What's Decidable About Arrays With Sums?

Roland Herrmann[1]✉[0009−0003−5748−7921]`roland.herrmann@ur.de` and Philipp Rümmer[1,2][0000−0002−2733−7098]`philipp.ruemmer@ur.de`

[1] University of Regensburg, Germany
[2] Uppsala University, Sweden

Abstract. The theory of arrays, supported by virtually all SMT solvers, is one of the most important theories in program verification. The standard theory of arrays, which provides *read* and *write* operations, has been extended in various different ways in past research, among others, by adding extensionality, constant arrays, function mapping (resulting in combinatorial array logic), counting, and projections. This paper studies array theories extended with *sum constraints,* which capture properties of the sum of all elements of an integer array. The paper shows that the theory of extensional arrays extended with constant arrays and sum constraints can be decided in non-deterministic polynomial time. The decision procedure works both for finite and infinite index sorts, as long as the cardinality is fixed a priori. In contrast, adding sum constraints to combinatorial array logic gives rise to an undecidable theory. The paper concludes by studying several fragments in between standard arrays with sums and combinatorial arrays with sums, aiming at providing a complete characterization of decidable and undecidable fragments.

1 Introduction

Arrays are fundamental in Computer Science because they provide a structured way to store and manage multiple values of the same type efficiently. They allow fast access to elements through indexing, enabling optimization of algorithms in terms of both time and space complexity. Reasoning about arrays is therefore a crucial part of verification of programs, and the theory of arrays is one of the most important theories supported by SMT solvers. The standard theory of arrays, proposed by McCarthy [10], has been extended over the years in various directions; among others, by adding extensionality, constant arrays and function mapping (resulting in combinatorial array logic CAL [11]), counting [5], or projections [3], aiming at a theory in which programs and specifications can be concisely encoded and efficiently analyzed.

In this paper, we consider array theories extended with *sum constraints,* which capture properties of the sum of elements in an integer array. Sum constraints are frequently used in specifications and are made available in specification languages like JML [7] and ACSL [2] in the form of *extended quantifiers,* yet are currently not supported by any SMT solver. In verification tools, sum constraints are therefore handled through quantified axioms [8] or intricate encodings [1] that are passed to the SMT solver. Exploring the possibility to support sum constraints directly in an SMT solver, we survey known decidability results for arrays extended with sums. We provide a simplified proof that Combinatorial Array Logic extended with sum constraints is undecidable. We then propose a new decision procedure for extensional arrays extended with sum constraints, that operates for both finite and infinite index sets uniformly, as long as their cardinality is fixed a priori. We overcome the interaction between the residual sums and the default values arising from constant arrays uniformly. The decision procedure is presented in a sequential manner, grouped into a preparation phase, simple rule applications, and rewriting of the sum constraints. This structure makes the procedure particularly comprehensible and easy to reason about.

Contributions of the paper: (1) We formalize our view of sum constraints and prove that the extension of CAL by sum constraints is undecidable in Section 4. (2) We survey results from the literature about arrays with sums in Section 5. (3) Our main contribution is a decision procedure for the satisfiability problem of quantifier-free formulae of extensional arrays with constant arrays and sums in Section 6, which runs in non-deterministic polynomial time. (4) We examine further extensions of our fragment on decidability in Section 7. Overall, this results in an overview of decidability and undecidability results and the problems that arise when we add sum constraints, serving both as a reference and starting point for ongoing research.

2 Related Work

A basic theory of arrays was introduced by McCarthy [10]. The satisfiability problem of quantifier-free formulae in an (extensional or non-extensional) array theory is NP-complete [15]. Our work uses the results and the framework of Combinatory Array Logic (CAL) [11] as a starting point, which extends extensional array theory by adding a constant array operator and element-wise function applications. An extension to CAL is Cartesian Array Logic (CaAL) [3], which supports n-ary arrays and adds a projection operator that corresponds to partial function application; adding projections to the theory comes at the cost of NEXPTIME-complexity of the corresponding satisfiability problem. All these theories have in common that the operators used in the constraints are designed to extract single elements of the array or act on an array uniformly. In contrast, an extension to statements specific to arrays as a whole is provided by the Array Folds Logic (AFL) [5], which enables counting elements in an array. However, AFL can not reason about the sum of all elements in an array. The projection operator in CaAL can only model finite sums (see Example 1). The work of Rodrigo Raya [13] is, to the best of our knowledge, the only decision procedure supporting summation of elements for infinite arrays. However, the results from the summation process are solely used to reason about cardinality constraints on index sets, and can neither be stored in another array nor compared to other array elements. The details are discussed in Section 5.

3 Preliminaries

We give some preliminaries and recollect the notions of the array theories that we refer to later. For foundations on logic we refer to [6]. First, we formally define the notion of a theory.

Definition 1 (Signature). *Let Σ^S be a set of sorts, Σ^F a set of function symbols and Σ^P a set of predicate symbols, where each $f \in \Sigma^F$, respectively $p \in \Sigma^P$ is endowed with a corresponding arity and sorts from Σ^S. Then we call the triple $\Sigma = (\Sigma^S, \Sigma^F, \Sigma^P)$ a signature.*

Definition 2 (Theory). *Let $\Sigma = (\Sigma^S, \Sigma^F, \Sigma^P)$ be a signature. A first-order formula over Σ is called Σ-sentence if it has no free variables. A Σ-theory is a signature Σ together with a collection of Σ-sentences closed under entailment.*

Axioms of particular interest are those that make a statement about the cardinality of our theory.

Definition 3 (cardinality axioms). *Let $n \in \mathbb{N}$. We define the following axioms (independent of the underlying signature) for theories with equality:*

$$\exists x_1, \dots, x_n. \bigwedge_{(i,j)\in\{1,\dots,n\}^2, i\neq j} x_i \neq x_j \qquad (\geq n) \qquad (1)$$

$$\forall x_1, \dots, x_n. \bigvee_{(i,j)\in\{1,\dots,n\}^2, i\neq j} x_i = x_j \qquad (< n) \qquad (2)$$

We say a theory

- *has cardinality at least n if it includes the axiom $(\geq n)$.*
- *has cardinality less than n if it includes the axiom $(< n)$.*
- *has cardinality n if it includes $(\geq n)$ and $(< n+1)$.*
- *is bounded if there exists an $n \in \mathbb{N}$ such that it includes $(< n)$.*
- *is unbounded if it includes the axioms $(\geq n)$ for all $n \in \mathbb{N}$.*
- *has fixed cardinality if it is either infinite or there exists an $n \in \mathbb{N}$ such that it has cardinality n.*

We describe array theories as conservative extensions of some base theory.

Definition 4 (conservative extension). *Let T_i be a theory with signature Σ_i and axioms A_i for $i \in \{1, 2\}$. Then T_2 is called a conservative extension of T_1 if (1) $\Sigma_1 \subseteq \Sigma_2$, and (2) formulae ϕ definable in T_1 are entailed by A_1 iff they are entailed by A_2, i.e., $A_1 \models \phi$ iff $A_2 \models \phi$.*

We assume a base theory with satisfiability problem of quantifier-free formulae in NP. For our purposes, arrays are introduced by adding a new sort denoted by $I \Rightarrow E$, for two sorts I and E, where we call I the index sort and E the element sort. Nested array sorts are allowed. The function symbols, predicate symbols, and axioms depend on the array theory we want to consider. In any case, they are conservative extensions of our base theory. The most basic array theory, also known as McCarthy arrays, can be defined as follows.

Definition 5. *Let T be our base theory with signature $(\Sigma^S, \Sigma^F, \Sigma^P)$ and corresponding axioms. Then the associated array theory is the following extension.*

- *Σ^S is extended to Σ^S_A, which is the least set such that $\Sigma^S \subseteq \Sigma^S_A$ and for all $I, E \in \Sigma^S_A$ we have $(I \Rightarrow E) \in \Sigma^S_A$.*
- *for all sorts $I, E \in \Sigma^S_A$, the set Σ^F is extended by an operator $read(\cdot,\cdot) : ((I \Rightarrow E), I) \to E$, sometimes also called select, and an operator $write(\cdot,\cdot,\cdot) : ((I \Rightarrow E), I, E) \to (I \Rightarrow E)$, sometimes called store. We abbreviate $read(a, i)$ by $a[i]$.*

The following axioms, representing the semantics of read and write, are added to extend our base theory to the theory of McCarthy arrays:

$$\forall a : (I \Rightarrow E), i : I, v : E. \qquad write(a, i, v)[i] = v \tag{3}$$

$$\forall a : (I \Rightarrow E), i : I, j : I, v : E. \qquad i = j \lor write(a, i, v)[j] = a[j]. \tag{4}$$

The above extension is indeed conservative. As a next extension, we consider extensionality of arrays, that is, we can reason about equality of arrays as a whole. Therefore, we add the following axioms:

$$\forall a : (I \Rightarrow E), b : (I \Rightarrow E).\ \exists i : I. \qquad a = b \lor a[i] \neq b[i]. \tag{5}$$

We call the resulting theory the extensional array theory. If the satisfiability problem of the base theory is in NP, then the satisfiability problem of a formula in the corresponding extensional array theory is in NP, too [15, Theorem 3]. To show NP membership, we can abstract away the array symbols and end up with an equi-satisfiable formula in the base theory.

For the rest of the paper, we assume that the quantifier-free theory of integers without multiplication, called Linear Integer Arithmetic (LIA) (we omit the prefix "quantifier-free" in the following), is included in the base theory, and we assume a "core solver" that decides satisfiability of quantifier-free formulae in the base theory in non-deterministic polynomial time. Note that LIA can be reduced in polynomial time to Presburger Arithmetic (PA), that is, the theory of natural numbers without multiplication, which is NP-complete.

Closely related to summation is the star operator introduced in [12].

Remark 1 (LIA).* For later use, we introduce the notion of a star operator on sets of integer vectors. Let $S \subseteq \mathbb{Z}^n$ be a set of integer vectors, then we define

$$S^* := \left\{ \sum_{i=1}^{n} \boldsymbol{x_i} \;\middle|\; n \in \mathbb{N} \text{ and } \boldsymbol{x_1}, \ldots, \boldsymbol{x_n} \in S \right\}. \tag{6}$$

Then LIA^* is the theory of Linear Integer Arithmetic extended by constraints of the form $\boldsymbol{x} \in \{\boldsymbol{z} \mid F(\boldsymbol{z})\}^*$, where $\boldsymbol{x}$ is a vector of integer terms and $\boldsymbol{z}$ a vector of integer variables, both of size $n \in \mathbb{N}$, and F is a LIA formula in which the variables $\boldsymbol{z}$ can occur. In [12] it is shown that the satisfiability problem of LIA^* formulae is NP-complete.

Another important extension of extensional arrays is Combinatorial Array Logic (CAL), which adds constant array constructors $K(\cdot) : E \to (I \Rightarrow E)$ and pointwise function applications $map_f : (I \Rightarrow E)^n \to (I \Rightarrow E)$ as new function symbols to the signature, where $f : E^n \to E$ is a function in the base theory. The semantics of the new functions are defined by the following axioms:

$$\forall v : E, i : I. \qquad K(v)[i] = v. \tag{7}$$

$$\forall a_1, \ldots, a_n : (I \Rightarrow E), i : I. \quad map_f(a_1, \ldots, a_n)[i] = f(a_1[i], \ldots, a_n[i]) \tag{8}$$

The satisfiability problem of quantifier-free CAL formulae is NP-complete as well. The decision procedure provided in [11] follows the same idea as those for the theory of extensional arrays: one carries reads through map_f terms and adds one additional read for every array, representing all values that are not constrained, to detect contradictions arising from the constant array operator.

4 Sum Constraints

We formalize what we mean by a sum constraint, which is a constraint on an array that refers to the sum of all array elements. The difficulty with sums is that we make a statement about all elements in the array, of which there may be infinitely many. There are two aspects we need to take into account: whether the sum exists, and, if it does, what its value is. Finite sums always exist, while the sum of infinite arrays only exists if almost all array elements have value zero. To address this undefinedness, we introduce a new relation symbol $sum : ((I \Rightarrow \mathbb{Z}), \mathbb{Z})$, which intuitively holds for an array a and an integer k if the sum of the elements of a exists and is equal to k.

The semantics of sum is defined by two axiom schemata. Evidently the axiom only applies to arrays with $\mathbb{Z}$ as element sort. If an array has only finitely many non-zero elements, the sum exists, which is expressed by the following axiom schema, where $distinct(i_1, \ldots, i_n)$ evaluates to true if and only if $i_1, \ldots, i_n$ are pairwise distinct:

$$\begin{aligned}&\forall i_1, \ldots i_n : I, v_1, \ldots, v_n, w : \mathbb{Z}.\ distinct(i_1, \ldots, i_n) \to \\ &\quad \left(sum(write(\ldots(write(K(0), i_1, v_1) \ldots i_n, v_n), w) \leftrightarrow w = \sum_{i=1}^{n} v_i \right)\end{aligned} \tag{9}$$

We used the fact that an array which is zero almost everywhere arises from a finite number of write operators from $K(0)$. However, the number of non-zero elements is not known in general, which results in infinitely many axioms, one for each $n \in \mathbb{N}$.

As finiteness cannot be expressed in first-order logic, as a consequence of the compactness Theorem [6, Theorem 4.6], we use an axiom schema in second-order logic to describe the cases in which a sum is not defined. We write *injective*(f) for the formula expressing injectivity of f:

$$\forall a : (I \Rightarrow \mathbb{Z}), v : \mathbb{Z}. \\ \Big(\exists f : \mathbb{Z} \to I. \big(\mathit{injective}(f) \land \forall n : \mathbb{Z}.\ a[f(n)] \neq 0\big)\Big) \to \neg \mathit{sum}(a, v) \tag{10}$$

Definition 6. *Let T be a base theory including* LIA*, where all sorts have fixed cardinality. Then we define Summation Array Logic (*SAL*) as the extensional array theory extended by the constant array operator and the sum predicate and their corresponding axioms.*

With our description of sums, we obtain the following undecidability result. The result is related to the one given in [13], which however holds for a richer array theory including Boolean algebra expressions over index sets (see Section 5.2).

Theorem 1. *Consider* CAL *extended by sum constraints and $\mathbb{Z}$ with Linear Integer Arithmetic as index and element sort. Then satisfiability of quantifier-free formulae in the resulting theory is undecidable.*

Proof. We prove this by reduction from Hilbert's tenth problem [9], which states that the problem whether diophantine equations have a solution is undecidable. To this end, it suffices to show that we can express equations of the form $z = x \cdot y$ for integer variables $x, y, z \in \mathbb{Z}$. The function $f : \mathbb{Z} \to \mathbb{Z}$, which evaluates to 1 if its argument is greater or equal to zero, and to zero otherwise, can be defined in LIA by the formula $(x_1 < 0 \land x_2 = 0) \lor (x_1 \geq 0 \land x_2 = 1)$. Furthermore, the function $g : \mathbb{Z}^2 \to \mathbb{Z}$ with $g(x_1, x_2) = x_2$ if $x_1 = 1$ and $g(x_1, x_2) = 0$ otherwise, can be defined in LIA by the formula $(x_3 = x_2 \land x_1 = 1) \lor (x_3 = 0 \land x_1 \neq 1)$. Hence, we can define the following formula in CAL.

$$b = \mathit{map}_f(a) \land \mathit{sum}(b, x) \land c = \mathit{map}_g(b, K(y)) \land \mathit{sum}(c, z). \tag{11}$$

The first constraint requires b to have values 0 or 1, since f has those as its image. Together with the second constraint on the sum of b, b necessarily has x entries with 1 and the rest with 0 in order to be satisfiable. Finally, the application of g to b and $K(y)$ sets all non-zero values of b to y and the rest to 0, which enforces b to have exactly x entries with the value y, i.e. $\mathit{sum}(c, z) \Leftrightarrow z = \sum_{i=1}^{x} y = x \cdot y$.

5 Arrays With Sums in the Literature

5.1 Cardinality of the Index Sort

Consider any array theory with finite (bounded) index sorts. If we add sum constraints, we obviously maintain decidability: we may simply impose reads on all array indices. Then we can replace the sum operator by the actual formula computing the sum, as in axiom 9. This transforms any formula with sum constraints to an ordinary formula in the theory without sum constraints. However, this will lead to a $\mathcal{O}(|I|)$ blowup of the formula for the imposed reads, where I is the index sort.

Lemma 1. *Let T be an array theory whose satisfiability problem is decidable and whose index sorts are bounded. Then the theory extended by sum constraints is decidable as well.*

Another procedure that is built on CAL is the Cartesian Array Logic (CaAL) [3]. It is introduced to represent quantum states as multi-dimensional arrays of the form $\{0,1\}^n \Rightarrow \mathbb{C}$, where $\mathbb{C}$ denotes the complex numbers. In this logic, a projection operator is available, which (as a special case) can model sum constraints, as shown by the following example.

Example 1. Let $proj_k(a, i)$ denote the projection operator from CaAL to the array $a : \{0,1\}^n \Rightarrow \mathbb{C}$, which is the projection of the k-th index variable to the value i, defined as follows using notation for multidimensional arrays:

$$proj_k(a,i)[i_0,\dots,i_{n-1}] = a[i_0,\dots,\underbrace{\quad i \quad}_{k\text{-th position}},\dots,i_{n-1}]. \tag{12}$$

Now consider $a : \{0,1\}^n \Rightarrow \mathbb{C}$. Then the sum s of a can be calculated by the following formula, where map_+ denotes element-wise addition of arrays:

$$a_0 = a \wedge s = a_n[\,] \wedge \bigwedge_{i=1}^{n} a_i = map_+(proj_0(a_{i-1},0), proj_0(a_{i-1},1)) \tag{13}$$

Although the size of this formula is only linear in n, solving the formula boils down to constructing a number of read operations that is exponential in n; in the end, a decision procedure has to construct reads for all locations of the array a. This is an explanation for why the decision procedure in [3] runs in non-deterministic exponential time in general. The construction of the sum using the projection operator is not possible for infinite index sorts in CaAL. Hence, the proof of Theorem 1 does not carry over to the case of CaAL.

In order to handle arrays with infinite index sort, it is necessary that we have information about all elements in the array, even if they do not explicitly occur in a constraint. In the decision procedure for CAL [11], a fresh index variable is introduced to represent the default value. However, this is only possible because the index sort has infinite cardinality. Some changes in the decision procedure for CAL are necessary to support finite index sorts [11].

5.2 Other Sum Constraints

A different definition of a sum constraint is used by Rodrigo Raya in [13]. In [14], Raya et al. describe a satisfiability preserving translation from CAL to quantifier-free Boolean algebra formulae with Presburger Arithmetic and interpreted index sets (QFBAPAI). A formula in QFBAPAI has the following form [14, Def. 5]

$$F(S_1, \dots, S_k) \wedge \bigwedge_{i=1}^{k} S_i = \{r \in I \mid \varphi_i(x_1[r], \dots, x_n[r], c_1, \dots, c_m)\}, \tag{14}$$

where F is a Boolean algebra formula with Linear Integer Arithmetic constraints on the cardinality of the set variables $S_1, \dots, S_k$, φ_i are formulae in the existential fragment of the element theory, I is the index sort of the arrays, $x_1, \dots, x_n$ are array variables and $c_1, \dots, c_m$ are variables of the element sort. Without going into further detail, the idea behind the transformation is to abstract away the constraints in the element theory to constraints on index sets, for example a read constraint $v = \mathit{read}(a, i)$ can be equivalently described by $\{i\} \subseteq \{l \mid a[l] = v\}$ as part of the formula F. In [13], sum constraints are added to this logic which results in formulae of the following form

$$F(S_1, \dots, S_k, \boldsymbol{\sigma}) \wedge \bigwedge_{i=1}^{k} S_i = \{n \in I \mid \varphi_i(\boldsymbol{c}[n])\} \wedge \boldsymbol{\sigma} = \sum (\!| \boldsymbol{c}[n] \mid \varphi_0(\boldsymbol{c}[n]) |\!), \tag{15}$$

where $\boldsymbol{c}$ is a tuple of arrays and $(\!| \cdot |\!)$ denotes multi-sets. First note that this does not allow variables in the element sort, which would result in undecidability similar to Theorem 1 and is stated in [13, Corollary 6]. The summation process has the same intended semantics as ours in Section 4.

Example 2. A minimal example of a sum in QFBAPAI with sums is the following formula

$$|S| < \sigma \wedge S = \{n \in I \mid \varphi(c[n])\} \wedge \sigma = \sum (\!| c[n] \mid \varphi_0(c[n]) |\!). \tag{16}$$

The formula φ specifies the set of indices included in S, depending on the value of the array c at the index. Formula φ could for example select those indices where the value at this index is even: $\varphi(x) \equiv \exists y.\ 2 \cdot y = x$. Formula φ_0 restricts the values of the array that add to the sum. To obtain the sum over the whole array we simply choose $\varphi_0 \equiv \mathit{true}$. Then the original formula translates to "the number of indices where an even value is stored is smaller than the sum over all values of the array". This is satisfiable for the array $K(0)$. No matter how we choose φ and φ_0, the resulting formula will always make a statement on the cardinality of an index set in relation to the (partial) sum of an array.

In comparison, formulas in our theory of arrays with sums can take the form $\mathit{sum}(a, v) \wedge v = \mathit{read}(b, i)$. The sum is used to constrain elements of the b. In particular, we can store the sum of one array in another array.

6 Array Theory With Constant Arrays and Sums

In this section, we provide a decision procedure which runs in non-deterministic polynomial time for the array theory with sums defined in definition 6. Note that we allow finite and infinite index sorts as long as they have fixed cardinality. Nested arrays are also allowed.

Remark 2 (flattened form). We assume that our formulae are in flattened form, meaning that the left-hand side of every equality is a variable. For readability, however, we will initially write literals in non-flattened form and immediately transform them afterwards. Any formula can be converted into flattened form in linear time by adding at most two fresh variables per atom, splitting the atom into three atoms, for instance $x+1 = y-2$ can be transformed to $a = x+1 \wedge b = y-2 \wedge a = b$ with fresh variables a, b.

We end up with the following set of constraints we want to reason about.

Definition 7 (Arr_{cs}). *Let Arr_{cs} be the set of all sets which consist of constraints of the following form.*

$$\begin{array}{ll}
v = \mathit{read}(a,i) & (\mathit{read}) \\
a = \mathit{write}(b,i,v) & (\mathit{write}) \\
a = b, a \neq b & (\mathit{(in)equality\ of\ arrays}) \\
x = y, x \neq y & (\mathit{(in)equality\ of\ variables}) \\
a = K(v) & (\mathit{const}) \\
\mathit{sum}(a,v) & (\mathit{sum\ predicate}) \\
p(v_1,\dots,v_n) & (\mathit{base\ theory\ predicate})
\end{array}$$

where p is a predicate in the base theory, not involving array variables. We say that an element $\varphi \in \mathit{Arr}_{cs}$ is satisfiable if and only if $\bigwedge_{c\in\varphi} c$ is satisfiable.

For the time being, we do not allow constraints of the form $\neg \mathit{sum}(a,v)$ in the input formula. There are two possibilities: the sum does not exist, or the sum does not have the desired value. The latter case can be expressed by $\mathit{sum}(a,w) \wedge v \neq w$.

The decision procedure to determine satisfiability of an element $\varphi \in \mathsf{Arr}_{cs}$ is presented as a sequence $\varphi = F_{-3}, F_{-2}, F_{-1}, F_0, \dots, F_n$ in Arr_{cs} such that F_{-3} is satisfiable if and only if F_n is. The steps to derive F_{i+1} from F_i are described in the following. We start by an example to illustrate the procedure.

Example 3. Consider the following satisfiable set of constraints, where $i, j, x, v, w : \mathbb{Z}$ and $a, b : \mathbb{Z} \Rightarrow \mathbb{Z}$ in Arr_{cs}:

$$\varphi = \begin{Bmatrix} a = \mathit{write}(b,i,v), b = \mathit{write}(c,j,w), c = K(x), \\ \mathit{sum}(a,w), \mathit{sum}(b,12) \end{Bmatrix} \tag{17}$$

First, we non-deterministically guess an *arrangement,* defining which variables have equal and which have distinct values. One possible guess could be to assume that all occurring index variables are pairwise distinct, i.e. $\mathit{distinct}(i,j)$, and $a = b \neq c$ holds. It will turn out that this is indeed an arrangement that yields satisfiability of the formula. In order to treat the sum constraints later, we impose reads on all occurring index and array variables, plus one additional read on a fresh index i_a on a, representing the "residual" of the sum, i.e., the sum of array elements at indices distinct from i and j. If we abbreviate $r_d^k := (v_{d,k} = \mathit{read}(d,k))$, φ is equi-satisfiable to

$$\varphi \cup \{r_d^k \mid k \in \{i,j,i_a\}, d \in \{a,b,c\}\} \cup \{\mathit{distinct}(i,j,i_a), a = b, b \neq c\} \tag{18}$$

since adding additional variables and reads does not change satisfiability. Next, we can infer additional constraints that we add to our set of constraints by a set of inference rules derived from the axioms of our theory. The inference rule derived from the axiom for the constant array operator implies $v_{c,i} = v_{c,j} = v_{c,i_a} = x$. The inference rules derived from the axioms on read and write infer $v_{b,i} = v_{b,i_a} = x, v_{b,j} = w, v_{a,i_a} = x, v_{a,j} = w, v_{a,i} = v$, which can be illustrated as follows.

$$\begin{array}{r|c|c|c|c|c|c|c}
\cline{2-8}
a = & \cdots & i : v & \cdots & j : w & \cdots & i_a : x & \quad\text{x}\quad \\
\cline{2-8}
\multicolumn{8}{c}{} \\
\cline{2-8}
b = & \cdots & i : x & \cdots & j : w & \cdots & i_a : x & \quad\text{x}\quad \\
\cline{2-8}
\end{array} \tag{19}$$

Since a and b have an infinite index sort, in order to satisfy $sum(a, w)$, respectively $sum(b, v)$, a and b need to be zero almost everywhere. Since they arise from $K(x)$ they have the value x almost everywhere, in particular we can deduce $x = 0$. Note that this implies that the array has value zero everywhere except at i and j, and that the residual sum is zero. Finally, we can rewrite the sum constraints to $v + w + x = w$ and $x + w + x = 12$, since we assume that the residual of the sums is "compressed" at i_a. We can solve these equations algebraically and obtain $x = v = 0, w = 12$. A model of the arrays as functions $\mathbb{Z} \to \mathbb{Z}$ can be constructed by setting c to 0 everywhere, and setting $a = b$ to the function that returns 12 for input 0 and zero elsewhere.

The first steps from F_{-3} to F_0 summarize the introduction of all necessary fresh variables and reads. In the above example, the (simplified) set F_0 is (18).

Definition 8 (Indices for extensionality). *Let $\varphi \in \mathsf{Arr}_{cs}$. For each pair of array variables of the same sort $a, b : I \Rightarrow E$, let $k_{a,b}$ be a fresh index variable, $v^a_{a,b}, v^b_{a,b}$ fresh element variables, and define $F_{-2} := F_{-3} \cup \{v^a_{a,b} = read(a, k_{a,b}) \mid a, b : I \Rightarrow E \text{ occur in } F_{-3}\}$ as the formula with indices for extensionality.*

The variable $k_{a,b}$ serves as an index variable to verify the extensionality axiom in the case of inequality, which may eventually be identified with an already existing variable.

Definition 9 (Arrangement). *Let K be a set of variables of the same sort. An arrangement of K is a partition of K into equivalence classes, that is sets $K_1, \ldots, K_n \subseteq K$ such that $\dot{\bigcup} K_i = K$. Let $\{K_1, \ldots, K_n\}$ be an arrangement of K, then we define*

$$arr(\{K_1, \ldots, K_n\}) = \bigcup_{i=1}^{n} \bigcup_{a,b \in K_i} \{a = b\} \cup \bigcup_{i=1}^{n} \bigcup_{j=1}^{i-1} \bigcup_{a \in K_i, b \in K_j} \{a \neq b\}. \tag{20}$$

Definition 10 (Chosen arrangements). *Let F_{-2} be a formula with indices for extensionality. For all variables of each sort σ in F_2 including array sorts non-deterministically choose an arrangement $\mathcal{K}_\sigma$. Let $\delta_a : E$ be a fresh variable for each array $a : I \Rightarrow E$ in F_{-2} which we will call the default value. Then choose an arrangement $\mathcal{K}_{\sigma,\delta}$ for all default values of each sort σ in F_2 as well.*

For each sort σ in F_2, either include $\bot$ into one equivalence class of $\mathcal{K}_{\sigma,\delta}$ or add a new equivalence to $\mathcal{K}_{\sigma,\delta}$ which consists only of $\bot$. We define $F_{-1} := F_{-2} \cup \bigcup_\sigma arr(\mathcal{K}_\sigma) \cup arr(\mathcal{K}_{\sigma,\delta})$ as the formula with chosen arrangements.

The default values serve as auxiliary variables that indicate the value of an array at an index that is not explicitly mentioned in the formula through a read. For instance, the elements of a constant array $K(x)$ necessarily need to be x, hence the default value should be x. The equivalence class with $\bot$ indicates that the arrays within this class (which may also be none) do not necessarily have a default value. This allows us to choose the values at indices that are not explicitly mentioned in the formula as illustrated by the following example.

Example 4. Consider the following satisfiable set of constraints, where $i : \mathbb{Z}$ and $a, b : \mathbb{Z} \Rightarrow \mathbb{Z}$ in Arr_{cs}

$$\varphi = \{a = \mathit{write}(b, i, v), \mathit{read}(b, i) = 8, \mathit{sum}(a, 10), \mathit{sum}(b, 12)\}. \tag{21}$$

We first choose an arrangement; as we will see, in this example, the arrangement $a \neq b$ gives rise to a solution. In contrast to Example 3, there is no constant array involved. Hence, we may choose $\delta_a = \delta_b = \bot$ as arrangement for default values. For comparison: in Example 3, the only satisfying arrangement for default values is $\delta_a = \delta_b = \delta_c \neq \bot$. Clearly a and b need to be zero almost everywhere to satisfy the sum constraints. However, this does not imply that their respective default values need to be zero, which would cause all elements except at i to be zero. Here the role of $\bot$ comes into play: we are free to choose infinitely many elements as zero but there might exist finitely many non-zero elements at indices other than i. We assume that those, adding up to the residual sum of the arrays, are compressed at a fresh index i_a. Since a and b are equal at all indices other than i the residual sums coincide as well, which is ensured by inference rules instantiating the read and write axioms. Hence, it suffices to add the constraints $\mathit{read}(a, i) = v, \mathit{read}(a, i_a) = s, \mathit{read}(b, i_a) = s, i \neq i_a$ to φ, which can be illustrated as follows.

$$\begin{array}{r|c|c|c|c|c}
a = & 0 & i : v & 0 & i_a : s & 0 \\
\hline
b = & 0 & i : 8 & 0 & i_a : s & 0
\end{array} \tag{22}$$

Note that the element at index i is not included in the residual sum of a and therefore explicitly added to the formula since otherwise the respective residual sums of a and b do not coincide. The sum constraints can then be rewritten as $v + s = 10$ and $8 + s = 12$ which can be solved algebraically and yields the solution $s = 4, v = 6$. Clearly, many other solutions exist, for instance by constructing arrays that store at four distinct indices the element 1; but to find just one solution, we can assume that the only non-zero entries are at i and i_a.

Lemma 2. *Let $\varphi \in \mathsf{Arr}_{cs}$. Then φ is satisfiable if and only if there exist arrangements $\mathcal{K}_{\sigma_1}, \dots, \mathcal{K}_{\sigma_n}$ for all occurring variables and sorts $\sigma_1, \dots, \sigma_n$ in φ such that $\varphi \cup \bigcup_{i=1}^n arr(\mathcal{K}_{\sigma_i})$ is satisfiable.*

Remark 3. Intuitively, the default value of an array should be preserved through write operators, for instance $a = \mathit{store}(K(x), i, v)$ should have the value x everywhere except at i, i.e. $\delta_a = x = \delta_{K(x)}$. However, this is only true for infinite index sorts. Consider arrays $a, b : \{0\} \Rightarrow \mathbb{Z}$ and the following formula:

$$a = K(0) \wedge b = \mathit{write}(a, 0, 1) \wedge c = K(1) \wedge b = c \tag{23}$$

Any array that is constant should have a corresponding default value. Then $a = K(0)$ and $b = c = K(1)$ would imply $\delta_a = 0$ and $\delta_b = \delta_c = 1$. If we allow to propagate default values through write operators we obtain $0 = \delta_a = \delta_b = \delta_c = 1$, which is a contradiction. However, the formula is satisfiable with $a = K(0)$ and $b = c = K(1)$ which is due to the fact that all elements of the index sort are already explicitly mentioned in the formula. Therefore, we need to compare the number of constrained indices with the cardinality of the index sort.

Definition 11 (Index equivalence classes). *Let $a : (I \Rightarrow E)$ be an array variable occurring in a formula $\varphi \in \mathsf{Arr}_{cs}$. We write $[i]$ for the equivalence class of variable i according to an arrangement. Then $\mathit{ct}(a)$ is the least set of equivalence classes (according to an arrangement) of index variables that satisfies:*

- *for all constraints $v = \mathit{read}(a, i)$, we have $[i] \in \mathit{ct}(a)$;*
- *for all constraints $a = \mathit{write}(b, i, v)$, we have $\mathit{ct}(b) \subseteq \mathit{ct}(a)$ and $[i] \in \mathit{ct}(a)$;*
- *for all constraints $a = b$ we have $\mathit{ct}(b) = \mathit{ct}(a)$.*

Definition 12 (Weak equivalence [4]). *Let $\varphi \in \mathsf{Arr}_{cs}$. Two array variables a, b are called weakly equivalent if there exists a sequence of array variables $a = a_1, \ldots, a_n = b$ such that for all $i \in \{1, \ldots, n-1\}$ either $a_i = \mathit{write}(a_{i+1}, j, v) \in \varphi$, $a_{i+1} = \mathit{write}(a_i, j, v) \in \varphi$ for some j, v or $a_i = a_{i+1} \in \varphi$. We define $\mathit{WE}(a) := \{b \mid b \text{ is weakly equivalent to } a\}$.*

Definition 13 (Indices for residual sums). *Let F_{-1} be a formula with chosen arrangements. Let S' be the set of all array variables $a : I \Rightarrow \mathbb{Z}$ with $\mathit{sum}(a, v) \in F_{-1}$ for some v and $|\mathit{ct}(a)| < |I|$. Let $S \subseteq S'$ be a subset that contains exactly one array variable for each weak equivalence class in S'. For $a \in S$ define $\mathit{ct}(\mathit{WE}(a)) := \bigcup_{c \in \mathit{WE}(a)} \mathit{ct}(c)$. Then we define the formula with all necessary reads and indices for residual sums as*

$$F_0 := F_{-1} \cup \bigcup_{a \in S} \left(\begin{array}{l} \quad \{v = \mathit{read}(b, i) \mid b \in \mathit{WE}(a), [i] \in \mathit{ct}(\mathit{WE}(a))\} \\ \cup \{\varepsilon_a = \mathit{read}(a, i_a) \mid |\mathit{ct}(\mathit{WE}(a))| < |I|\} \\ \cup \{i \neq i_a \mid [i] \in \mathit{ct}(\mathit{WE}(a)), |\mathit{ct}(\mathit{WE}(a))| < |I|\} \end{array} \right) \tag{24}$$

Note that only one ε_a for each weak equivalence class suffices to represent the residual of the sum on the indices which are not constrained in the formula.

Deriving F_0 from F_{-3}, only a polynomial number of constraints and fresh variables have been added. The next steps can be described by a set of inference rules, similar to those in the decision procedure for CAL in [11]. The rules are depicted in Figure 1. They can be read top-down, i.e., the premise is the set of constraints above the line and the consequence is the set of constraints below the

$$\frac{a = \mathit{write}(b,i,v)}{a[i] = v}\ \mathit{idx} \qquad \frac{a = \mathit{write}(b,i,v) \quad w = a'[j] \quad a' = a \quad i \neq j}{w = b[j]}\ \Downarrow$$

$$\frac{a = K(v) \quad w = a'[j] \quad a' = a}{a[j] = v}\ K\Downarrow \qquad \frac{a = \mathit{write}(b,i,v) \quad w = b'[j] \quad b' = b \quad i \neq j}{w = a[j]}\ \Uparrow$$

$$\frac{a : (I \rightarrow E) \quad b : (I \rightarrow E) \quad a \neq b}{a[k_{a,b}] \neq b[k_{a,b}]}\ \mathit{ext} \qquad \frac{a = K(v)}{\delta_a = v}\ \delta_a\text{-}\mathit{const}$$

$$\frac{b = \mathit{write}(a,i,v) \quad \delta_a \neq \delta_b}{\bot}\ \delta_a\text{-}\mathit{write} \qquad \frac{a = b \quad \delta_a \neq \delta_b}{\bot}\ \delta_a\text{-}\mathit{equality}$$

$$\frac{x = y \quad f(x) \neq f(y)}{\bot}\ \mathit{cong}$$

Fig. 1. Inference rules

line. Premises $a' = a$ between array variables express that a rule is applicable if a', a coincide, or if those arrays are assumed to be equal in the arrangement. If $\phi \subseteq F_i$ and we have an inference rule with ϕ as the premise, then the consequence ψ can be added to F_i, i.e., $F_{i+1} = F_i \cup \psi$. The inference rules $\mathit{idx}, \Downarrow, \Uparrow, \mathit{ext}, K \Downarrow$ are adjusted versions of those in [11] for CAL and model the axioms. Rules δ_a-*const*, δ_a-*write* and δ_a-*equality* are introduced to handle the default values. We restrict the usage of δ_a-*write* and δ_a-*equality* to the case where $|ct(b)| < |I|$, as discussed in Remark 3. Rule *cong* ensures functional consistency for any function symbol f. We assume that the core solver returns unsatisfiable if $\bot$ occurs.

Lemma 3. *The inference rules δ_a-const, δ_a-write and δ_a-equality are sound in the following sense: if a formula $\varphi \in \mathsf{Arr}_{cs}$ is satisfiable then there exists an arrangement of the default values such that whenever F_i is satisfiable, then the set F_{i+1} obtained by applying one of the rules is satisfiable as well.*

Proof. Rule δ_a-*const* is sound since there are no axioms referring to the default values that could be violated after the rule application and it is already ensured by the axiom for the constant array constructor that if $a = K(v)$ and $a = K(w)$ then $v = w$, hence $\delta_a = v$ is well-defined. Assume for a contradiction that for all arrangements of the default values, there is an application of δ_a-*write*, respectively δ_a-*equality*, possible. The only way to obtain such a situation according to our rules is that both arrays a, b arise from a constant array along a sequence of write and equality constraints, otherwise we would be able to change the default values to make a rule application impossible. Therefore there exists $k \in I \setminus \bigcup ct(b) \neq \emptyset$ such that $\mathit{read}(a,k) = \delta_a \neq \delta_b = \mathit{read}(b,k)$, which is a contradiction to the axiom for read and write, respectively extensionality.

We apply the inference rules until we reach a fixed point F_k for some $k \in \mathbb{N}$. In order to handle the sum constraints, we use a set of rewrite rules.

$$\mathit{sum}(a,v) \rightsquigarrow_\infty \sum_{[i] \in ct(a)} \mathit{read}(a,i) = v \wedge \delta_a = 0 \tag{25}$$

$$sum(a,v) \rightsquigarrow_n \sum_{[i]\in ct(a)} read(a,i) = v \tag{26}$$

$$sum(a,v) \rightsquigarrow_n^{\delta_a} \sum_{[i]\in ct(a)} read(a,i) + (|I| - |ct(a)|)\cdot \delta_a = v \tag{27}$$

A rewrite rule $\phi \rightsquigarrow \psi$ applied to F_i results in $F_{i+1} = (F_i \setminus \phi) \cup \psi$. The first rewrite rule is restricted to $|I| = \infty$, the second one to $|I| < \infty$ and $\delta_a = \bot$ and the third one is restricted to $|I| < \infty$ and $\delta_a \neq \bot$.

Lemma 4. *Let $F_i \in \mathsf{Arr}_{cs}$ be the current set of constraints such that the application of inference rules has reached a fixed point. Then the rewrite rules on sum constraints preserve satisfiability of F_i.*

Proof. For $i_1, \ldots, i_{n+m}$ such that $distinct(i_1, \ldots, i_{n+m})$ holds, we have

$$\begin{aligned} &sum(write(\ldots(write(K(0), i_1, v_1)\ldots i_{n+m}, v_{n+m}), \sum_{i=1}^{n+m} v_i) \\ \Leftrightarrow\ &sum(write(\ldots(write(K(0), i_1, v_1)\ldots i_{n+1}, \sum_{i=n+1}^{n+m} v_i), \sum_{i=1}^{n+m} v_i). \end{aligned} \tag{28}$$

Therefore, if a sum constraint satisfies the axioms, then there exists a representation of the sum as rewritten that satisfies the formula. The residual of the sum is compressed at the fresh indices that may have been introduced previously before applying the inference rules. The soundness of $\rightsquigarrow_n^{\delta_a}$ follows from the fact that additional reads evaluate to the default value as in the proof of Lemma 3.

Theorem 2. *The following procedure is sound and complete for determining satisfiability of quantifier-free formulae in Arr_{cs}, i.e. there exists an arrangement such that the procedure returns satisfiable iff the original formula is satisfiable.*

1. *Step by step add the additional variables, reads and arrangements to subsequently obtain the sequence $F_{-3}, F_{-2}, F_{-1}, F_0$, as defined in Definitions 8, 10 and 13.*
2. *Apply the inference rules until a fixed point F_k is reached.*
3. *Apply the rewrite rules to all sum constraints to obtain F_{k+n}.*
4. *Let a core solver decide satisfiability of F_{k+n}.*

Proof. Soundness: Adding additional variables and reads does not change satisfiability. Adding arrangements is sound by Lemma 2. The inference rules are sound since they are direct instantiations of the axioms and by Lemma 3. The rewrite rules on sums are sound by Lemma 4. The last step is sound by assumption on our core solver.

Completeness: It suffices to show that any satisfiable model of the core solver in the last step gives rise to a model that satisfies the original formula.
Let M be a model provided by the core solver. We extend M to a model M^λ for our original formula as in [11]. Array variables are interpreted as functions.

The read, write and constant array operator have their expected interpretations, see [11] for details. We choose for every sort E a default value $\delta_E \in M^\lambda(E)$ and $\delta_\mathbb{Z} = 0$. Then we define the interpretation of an array variable $a : I \Rightarrow E$ as

$$M^\lambda(a)(\iota) = \begin{cases} M^\lambda(v) & \text{if } v = \mathit{read}(a,i) \text{ exists and } M^\lambda(i) = \iota \\ M^\lambda(\delta_a) & \text{else if } \delta_a \neq \bot \\ \delta_E & \text{else if } \delta_a = \bot \end{cases} \tag{29}$$

This interpretation is well-defined: Assume we have $v = \mathit{read}(a,i) \wedge w = \mathit{read}(a,j)$ with $i = j$. Then by the inference rule for congruence for the function symbol *read*, we obtain $M(v) = M(\mathit{read}(a,i)) = M(\mathit{read}(a,j))) = M(w)$, in particular $M^\lambda(a)(M^\lambda(i))$ is well-defined. We show that all axioms are satisfied.

- Write-axioms and Extensionality-axiom: We only show that the write axioms hold, the extensionality axiom works analogously. Assume we have $a = \mathit{write}(b,i,v)$. Then $M^\lambda(a)(M^\lambda(i)) = M^\lambda(v)$ is fulfilled by the inference rule *idx*. For all $\iota \in M^\lambda(\bigcup ct(b)) \setminus \{M^\lambda(i)\}$, we have $M^\lambda(a)(\iota) = M^\lambda(b)(\iota)$ by definition of the interpretation of arrays on $M^\lambda(\bigcup ct(b))$ and the inference rules $\Downarrow$ and $\Uparrow$. For all $\iota \in M^\lambda(I) \setminus M^\lambda(\bigcup ct(a))$ we have $M^\lambda(a)(\iota) = M^\lambda(\delta_a) = M^\lambda(\delta_b) = M^\lambda(b)(\iota)$, since the rule $\delta_a - \mathit{write}$ is not applicable.
- Constant array axiom: Assume we have $a = K(v)$. Then we have $M^\lambda(a)(\iota) = M^\lambda(v)$ for all $\iota \in M^\lambda(I)$, either because of $K \Downarrow$ for $\iota \in M^\lambda(\bigcup ct(a))$ or because of δ_a-*const*.
- sum-axioms: The choice $\delta_\mathbb{Z} = 0$ immediately shows that the axioms on infinite sums trivially hold. Then the rewrite rules are direct instantiations of the sum axioms for finite sums, according to our interpretation of arrays.

Theorem 3. *The above decision procedure runs in non-deterministic polynomial time.*

7 Overview of Decidability Results

We have shown that CAL with sums is undecidable, but that by removing the element-wise function applications, e.g., the map_f operators, we can retain decidability. We consider the following extensions of Arr_{cs} to examine the frontier of decidability:

- Adding another specific aggregation function. As an example we examine the minimum, respectively maximum of an array, i.e., $\mathit{min}(a) = x$.
- Pointwise function applications of functions with arity one, e.g. $b = \mathit{map}_f(a)$.

Lemma 5. *Minimum* $\min(a)$ *and maximum* $\max(a)$ *of an array can be defined in CAL.*

Proof. We consider the maximum operator. As an auxiliary function, the minimum $z = \min(x,y)$ of two integers x,y is defined by the following LIA formula:

$$z = \min(x,y) \;\Leftrightarrow\; (z = x \wedge x \leq y) \vee (z = y \wedge y \leq x \wedge x \neq y). \tag{30}$$

Then $\mathit{max}(a) = x$ if and only if $a = \mathit{map}_{\mathit{min}(\cdot,\cdot)}(a, K(x)) \wedge \exists i.\ \mathit{read}(a,i) = x$.

Theorem 4. *The satisfiability problem of formulas in Arr_{cs}, extended by constraints of the form $max(a) = c$ and $min(a) = c$, for constants $c \in \mathbb{Z}$, is decidable.*

Proof. Since minimum and maximum operator are subsumed by CAL, we can propagate reads through the corresponding *map* operators as in the decision procedure in [11]. If an array is not weakly equivalent to both an array involved in a sum constraint and an array in a minimum, respectively maximum operator the respective decision procedures from Section 6 and [11] apply. Therefore, without loss of generality let a be an array variable with infinite index sort, occurring in constraints $max(a) = c$ and $sum(a, v)$, where $c \in \mathbb{Z}$ is an integer constant. Then $sum(a, v)$ can be rewritten as

$$\Big(v - \sum_{[i]\in ct(a)} read(a, i)\Big) \in \{x \mid x \leq c\}^*. \tag{31}$$

Since c is an integer constant, the above is a LIA* formula and can be decided in non-deterministic polynomial time.

Remark 4. If we allow minimum/maximum constraints comparing an array with a variable, like in $\max(a) = x$, the resulting formula of the rewrite process in the proof of Theorem 4 is not a LIA* formula, since the characteristic function of the set for the star operator contains additional variables. It is still open whether the satisfiability problem of this extension is decidable or not.

We adapt the definition of weak equivalence in the presence of map_f by also allowing $a_i = map_f(a_{i+1})$ in Definition 12. Furthermore, if $a = map_f(b)$ occurs, then $ct(a)$ also satisfies $ct(b) = ct(a)$.

Theorem 5. *The satisfiability problem of formulas in Arr_{cs}, extended by point-wise function applications of functions of arity one, i.e., constraints of the form $a = map_f(b)$ where f is a LIA function of arity one, restricted to at most one sum constraint per weak equivalence class of array variables, is decidable.*

Proof. Let a be an array variable occuring in a sum constraint $sum(a, v)$ in the formula. Due to Lemma 1 we can assume that a has infinite index sort. We make a case distinction depending on whether a is weakly equivalent to a constant array. (1) If a is weakly equivalent to a constant array, then $\delta_a \neq \bot$. Hence, we have $\delta_a = 0$ in order to satisfy the sum constraint. In this case, the sum constraint can be rewritten as in (25) and the decision procedure for CAL applies. (2) If a is not weakly equivalent to a constant array, we can freely choose all remaining values. For each function application map_f in the formula we can compute the range of f, which is a semilinear set. Since $map_f(map_g(b)) = map_{f\circ g}(b)$ we can assume that all *map* operators are of the form $a = map_{f_i}(a_i)$ for $i \in \{1, \dots, n\}$. Let F_i be the respective formula that describes the image of f_i as a semilinear set, then define $F := \wedge_{i=1}^{n} F_i$. Then a sum constraint $sum(a, v)$ can be rewritten as

$$\Big(v - \sum_{[i]\in ct(a)} read(a, i)\Big) \in \{x \mid F(x)\}^*. \tag{32}$$

The above constraint asks whether the values that are not explicitly mentioned in the formula can form a sum that coincides with the difference between v and the sum of all explicitly mentioned values of a, which is decidable as a LIA^* formula.

Remark 5. The minimum and maximum operator without variables can be modeled by functions of arity one: $max(a) = c \Leftrightarrow a = map_{min(\cdot,c)}(a) \wedge read(a,i) = x$. For variables to be meaningfully included in an element-wise function application to an array, a function of arity greater than one is needed. For instance, $a = map_{min}(a, K(x))$ states $max(a) \leq x$, whereas $a = map_{min(\cdot,c)}(a)$ states $max(a) \leq c$ and c can not be a variable since otherwise $min(\cdot, c)$ is not a LIA function. Note that the undecidability proof in Theorem 1 only needs function applications of arity two.

8 Conclusion and Future Work

We provided a sound and complete decision procedure for an extensional array theory with constant arrays and sum constraints. The implementation of the procedure is ongoing work. We furthermore discussed the difficulties of defining sums over arrays and the borders of decidability that arise. Our decision procedure proves that decidability can be preserved for adding sums, however gets lost if we add pointwise function applications as in CAL. We examined additional decidability results for further extensions of our fragment.

Sum constraints can be seen as one of the most used and desired operators of the form $(I \Rightarrow E) \rightarrow E$. A direct generalization of our decision procedure would be to consider the element sort of the array to be an arbitrary Abelian semigroup $(E, *)$. Furthermore, one could look for other interesting operators for array theories to add to the decision procedure, such as minimum and maximum in the general case (discussed in Section 7). The question whether adding specific function applications, such as minimum and maximum operators with variables, preserves decidability remains open if we allow variables. A further goal is to explore the space between the theories CAL and CaAL to find array theories that can succinctly model quantum circuits (which is the main purpose of CaAL), have lower complexity than NEXPTIME, but also include summation to reason about probabilities.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Acknowledgement. This work was supported by the Swedish Research Council through grant 2021-06327, and by the Knut and Alice Wallenberg Foundation through project UPDATE. The authors thank the CADE reviewers for insightful comments!

References

1. Amilon, J., Esen, Z., Gurov, D., Lidström, C., Rümmer, P.: Automatic program instrumentation for automatic verification. In: Enea, C., Lal, A. (eds.) Computer

Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13966, pp. 281–304. Springer (2023). https://doi.org/10.1007/978-3-031-37709-9_14, https://doi.org/10.1007/978-3-031-37709-9_14

2. Baudin, P., Filliâtre, J.C., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI/ISO C Specification Language, http://frama-c.com/acsl.html
3. Chen, Y.F., Rümmer, P., Tsai, W.L.: A Theory of Cartesian Arrays (with Applications in Quantum Circuit Verification). In: Pientka, B., Tinelli, C. (eds.) Automated Deduction – CADE 29. pp. 170–189. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-38499-8_10
4. Christ, J., Hoenicke, J.: Weakly Equivalent Arrays. In: Lutz, C., Ranise, S. (eds.) Frontiers of Combining Systems. pp. 119–134. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-24246-0_8
5. Daca, P., Henzinger, T.A., Kupriyanov, A.: Array Folds Logic. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification. pp. 230–248. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_13
6. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, Cambridge (2009). https://doi.org/10.1017/CBO9780511576430, https://www.cambridge.org/core/books/handbook-of-practical-logic-and-automated-reasoning/EB6396296813CB562987E8C37AC4520D
7. Leavens, G.T., Baker, A.L., Ruby, C.: JML: A notation for detailed design. In: Kilov, H., Rumpe, B., Simmonds, I. (eds.) Behavioral Specifications of Businesses and Systems, The Kluwer International Series in Engineering and Computer Science, vol. 523, pp. 175–188. Springer (1999), https://doi.org/10.1007/978-1-4615-5229-1_12
8. Leino, K.R.M., Monahan, R.: Reasoning about comprehensions with first-order SMT solvers. In: Shin, S.Y., Ossowski, S. (eds.) Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009. pp. 615–622. ACM (2009). https://doi.org/10.1145/1529282.1529411, https://doi.org/10.1145/1529282.1529411
9. Matiyasevich, Y.V.: Hilbert's Tenth Problem. MIT Press, Cambridge, MA (1993)
10. McCarthy, J.: Towards a Mathematical Science of Computation. In: Fetzer, J.H., Colburn, T.R., Fetzer, J.H., Rankin, T.L. (eds.) Program Verification, vol. 14, pp. 35–56. Springer Netherlands, Dordrecht (1993). https://doi.org/10.1007/978-94-011-1793-7_2, http://link.springer.com/10.1007/978-94-011-1793-7_2, series Title: Studies in Cognitive Systems
11. Moura, L.d., Bjørner, N.: Generalized, Efficient Array Decision Procedures (Sep 2009), https://www.microsoft.com/en-us/research/publication/generalized-efficient-array-decision-procedures/
12. Piskac, R., Kuncak, V.: Linear Arithmetic with Stars. In: Gupta, A., Malik, S. (eds.) Computer Aided Verification. pp. 268–280. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_25
13. Raya, R.: Combinatory Array Logic with Sums (May 2024). https://doi.org/10.48550/arXiv.2311.06582, http://arxiv.org/abs/2311.06582, arXiv:2311.06582 [cs] version: 2
14. Raya, R., Kunčak, V.: NP Satisfiability for Arrays as Powers. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 301–318. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-94583-1_15

15. Stump, A., Barrett, C., Dill, D., Levitt, J.: A decision procedure for an extensional theory of arrays. In: Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. pp. 29–37. IEEE Comput. Soc, Boston, MA, USA (2001). `https://doi.org/10.1109/LICS.2001.932480`, `http://ieeexplore.ieee.org/document/932480/`