)\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\} & \\
\text{Redplus1} & \overline{\phantom{A_1' \cup \{f_1 \neq f_2,\, a = wr(b,l,f_1)\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\}}} & a = wr(b,l,f_1) \\
& A_1 \cup \{f_1 \neq f_2\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\} & \\
\text{Propagate1} & \overline{\phantom{A_1 \cup \{f_1 \neq f_2\} \cup C \mid B_1 \cup C \cup \{f_1 \neq f_2\}}} & a = wr(b,l,f_1) \wedge f_1 \neq f_2 \\
& A_1 \cup \{f_1 \neq f_2\} \cup C \mid B_1 \cup C & \\
\text{Redminus1} & \overline{\phantom{A_1 \cup \{a \neq b,\, f_1 \neq f_2\} \cup C \mid B_1 \cup C}} & a = wr(b,l,f_1) \wedge f_1 \neq f_2 \\
& A_1 \cup \{a \neq b,\, f_1 \neq f_2\} \cup C \mid B_1 \cup C & \\
\text{Redplus1} & \overline{\phantom{A_1 \cup \{a \neq b,\, f_1 \neq f_2\} \cup C \mid B_1 \cup C}} & a = wr(b,l,f_1) \wedge f_1 \neq f_2 \\
& A_1 \cup \{a \neq b\} \cup C \mid B_1 \cup C & \\
\text{Define0*} & \overline{\phantom{A_1 \cup \{a \neq b\} \cup C \mid B_1 \cup C}} & \varphi_2 \\
& A_1 \cup \{a \neq b\} \mid B_1 &
\end{array}$$

where
$C \equiv \{\mathtt{diff}(a,b) = l,\ rd(a,l) = f_1,\ rd(b,l) = f_2\}$
$A_1' \equiv A_1 \setminus \{a = wr(b,i,d)\}$
$B_1' \equiv B_1 \cup \{f_1 \neq f_2,\ a = wr(b,l,f_1)\}$
$B_1'' \equiv B_1' \setminus \{rd(a,j) = e_1\}$
$\varphi_2 \equiv (a = wr(b, \mathtt{diff}(a,b), rd(a, \mathtt{diff}(a,b))) \wedge rd(a, \mathtt{diff}(a,b)) \neq rd(b, \mathtt{diff}(a,b)))$

Figure 3: Interpolant derivation for Π_2 using metarules. The derivation is to be read bottom-up. The labels for the rules are shown on the left, while the partial interpolants, computed top-down, are shown on the right.

The final interpolant is computed by combining the interpolants for Π_1 and Π_2 by means of (Disjunction1), yielding

$$\begin{array}{rcl}
\varphi & \equiv & \varphi_1 \vee \varphi_2 \equiv \\
& \equiv & (a = b \vee (a = wr(b, \mathtt{diff}(a,b), rd(a, \mathtt{diff}(a,b))) \wedge \\
& & \wedge\ rd(a, \mathtt{diff}(a,b)) \neq rd(b, \mathtt{diff}(a,b)))
\end{array}$$

which can be simplified to $\varphi \equiv (a = wr(b, \mathtt{diff}(a,b), rd(a, \mathtt{diff}(a,b))))$.

7. Related work and Conclusions

There are two main lines of work in the literature which is relevant for our paper: satisfiability procedures for variants and extensions of the theory of arrays and interpolation methods related to the theory of arrays. Below, we discuss the works which are more closely related to our approach in some details.

7.1. Satisfiability.

Satisfiability. Since its introduction by McCarthy in [43], the theory of arrays have received a lot of attention in automated theorem proving and verification because of its importance in modelling fundamental mechanisms of hardware and software systems such as memory read and write operations. For example, a lot of papers have been devoted to design, prove correct, and build decision procedures for the satisfiability problem of quantifier-free and selected classes of quantified formulae in (various extensions of) the theory of arrays; e.g., [2, 8, 14, 25, 29–31, 38, 41, 55]. The interested reader is pointed to the 'related work' sections of [25, 30] for a comprehensive overview. Here, we notice that many of them are based on instantiating the axioms of the theory so that rd and wr can be considered as uninterpreted functions and state-of-the-art procedures for the theory of equality can be used. Notable exceptions are [2, 41, 55] where techniques based on rewriting or constraint solving are used.

In [2], the standard superposition calculus [5] is proven to terminate on the union of the theory of arrays and a set of ground literals; thereby, providing a decision procedure for the quantifier-free satisfiability problem because of the refutation completeness of the calculus. (The efficiency of the approach is explored in [1].) While the saturation (roughly, the exhaustive application of the rules of the superposition calculus) can be seen as a generalization of completion where clauses, and not only equalities, are handled, our Gaussian completion[9] has some distinctive features. In fact, while the three critical pairs (C3), (C4), and (C5) in Section 5.2 can be regarded as instances of the inference rules of a superposition calculus (see [2] for details), the critical pairs (C1) and (C2), exploiting the equivalences in Figure 1, are impossible to recast in any standard completion procedure (see, e.g., [4]). In fact, the way in which the critical pairs (C1) and (C2) are eliminated involves the addition of equalities containing rd's (in order to constrain the values stored at certain locations in the arrays mentioned in the rules of the critical pair) besides the replacement of one or both the parent rewrite rules by an equality. Only in this way, we were able to eliminate badly orientable rules. It seems difficult to adapt the approach in [2] to the problem under consideration mainly because of the chosen order $>$ over terms. In fact, we orient the equality $a \to wr(b, i, e)$ from left to right if $a > b$, and use the equivalences in Figure 1 when $b > a$ (or a and b are identical). This allows us to eliminate all critical pairs with rules (4.1)–(4.4) in Definition 4.2 since such rules contain just one variable of sort `ARRAY` and, trivially, no critical pairs involving the variable should be considered. If we choose the other way of orienting the equalities of the form $a = wr(b, i, e)$, several critical pairs would arise. Although the completion of these pairs terminate under suitable assumptions (as shown in [2]), this creates serious problems when considering the computation of interpolants.

In [41], a satisfiability procedure for the theory of arrays with extensionality is designed so as to be easily combined with other procedures by the Shostak combination method (see, e.g., [51]). Two interface functionalities are required by the Shostak combination method: (i) normalizing terms and (ii) solving equalities. We consider each activity in details.

(i) In Chapter 5 of [41], a canonical form for terms built out by using a single rd or several wr's is defined by using a simplification ordering. The canonical terms are similar to those occurring in a modular constraint according to Definition 4.2 above. A major difference is the use of if-then-else's to normalize read-terms in [41] while our procedure does not use them because item (i) of Definition 4.2 implies that any two indexes in a constraint in normal form are known to be distinct. This choice makes the proof of the correctness of our procedure much easier with respect to the argument for the correctness proposed in [41] which "*has proved elusive to the authors*" of [55]. So called 'lazy' SMT solvers, based on the integration of a SAT solver and a satisfiability

[9]The Gauss elimination procedure for systems of linear equalities has been lifted to elementary theories in [3] and, since the theory of arrays is close to being Gaussian [15], we show that 'Gaussian-like' steps can be exploited during completion phase.

procedures for conjunction of literals, seem to be able to easily implement the case-splitting required to derive a complete partition by resorting to the available SAT solver as explained, e.g., in [9].

(ii) To compare with the activity of solving equalities in [41], let us preliminarily observe that the logical equivalences in Figure 1 can be considered as rewrite rules (either from left to right or viceversa) that help us replace badly orientable equalities (recall the definition at the beginning of Section 4) with equalities which are oriented from left to right. This is precisely how the equivalences in Figure 1 are used in the Gaussian completion procedure (of Section 5.2) to eliminate critical pairs. Similarly, in order to provide one of the basic functionalities required by the Shostak combination framework, [41] designs a solver for equalities involving wr operations. For example, the procedure in [41] allows one to solve the equality $a = wr(b, i, e)$ for b. We can adapt our procedure (in particular, by using the equivalences **Symm** and **Refl** of Figure 1) to do the same. The main difference is that our normalization is done off-line, i.e. the signature is fixed since all terms appearing in the constraint are given, while the procedure in [41] must be on-line since is to be integrated in a Shostak combination algorithm which requires that to process equalities one at a time, as soon as they become available. Because of this, the completion algorithm can be simplified (since there is no need to compute intermediate normal forms) and standard techniques to show its termination can be used. In contrast, [41] gives only a brief sketch of the termination of his procedure. For a more comprehensive comparison of on-line and off-line completion algorithms revisiting the Shostak congruence closure algorithm, the reader is pointed to [6, 36].

The procedure in [55] share with [41] and ours the key activity of solving equalities. The main difference is that no canonical forms for terms or constraints are defined in [55]; rather a special form of equality over arrays is introduced, called partial equality, which compares the content of two arrays only at a (finite) set of indexes. Formally, this is defined as follows: $a =_I b$ iff for every index i not in the set I, the content of a at i is equal to that of b at the same index I. Thus, an equality of the form $wr(a, i, e) = b$ can be rewritten as $a =_{\{i\}} b \wedge rd(b, i) = e$. The key insight of [55] is that it is possible to eliminate all wr's, so that arrays can be considered as uninterpreted functions and rd as function application, and a slightly modified congruence closure (to cope with partial equality) can be used to check satisfiability. While no standard rewriting techniques are used in [55], it is interesting to notice that two arrays a and b are cardinality dependent iff there exists a finite set I of indexes such that $a =_I b$. We do not introduce a new predicate symbol and use it in designing a satisfiability procedure, however we nevertheless exploit this notion and its preservation through embeddings (see Lemma 3.1) during our semantic interpolation proofs.

7.2. **Interpolation.**

After McMillan's seminal work on interpolation for model checking [45, 48], several papers [11, 21, 22, 34, 37, 39, 42, 46, 52, 54, 56] appeared whose aim was to design techniques for the efficient computation of interpolants in first-order theories of interest for verification, mainly uninterpreted function symbols, fragments of Linear Arithmetic, or their combination. An interpolating theorem prover is described in [47], where a sequent-like calculus is used to derive interpolants from proofs in propositional logic, equality with uninterpreted functions, linear rational arithmetic, and their combinations. The method described in [56] proposes a framework suitable for lazy SMT-solvers, in which the theory solver is required to derive partial interpolants for each theory lemmata it produces. The global interpolant can then be computed at the propositional level. The paper also illustrates a method to derive interpolants in a Nelson-Oppen combination procedure, under certain restrictions on the theories to combine. More recently, in [22] the ideas of [56] are adapted to cope with state-of-the-art SMT-solving strategies for combinations of the theories of uninterpreted functions and a fragment of Linear Arithmetic (called difference logic). In [37], a method to compute interpolants in data structures theories, such as sets and arrays (with extensionality), by axiom instantiation and interpolant computation in the theory of uninterpreted functions is described. It is also shown that the theory of arrays with extensionality does not admit quantifier-free interpolation. The "split" prover in [34] applies a sequent calculus for the synthesis of interpolants along the lines of that in [47] and is tuned for predicate abstraction [53]. In particular, the method is shown to be complete in the sense that the computed interpolants are guaranteed to provide the "right" level abstraction to prove a certain property, if one exists. The "split" prover can handle a combination of theories among which also the theory of arrays without extensionality is considered. In [34], it is pointed out that the theory of arrays poses serious problems in deriving quantifier-free interpolants because it entails an infinite set of quantifier-free formulae, which is indeed problematic when interpolants are to be used for predicate abstraction. To overcome the problem, [34] suggests to constrain array valued terms to occur in equalities of the form $a = wr(a, I, E)$ in the notation of this paper. It is observed that this corresponds to the way in which arrays are used in imperative programs. Further limitations are imposed on the symbols in the equalities in order to obtain a complete predicate abstraction procedure. In [35], the method described in [34] is specialized to apply CEGAR techniques [23] for the verification of properties of programs manipulating arrays. The method of [34] is extended to cope with range predicates which allow one to describe unbounded array segments which permit to formalize typical programming idioms of arrays, yielding property-sensitive abstractions. In [54], it is shown how to extend satisfiability procedures based on axiom instantiation to compute interpolants. However, the theory of arrays is not considered. In [52], the approach of [54] is specialized to compute interpolants in the combination of Linear Rational Arithmetic and the theory of uninterpreted function symbols; again, the theory of arrays is not considered. A method for deriving interpolants in the theory of equality with uninterpreted functions is also given in [28] by extending a congruence closure algorithm. In [39], a method to derive quantified invariants for programs manipulating arrays and integer variables is described. A resolution-based prover is used to handle an *ad hoc* axiomatization of arrays by using predicates. Neither McCarthy's theory of arrays nor one of its extensions are considered in [39]. The invariant synthesis method is based on the computation of interpolants derived from the proofs of the resolution-based prover and constraint solving techniques to handle the arithmetic part of the problem. The resulting interpolants may contain even alternation of quantifiers.

Latest research on interpolating procedures has been focusing on (extensions of) Linear Integer Arithmetic. An interpolating procedure for linear Diophantine equalities is outlined in [33]. A procedure for full Linear Integer Arithmetic based on a sequent calculus can be found in [11]. In [12], the procedure in [11] is extended to cope with the theory of arrays without extensionality by axiom instantiation and interpolation in the combination of Presburger Arithmetic and uninterpreted function. Quantifiers can occur in the interpolants returned by the procedure. Recently [16], we have proposed a quantifier-free interpolation solver for $\mathcal{AX}_{\texttt{diff}}$ when combined with integer difference logic over indexes.

7.3. **Conclusions and Future Work.** We believe that the procedure proposed in this paper is a significant step forward to make model-checking more widely applicable to programs whose properties depend crucially on the manipulations of arrays. To the best of our knowledge, in fact, our interpolation procedure is the first to compute quantifier-free interpolants for a natural variant of the theory of arrays with extensionality obtained by replacing the extensionality axiom with its Skolemization. This variant is 'natural' in the sense that it is sufficient to detect unsatisfiability of formulae as it is usually the case in standard model checking methods for infinite state systems.

Despite the work reported in this paper is a significant step forward in widening the scope of applicability of interpolation in model checking of array manipulating programs, we discuss some interesting directions for further work.

The implementation of the interpolating procedure proposed here is crucial for showing the practical viability of our approach. In this respect, the first step is to implement the satisfiability solver in Section 5. Recall that this requires guessing, a pre-processing phase, and Gaussian completion phase. Guessing, as already observed in Section 7.1 item **(i)** when discussing the relationship with the solver of [41], can be implemented by adapting the mechanism to handle arrangements when combining satisfiability procedures in the Delayed Theory Combination approach of [9]. The main advantage of this approach is to use state-of-the-art SAT techniques to efficiently enumerate all possible partitions of indexes. The pre-processing phase can be implemented by using the data structures and basic expression manipulating procedures available in many state-of-the-art SMT solvers. The Gaussian completion phase requires more effort but it can adapt and reuse well-known techniques developed in rewriting for completion procedures (see, e.g., [4]). The second step to build the interpolating procedure of Section 6 is to implement the interpolating metarules of Table 1. This is relatively simple and does not require much ingenuity and can be done on top of the existing infrastructure for proof generation that is available in many state-of-the-art SMT solvers.

We are currently developing an implementation of the procedure presented here in the SMT-solver OpenSMT [18]. Preliminary experiments are encouraging although a more extensive experimental evaluation is needed. In fact, it is well-known that the convergence of interpolation based model checking procedures crucially depends on the "quality" of the computed interpolants. There have been attempts (see, e.g., [34, 49]) to build interpolating procedures that return "high quality" interpolants that guarantee the convergence of model checking for valid properties. Recently, it has been observed [26, 44] that a certain degree of flexibility for tuning the computation of interpolants in interpolation procedures would be desirable to facilate their integration in model checking. In this respect, it would be particularly interesting to investigate how the order in which the interpolating metarules of Table 1 are applied, particularly those on AB-common terms, may influence the "quality" of the interpolants. An interesting alternative to investigate the flexibility of generating interpolants (suggested in [44]) would be to use the procedure presented here in the framework for computing quantified interpolants of [44].

Finally, there are two more interesting points that deserve further investigations. First, it would be interesting to study the size of the interpolating metarules refutations and compare them with interpolating procedures based on a proof calculus. The preliminary experiments with our implementation of the procedure in Open SMT show that our refutations are quite compact but a more systematic comparison with available procedures based on a proof calculus, e.g., [47] is needed to clarify this issue. Second, since in model checking it is useful to compute interpolants for several partitions of the same (unsatisfiable) formula, it would be interesting to design a method that permit the partial reuse of the interpolants returned for a partition to compute the interpolant for the next one so as to permit reuse and avoid degradation of performances due to partial recomputation of parts of interpolating metarules refutation. In this respect, it seems possible to adapt techniques developed for computing chains of interpolants in [13].

Acknowledgements. We wish to thank two anonymous referees for their comments on a draft of this paper and an anonymous referee of RTA'11 for the criticisms that helped improving the presentation.

The work of the third author was partially supported by the "Automated Security Analysis of Identity and Access Management Systems (SIAM)" project funded by Provincia Autonoma di Trento in the context of the "team 2009 - Incoming" COFUND action of the European Commission (FP7) and the FP7-ICT-2007-1 Project no. 216471.

References

[1] A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.
[2] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. A rewriting approach to satisfiability procedures. *Inform. and Comput.*, 183(2):140–164, 2003. RTA 2001 (Utrecht).
[3] F. Baader, S. Ghilardi, and C. Tinelli. A new combination procedure for the word problem that generalizes fusion decidability results in modal logics. *Inform. and Comput.*, 204(10):1413–1452, 2006.
[4] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, Cambridge, 1998.
[5] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *J. Log. Comput.*, 4(3):217–247, 1994.
[6] L. Bachmair and A. Tiwari. Abstract Congruence Closure and Specializations. In *Conference on Automated Deduction, CADE '2000*, volume 1831 of *LNCS*, pages 64–78. Springer-Verlag, 2000.
[7] P. D. Bacsich. Amalgamation properties and interpolation theorems for equational theories. *Algebra Universalis*, 5:45–55, 1975.
[8] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodrguez-Carbonell, and A. Rubio. A Write-Based Solver for SAT Modulo the Theory of Arrays. In *FMCAD*, pages 101–108, 2008.
[9] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Van Rossum, S. Ranise, and R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In *CAV'05*, pages 335–349, 2005.
[10] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation.* Springer, 2007.
[11] A. Brillout, D. Kroening, P. Rümmer, and W. Thomas. An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic . In *IJCAR*, 2010.
[12] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Program Verification via Craig Interpolation for Presburger Arithmetic with Arrays. In *Verification Workshop at FLoC*, 2010.
[13] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. *Journal of Automated Reasoning*, 47:341–367, 2011.
[14] R. Brummayer and A. Biere. Lemmas on Demand for the Extensional Theory of Arrays. *JSAT*, 2009.
[15] R. Bruttomesso. *Problemi di combinazione nella dimostrazione automatica e nella verifica del software.* Università degli Studi di Milano, 2004. Master Thesis.
[16] R. Bruttomesso, S. Ghilardi, and S. Ranise. A Combination of Rewriting and Constraint Solving for the Quantifier-free Interpolation of Arrays with Integer Difference Constraints. In *FroCoS*, 2011.
[17] R. Bruttomesso, S. Ghilardi, and S. Ranise. Rewriting-based Quantifier-free Interpolation for a Theory of Arrays. In *RTA*, 2011.

[18] R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The OpenSMT Solver. In *TACAS*, pages 150–153, 2010.

[19] Roberto Bruttomesso, Silvio Ghilardi, and Silvio Ranise. From Strong Amalgamability to Modularity of Quantifier-Free Interpolation. Technical Report RI 337-12, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, 2012.

[20] C. Chang and J. H. Keisler. *Model Theory*. North-Holland, Amsterdam-London, third edition, 1990.

[21] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, pages 397–412, 2008.

[22] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolation Generation in Satisfiability Modulo Theories. *ACM Trans. Comput. Logic*, 12:1–54, 2010.

[23] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, pages 154–169, 2000.

[24] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, pages 269–285, 1957.

[25] L. de Moura and N. Bjørner. Generalized, Efficient Array Decision Procedures. In *FMCAD*, pages 45–52, 2009.

[26] V. D'Silva, M. Purandare, G. Weissenbacher, and D. Kroening. Interpolant Strength. In *Proceedings of VMCAI 2010*, volume 5944 of *LNCS*, pages 129–145. Springer, 2010.

[27] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.

[28] A. Fuchs, A. Goel, J. Grundy, S. Krstić, and C. Tinelli. Ground Interpolation for the Theory of Equality. In *TACAS*, pages 413–427, 2009.

[29] V. Ganesh and D. L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In *CAV*, pages 519–531, 2007.

[30] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision procedures for extensions of the theory of arrays. *Annals of Mathematics and Artificial Intelligence*, 50:231–254, 2007.

[31] A. Goel, S. Krstić, and A. Fuchs. Deciding Array Formulas with Frugal Axiom Instantiation. In *SMT*, 2008.

[32] W. Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1993.

[33] H. Jain, E. Clarke, and O. Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. *Form. Methods Syst. Des.*, 35(1):6–39, 2009.

[34] R. Jhala and K. L. McMillan. A Practical and Complete Approach to Predicate Refinement. In *TACAS*, pages 459–473, 2006.

[35] R. Jhala and K. L. McMillan. Array Abstractions from Proofs. In *CAV*, pages 193–206, 2007.

[36] D. Kapur. Shostak's Congruence Closure as Completion. In *8th Int. Conf. on Rewriting Techniques and Applications, volume 1232 of LNCS*, pages 23–37. Springer-Verlag, 1997.

[37] D. Kapur, R. Majumdar, and C. Zarba. Interpolation for Data Structures. In *SIGSOFT'06/FSE-14*, pages 105–116, 2006.

[38] D. Kapur and C. G. Zarba. A reduction approach to decision procedures. Technical report, Computer Science Dep., University of New Mexico, USA, 2005.

[39] L. Kovács and A. Voronkov. Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. In *FASE*, pages 470–485, 2009.

[40] J. Krajícek. Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.

[41] J. Levitt. *Formal Verification Thechniques for Digital Systems*. PhD thesis, Department of Computer Science, Stanford University, 1996.

[42] C. Lynch and Y. Tang. Interpolants for Linear Arithmetic in SMT. In *ATVA*, LNCS, 2010.

[43] J. McCarthy. Towards a Mathematical Science of Computation. In *IFIP Congress*, pages 21–28, 1962.

[44] K. McMillan. Interpolants from Z3 proofs. In *Proc. of FMCAD*, 2011.

[45] K. L. McMillan. Interpolation and SAT-Based Model Checking. In *CAV*, pages 1–13, 2003.

[46] K. L. McMillan. An Interpolating Theorem Prover. In *TACAS*, pages 16–30, 2004.

[47] K. L. McMillan. An Interpolating Theorem Prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[48] K. L. McMillan. Applications of Craig Interpolation to Model Checking. In *TACAS*, pages 1–12, 2005.

[49] K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, pages 413–427, 2008.

[50] P. Pudlák. Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[51] S. Ranise, C. Ringeissen, and D. Tran. Nelson-Oppen, Shostak and the Extended Canonizer: A Family Picture with a Newborn. In *ICTAC*, pages 372–386, 2004.

[52] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint Solving for Interpolation. In *VMCAI*, 2007.

[53] H. Saidi and S. Graf. Construction of abstract state graphs with PVS. In *CAV*, pages 72–83, 1997.

[54] V. Sofronie-Stokkermans. Interpolation in Local Theory Extensions. In *IJCAR'06: Int. Conf. on Automated Reasoning*, volume 4130 of *LNCS*, pages 235–250, 2006.

[55] A. Stump, C. Barrett, D. Dill, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *IEEE Symposium on Logic in Computer Science*, 2001.

[56] G. Yorsh and M. Musuvathi. A Combination Method for Generating Interpolants. In *CADE*, pages 353–368, 2005.

Appendix A. Proof of Theorem 2.3

Theorem 2.3 [7] *Let T be universal. Then, T admits quantifier-free interpolation iff T has the amalgamation property.*

Proof. Suppose first that T has amalgamation; let A, B be quantifier-free formulae such that $A \wedge B$ is not T-satisfiable. Let us replace variables with free constants in A, B; let us call Σ^A the signature Σ of T expanded with the free constants from A and Σ^B the signature Σ expanded with the free constants from B (we put $\Sigma^C \equiv \Sigma^A \cap \Sigma^B$). For reductio, suppose that there is no ground formula C such that: (a) A T-entails C; (b) $C \wedge B$ is T-unsatisfiable; (c) only free constants from Σ^C occur in C.

As a first step, we build a maximal T-consistent set Γ of ground Σ^A-formulae and a maximal T-consistent set Δ of ground Σ^B-formulae such that $A \in \Gamma$, $B \in \Delta$, and $\Gamma \cap \Sigma^C = \Delta \cap \Sigma^C$.[10] For simplicity[11] let us assume that Σ is at most countable, so that we can fix two enumerations

$$A_1, A_2, \ldots \qquad B_1, B_2, \ldots$$

of ground Σ^A- and Σ^B-formulae, respectively. We build inductively Γ_n, Δ_n such that for every n (i) Γ_n contains either A_n or $\neg A_n$; (ii) Δ_n contains either B_n or $\neg B_n$; (iii) there is no ground Σ^C-formula C such that $\Gamma_n \cup \{\neg C\}$ and $\Delta_n \cup \{C\}$ are not T-consistent. Once this is done, we can get our Γ, Δ as $\Gamma := \bigcup \Gamma_n$ and $\Delta := \bigcup \Delta_n$.

We let Γ_0 be $\{A\}$ and Δ_0 be $\{B\}$ (notice that (iii) holds by (a)-(b)-(c) above). To build Γ_{n+1} we have two possibilities, namely $\Gamma_n \cup \{A_n\}$ and $\Gamma_n \cup \{\neg A_n\}$. Suppose they are both unsuitable because there are $C_1, C_2 \in \Sigma^C$ such that the sets

$$\Gamma_n \cup \{A_n, \neg C_1\}, \quad \Delta_n \cup \{C_1\}, \quad \Gamma_n \cup \{\neg A_n, \neg C_2\}, \quad \Delta_n \cup \{C_2\}$$

are all T-inconsistent. If we put $C := C_1 \vee C_2$, we get that $\Gamma_n \cup \{\neg C\}$ and $\Delta_n \cup \{C\}$ are not T-consistent, contrary to induction hypothesis. A similar argument shows that we can also build Δ_n.

Let now $\mathcal{M}_1$ be a model of Γ and $\mathcal{M}_2$ be a model of Δ. Consider the substructures $\mathcal{N}_1, \mathcal{N}_2$ of $\mathcal{M}_1, \mathcal{M}_2$ generated by the interpretations of the constants from Σ^C: since the related diagrams are the same (because $\Gamma \cap \Sigma^C = \Delta \cap \Sigma^C$), we have that $\mathcal{N}_1$ and $\mathcal{N}_2$ are Σ_C-isomorphic. Up to renaming, we can suppose that $\mathcal{N}_1$ and $\mathcal{N}_2$ are just the same substructure (let us call it $\mathcal{N}$ for short). Since the theory T is universal and truth of universal sentences is preserved by substructures, we have that $\mathcal{N}$ is a model of T. By the amalgamation property, there is a T-amalgam $\mathcal{M}$ of $\mathcal{M}_1$ and $\mathcal{M}_2$ over $\mathcal{N}$. Now A, B are ground formulae true in $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, hence they are both true in $\mathcal{M}$, which is impossible because $A \wedge B$ was assumed to be T-inconsistent.

Suppose now that T has quantifier free interpolants. Take two models $\mathcal{M}_1 = (M_1, \mathcal{I}_1)$ and $\mathcal{M}_2 = (M_2, \mathcal{I}_2)$ of T sharing a substructure $\mathcal{N} = (N, \mathcal{J})$. In order to show that a T-amalgam of $\mathcal{M}_1, \mathcal{M}_2$ over $\mathcal{N}$ exists, it is sufficient (by Robinson Diagram Lemma 2.2) to show that $\delta_{\mathcal{M}_1}(M_1) \cup \delta_{\mathcal{M}_2}(M_2)$ is T-consistent. If it is not, by the compactness theorem of first order logic, there exist a $\Sigma \cup M_1$-ground sentence A and a $\Sigma \cup M_2$-ground sentence B

[10]By abuse, we use Σ^C to indicate not only the signature Σ^C but also the set of formulae in the signature Σ^C.

[11]This is just to avoid a (straightforward indeed) transfinite induction argument.

such that (i) $A \wedge B$ is T-inconsistent; (ii) A is a conjunction of literals from $\delta_{\mathcal{M}_1}(M_1)$; (iii) B is a conjunction of literals from $\delta_{\mathcal{M}_2}(M_2)$. By the existence of quantifier-free interpolants, taking free constants instead of variables, we get that there exists a ground $\Sigma \cup N$-sentence C such that A T-entails C and $B \wedge C$ is T-inconsistent. The former fact yields that C is true in $\mathcal{M}_1$ and hence also in $\mathcal{N}$ and in $\mathcal{M}_2$, because C is ground. However, the fact that C is true in $\mathcal{M}_2$ contradicts the fact that $B \wedge C$ is T-inconsistent. $\square$