A Decision Procedure for an Extensional Theory of Arrays

Aaron Stump, Clark W. Barrett, and David L. Dill
Computer Systems Laboratory
Stanford University, Stanford, CA 94305, USA
E-mail: {stump,dill,barrett}@cs.stanford.edu

Jeremy Levitt
0-In Design Automation, Inc.
San Jose, CA 95110, USA
Email: levitt@0-In.com

Abstract

A decision procedure for a theory of arrays is of interest for applications in formal verification, program analysis, and automated theorem-proving. This paper presents a decision procedure for an extensional theory of arrays and proves it correct[1].

1. Introduction

A decision procedure for a theory of arrays is of interest for applications in formal verification and program analysis. Such a procedure is also of value for theorem-provers. The PVS theorem-prover [11] has an undocumented decision procedure for a theory of arrays [12], and HOL has some automatic support for a theory of arrays via a library for finite partial functions [3].

Two kinds of array theories have been studied previously. Extensional theories require that if two arrays store the same value at index i, for each index i, then the arrays must be the same. Non-extensional theories do not make this requirement. This paper is the first to present a procedure for checking satisfiability of arbitrary quantifier-free formulas in an extensional theory of arrays and prove its correctness.

2. Theories of arrays

Decision procedures for various theories of arrays have been studied previously. Most of these theories can be divided into extensional and non-extensional varieties. In this section, several families of array theories are axiomatized in classical first-order multi-sorted logic with equality. The theory **Arr** decided in this paper is then presented and compared to previously decided theories.

[1]This paper was published in the proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS), 2001.

2.1. The language

Sorts The language has a basic sort I for indices into arrays. It also has value sorts, which are the sorts of individuals that may be stored in arrays. The sort V is the sort for primitive values stored in arrays. The set of value sorts is defined to be the least set X satisfying

- $V \in X$
- $\tau \in X \rightarrow \mathit{array}_\tau \in X$

Every value sort except V is an array sort. The value sorts together with I are all the sorts of the language. V and I need not be distinct.

Definition 1 (dimensionality of a value sort) *The dimension $\mathit{dim}(\tau)$ of a value sort τ is defined by*

- $\mathit{dim}(V) = 0$
- $\mathit{dim}(\mathit{array}_\tau) = \mathit{dim}(\tau) + 1$

Terms The language has countably infinitely many variables and constants of each distinct sort. The constants are uninterpreted, in the sense they will not occur in any axiom or axiom scheme. The function symbols of the language are

- read_τ of type $(\mathit{array}_\tau \rightarrow I \rightarrow \tau)$, for every value sort τ
- write_τ of type $(\mathit{array}_\tau \rightarrow I \rightarrow \tau \rightarrow \mathit{array}_\tau)$, for every value sort τ

Subscripts on *read* and *write* will generally be omitted. Informally, $\mathit{read}(a, i)$ will denote the value stored in array a at index i, and $\mathit{write}(a, i, v)$ will denote an array which stores the same value as a for every index except possibly i, where it stores value v.

Terms are built up in the usual way from constants and variables using the function symbols. Terms whose sort is an array sort will be called array terms. Terms whose sort is I will be called index terms. The dimension $dim(a)$ of an array term a is the dimension of its sort. If $dim(a) = n$, array a is said to be n-dimensional. If $n > 1$, a is also said to be multi-dimensional.

Formulas The atomic formulas of the language are the equations between terms of the same sort. Formulas are built up from atomic formulas using propositional connectives and quantifiers in the usual way. A formula is *closed* if it has no free variables. A *literal* is an atomic formula or the negation of an atomic formula. A *theory* is a set of closed formulas.

2.2. Theories

Some theories restrict which array sorts are allowed. If a theory allows array sorts of dimension at most n, it is said to have just n-dimensional arrays. If a theory allows all array sorts, it is said to have multi-dimensional arrays.

The following scheme, which is schematic in a value sort τ, is called the read-over-write axiom scheme. Informally, it says that for all arrays a, indices i and j, and values v of suitable type, reading the value stored at index j of $write(a, i, v)$ is v if the two indices are equal and $read(a, j)$ if they are different.

Axiom scheme 1 (read-over-write)

$$\begin{aligned}&\forall\, a : array_\tau \,.\, \forall\, i : I \,.\, \forall\, j : I \,.\, \forall\, v : V \,.\\ &\quad (i = j \rightarrow read(write(a, i, v), j) = v) \wedge\\ &\quad (i \neq j \rightarrow read(write(a, i, v), j) = read(a, j))\end{aligned}$$

The following scheme, which is schematic in a value sort τ, is called the extensionality axiom scheme. Informally, it expresses a principle of extensionality for arrays: if two arrays store the same value at index i, for each index i, they are equal.

Axiom scheme 2 (extensionality)

$$\begin{aligned}&\forall\, a : array_\tau \,.\, \forall\, b : array_\tau \,.\\ &\quad (\forall\, i : I \,.\, read(a, i) = read(b, i)) \rightarrow a = b\end{aligned}$$

The extensional theories are those axiomatized by the read-over-write and extensionality axiom schemes. The non-extensional theories are those axiomatized by just the read-over-write axiom scheme. Note that since a theory is a set of closed formulas, quantifier-free array theories have no variables; all 0-ary symbols are (uninterpreted) constants.

2.3. The theory Arr

The theory **Arr** decided in this paper is the quantifier-free fragment of the extensional theory with multi-dimensional arrays where sort V is defined to be sort I. So indices are the values stored in 1-dimensional arrays.

The restriction to the quantifier-free fragment is justified by the fact that the fully quantified theory is undecidable, even in the absence of the function symbols $write_\tau$ and the read-over-write scheme. This is because single-sorted first-order theories with function symbols and equality may be translated into this array theory in such a way that a first-order formula is valid iff its translation is. The translation maps constant symbols to index constants, n-ary function symbols to n-dimensional array constants, and terms like $f(i_1, \ldots, i_n)$ to nested read expressions $read(\ldots read(read(f', i'_1), i'_2) \ldots, i'_n)$, where $f', i'_1, \ldots, i'_n$ are the translations of $f, i_1, \ldots, i_n$. The undecidability results for classical first order logic with just function symbols and equality (see, e.g., [5]) can then be applied to show that even quite restricted quantified fragments of the extensional theory of arrays are undecidable.

A decision procedure for **Arr** may be useful even for applications which require a fully quantified logic. Many theorem provers, such as the widely used PVS [11], provide strategies to reduce goals to subgoals in decidable fragments of their logic.

2.4. Comparison with related work

In this section, related work is summarized by describing which theories are decided. These theories often use axiomatizations different from but equivalent to that of **Arr**. All the theories decided are quantifier-free. Kaplan is the only one to distinguish the sorts V and I. Many of the previous theories allow arithmetic operators or uninterpreted functions over sort I to be used in addition to the symbols *read* and *write*. The restriction here to just the essential theory of arrays is justified by the fact that, as will be shown in Section 6 below, the satisfiability procedure for **Arr** is suitable for incorporation into a framework for cooperating decision procedures [2]. In such a framework, separate decision procedures for arithmetic and uninterpreted functions may be combined with the decision procedure for **Arr** to decide the combined theory.

The first two works present axioms but no decision procedure for their theories. With the exception of Levitt's work, the others give decision procedures for theories that are strictly weaker than **Arr**, either because they restrict the form of formulas in the theory (e.g., to just equations), disallow equations between arrays, or are non-extensional.

McCarthy In [8], McCarthy introduces the function symbols *read* and *write* and gives an informal semantics for an extensional theory of arrays based on them.

Collins and Syme Collins and Syme present in HOL a theory of finite higher-order partial functions similar to a theory with multi-dimensional arrays [3].

Kaplan In [6], Kaplan gives a decision procedure for a non-extensional equational theory with just 1-dimensional arrays. He considers equations between index terms only, which is reasonable since his theory contains no non-trivial equations between arrays. He then shows how to extend his procedure to decide an extensional equational theory, where the equations may be between array as well as index terms. He imposes the restriction that distinct variables of sort I must receive distinct interpretations.

Suzuki and Jefferson In [15], Suzuki and Jefferson present a decision procedure for a theory with just 1-dimensional arrays, where equations between arrays are not allowed. The theory has axioms for extensionality and the existence of constant arrays (arrays that store the same value at all indices), but these appear to be included for technical reasons only; the theory decided is equivalent to the one without those axioms under the restrictions they impose. They extend their procedure to decide a theory with a new predicate symbol *PERM*, where $\mathit{PERM}(a, b)$ holds iff the multiset of the values stored in a is contained in the multiset of the values stored in b. Sentences of the theory are restricted to the form $P \rightarrow \mathit{PERM}(a, b)$, where P is any (quantifier-free) sentence not containing *PERM*. **Arr** does not have the *PERM* predicate, but inspection of the way Suzuki and Jefferson extend their algorithm to treat *PERM* shows that it could just as easily be used to extend the algorithm for **Arr**, as long as their restriction disallowing equations between array terms were retained.

Downey and Sethi In [4], Downey and Sethi present a decision procedure for an extensional equational theory with just 1-dimensional arrays. Equations between array terms are allowed. They prove that determining the invalidity of an equation in their theory of arrays is NP-complete.

Nelson and Oppen In [10], Nelson and Oppen describe an extensional theory of arrays. Their theory allows multi-dimensional arrays. They do not present their satisfiability procedure for the extensional theory, but in [9], Nelson gives a detailed presentation of a satisfiability procedure for a non-extensional theory.

Levitt In Chapter 5 of his PhD thesis [7], Levitt presents a decision procedure for an extensional theory of arrays based on solving equations and canonizing terms, in the style of Shostak [13]. A detailed proof of correctness is not given, and has proved elusive to the authors. In contrast, a detailed proof of correctness is given below for the procedure for **Arr**.

3. The satisfiability procedure for Arr

Arr is decided by a refutation procedure. The procedure decides satisfiability of conjunctions of literals, which are equations and disequations between terms. Deciding satisfiability of arbitrary boolean combinations of atomic formulas can be reduced to this problem by well-known means. A conjunction of literals whose satisfiability is to be tested will be called a goal. Comma will be used to denote conjunction. Two goals are said to be *equisatisfiable* when one is satisfiable iff the other is.

3.1. Informal overview

The procedure works in two phases. In the first phase, the original goal is transformed into a set of subgoals such that (i) no subgoal contains *write* and (ii) the original goal is satisfiable iff one of the subgoals is. Eliminating write expressions is straightforward except when they occur as the left or right hand side of an equation. How to eliminate such occurrences of write expressions is the crucial insight of this algorithm.

Definition 2 ($=_\_$)

$$a =_{\mathcal{I}} b \quad \Leftrightarrow_{def} \quad \forall\, i : I \,.\, i \notin \mathcal{I} \rightarrow \mathit{read}(a, i) = \mathit{read}(b, i)$$

Formulas of the form $a =_{\mathcal{I}} b$ *with* $\mathcal{I} \neq \emptyset$ *are called partial equations.*

The crucial observation is that

$$\mathit{write}(a, i, v) = b \;\Leftrightarrow\; (a =_{\{i\}} b \wedge \mathit{read}(b, i) = v).$$

write expressions occurring as sides of equations may thus be eliminated by introducing partial equations.

The second phase of the procedure is based on the observation that in the absence of *write*, arrays behave like uninterpreted functions and *read* behaves like function application. So in the absence of *write*, a congruence closure algorithm (cf. [1]) could be used to decide the theory. The algorithm must be modified to work with partial equations as well as equations, but this can be done. For simplicity, the very simple congruence closure algorithm described in [14] is used, but it should be possible to modify a more complex algorithm.

3.2. Formal presentation

Figure 1 presents our procedure as a proof system. The proof system determines a non-deterministic procedure, where rules are applied bottom-up to analyze a goal into one or more subgoals. The system may be thought of as a rewrite system, where, for each rule, the goal below the line is rewritten to the subgoals above the line. The system resembles a Gentzen-Schütte system where only left rules of the corresponding sequent system are used (i.e., a sequent system where sequents are restricted to be of the form $\Gamma \Rightarrow \bot$). The derivable objects of this system are sets of literals. It is intended that a set of literals be derivable iff their conjunction is unsatisfiable. A *deduction* of a goal is a tree obtained by applying the proof rules bottom-up to that goal. A goal to which no rule can be applied is said to be *normal*.

Phase 1:

(ext) $$\frac{\Gamma,\ \mathit{read}(a,k) \neq \mathit{read}(b,k)}{\Gamma,\ a \neq b}$$ k is not free in the conclusion; a and b are arrays

(r-over-w) $$\frac{\Gamma[v],\ i = j \quad \Gamma[\mathit{read}(a,j)],\ i \neq j}{\Gamma[\mathit{read}(\mathit{write}(a,i,v),j)]}$$

(w-elim) $$\frac{\Gamma,\ a =_{\mathcal{I}} b,\ i \in \mathcal{I} \quad \Gamma,\ a =_{i,\mathcal{I}} b,\ \mathit{read}(b,i) = v,\ i \notin \mathcal{I}}{\Gamma,\ \mathit{write}(a,i,v) =_{\mathcal{I}} b}$$

(w-elim-helper) $$\frac{\Gamma,\ b =_{\mathcal{I}} a}{\Gamma,\ a =_{\mathcal{I}} b}$$ b is a write expression, and a is not

Phase 2:

(partial-eq) $$\frac{\Gamma,\ a =_{\mathcal{I}} b,\ \mathit{read}(a,i) = \mathit{read}(b,i),\ i \notin \mathcal{I} \quad \Gamma,\ a =_{\mathcal{I}} b,\ i \in \mathcal{I}}{\Gamma,\ a =_{\mathcal{I}} b}$$
where $a \succeq b$; $\mathcal{I} \neq \emptyset$; $\mathit{read}(a,i)$ occurs in Γ

(trans) $$\frac{\Gamma,\ a =_{\mathcal{I}} b,\ a =_{\mathcal{I}'} c,\ b =_{\mathcal{I} \cup \mathcal{I}'} c}{\Gamma,\ a =_{\mathcal{I}} b,\ a =_{\mathcal{I}'} c}$$ $\mathcal{I} \neq \emptyset$ and $\mathcal{I}' \neq \emptyset$

(subst) $$\frac{\Gamma[y],\ x = y}{\Gamma[x],\ x = y}$$ $x \succeq y,\ x \not\equiv y$, x not in $\Gamma[\,]$

(symm) $$\frac{\Gamma,\ y =_{\mathcal{I}} x}{\Gamma,\ x =_{\mathcal{I}} y}$$ $x \prec y$

Both phases:

($\in$-split) $$\frac{\Gamma,\ i = j \quad \Gamma,\ i \in \mathcal{I}}{\Gamma,\ i \in (j, \mathcal{I})}$$ ($\notin$-expand) $$\frac{\Gamma,\ i \notin \mathcal{I},\ i \neq j}{\Gamma,\ i \notin (j, \mathcal{I})}$$

($\in$-empty) $$\frac{}{\Gamma,\ i \in \emptyset}$$ (ax) $$\frac{}{\Gamma,\ x \neq x}$$

Figure 1. The decision procedure as a proof system

The system has two phases. Some rules may be applied in just one phase, while others may be applied in either phase. The rules of phase 1 are applied to a goal until no rule applies, and then the rules of phase 2 are applied. The procedure stops and reports that the original conjunction is satisfiable if it encounters a normal subgoal. Otherwise, it reports that the original goal is unsatisfiable. As mentioned before, phase 2 is a modified congruence closure algorithm. The core congruence closure algorithm consists of just the rules (symm) and (subst) [14].

The set-theoretic operators have their usual meanings; note that $i, \mathcal{I}$ denotes $\{i\} \cup \mathcal{I}$, where $\mathcal{I}$ does not contain i. $\Gamma[\,]$ denotes a *context*, which is an expression containing one or more occurrences of a single free variable. The expression obtained by substituting the term t for the context's free variable is written $\Gamma[t]$. In the rule (subst), since the side condition requires that $\Gamma[\,]$ contain no occurrences of the term x, applying (subst) replaces all occurrences of x in $\Gamma[x]$ with the term y. $\equiv$ denotes syntactic identity. The symbol $\preceq$ denotes an ordering on terms by size, which is defined on terms in the usual way. Let $x \preceq y$ iff x and y are such that the size of x is less than or equal to the size of y. The variants $\prec$ and $\succeq$ are derived from $\preceq$ in the usual way.

3.3. Avoiding non-termination in phase 2

In phase 2, applications of (partial-eq) and (trans) must be restricted to avoid certain sources of non-termination. There is nothing preventing (partial-eq) and (trans) from being applied repeatedly with the same partial equations, because for both rules, the partial equations are retained in the goal. For (partial-eq), this form of non-termination may be prevented by adding a side condition to the rule that prevents it from being applied if, informally, $\mathit{read}(a,i)$ and $\mathit{read}(b,i)$ are already known to be equal or if i is already known to be equal to an element of $\mathcal{I}$. Formally, the procedure can test whether or not t and t' are already known to be equal by applying all the rules of phase 2 except (partial-eq) and (trans) to the current goal with $t \neq t'$ added, and seeing whether or not that goal is reported unsatisfiable. If neither ($\in$-split) nor ($\notin$-expand) applies to the current goal, then this is equivalent just to comparing normal forms as determined by the core congruence closure algorithm. So in an implementation, this non-termination may easily be prevented. A similar approach can be used to prevent (trans) from being applied repeatedly to the same formulas. The required machinery, however, has been omitted from the proof system for simplicity.

4. Correctness of the Procedure

A satisfiability procedure is *sound* iff when it reports a goal unsatisfiable, the goal is indeed unsatisfiable. A procedure is *complete* iff when it reports a goal satisfiable, the goal is indeed satisfiable. A procedure is *correct* iff it terminates on all inputs, and it is sound and complete. In this section, a detailed proof of completeness for the satisfiability procedure for **Arr** is given. The proof of termination is routine and omitted for lack of space. The following theorem implies soundness.

Theorem 1 (equisatisfiability) *The conclusion of each rule of the system is satisfiable iff one of its premises is satisfiable.*

Proof: The proof is routine. Consider just the rule (trans). If $a =_{\mathcal{I}} b$ and $a =_{\mathcal{I}'} c$ are true in some model, then it is easy to see by the definition of $=_{\_}$ that $b =_{\mathcal{I}\cup\mathcal{I}'} c$ is also true in some model. If c agrees with a at every index except those in $\mathcal{I}'$ and a agrees with b at every index except those in $\mathcal{I}$, then clearly $i \notin \mathcal{I} \cup \mathcal{I}'$ implies that c agrees with a at i and also that a agrees with b at i. Hence, c agrees with b at i. For the other direction, if the premise has a model, so does the conclusion, since the conclusion is a subset of the premise. □

Recall that a normal goal is one to which no rule applies. By the equisatisfiability theorem, to prove completeness of the algorithm it suffices to show that any normal goal is satisfiable. This may be done by constructing a model for a normal goal. The following lemma is easily established.

Lemma 1 (effect of phase 1) *A goal that is normal with respect to phase 1 of the algorithm contains no write expressions and no disequations between array expressions.*

4.1. A convenient form for normal goals

In preparation for constructing a model, several transformations, which are not actually performed by the algorithm, are applied to a normal goal to give an equisatisfiable normal goal Γ, which is in a more convenient form. If the normal goal contains equations of the form $x = x$, clearly they may be removed and the result will be equisatisfiable. Next, modify the goal by doing the following. Let G be the goal as it currently stands. If there is a term of the form $\mathit{read}(a,i)$ in G that is not the left hand side of any equation in G, choose a constant symbol c not occurring in G, and modify G by replacing $\mathit{read}(a,i)$ everywhere in it with c and adding the equation $\mathit{read}(a,i) = c$ to it. If there is no such term $\mathit{read}(a,i)$ in G, stop. It is easy to show that the resulting goal is normal and equisatisfiable with the original normal goal. This resulting goal consists of formulas of one of the following four forms, where x, y, and z are constant symbols:

I. $\mathit{read}(x,y) = z$

II. $x \neq y$

III. $x =_{\mathcal{I}} y$, where every element of $\mathcal{I}$ is a constant symbol

IV. $x = y$

Since this resulting goal is normal, no formula $x = y$ of the form (IV) has its left hand side appearing anywhere else in the goal, since otherwise (subst) would apply. Let Γ be this resulting goal, except without the equations of the form (IV). Γ will be said to be in *convenient normal form*. Any model $\mathcal{M}$ of Γ may be extended to a model of Γ with those equations of the form (IV) by giving the same interpretation for the constant x as for the constant y, if $\mathcal{M}$ interprets y, and a single arbitrary interpretation for both x and y otherwise.

4.2. Construction of a model

In this section, a kind of term model for the goal Γ in convenient normal form is constructed. Several definitions, in terms of Γ, are required. The fact that the core congruence closure algorithm (rules (subst) and (symm)) is correct is used (see [14] for the proof).

Definition 3 ($\rightarrow\_$ and $\leftarrow\_$) *Let $\rightarrow\_$ and $\leftarrow\_$ be the ternary relations defined, respectively, by*

$$\begin{array}{lll} a \rightarrow_{\mathcal{I}} b & \textit{iff} & (a =_{\mathcal{I}} b) \in \Gamma \\ a \leftarrow_{\mathcal{I}} b & \textit{iff} & (b =_{\mathcal{I}} a) \in \Gamma \end{array}$$

Note that for any $\mathcal{I}$, $\rightarrow_{\mathcal{I}}$ and $\rightarrow_{\mathcal{I}}$ need not be symmetric, since $(a =_{\mathcal{I}} b) \in \Gamma$ does not imply $(b =_{\mathcal{I}} a) \in \Gamma$.

Definition 4 ($\approx\_$) *Let $\approx\_$ be the least ternary relation satisfying*

1. *$a \approx_{\emptyset} a$, for every array constant a appearing in Γ*
2. *$(a \rightarrow_{\mathcal{I}} b) \vee (b \rightarrow_{\mathcal{I}} a) \rightarrow a \approx_{\mathcal{I}} b$*

Definition 5 ($\overset{*}{\approx}\_$) *Let $\overset{*}{\approx}\_$ be the least ternary relation containing $\approx\_$ and satisfying*

$$(\exists\, c\,.\, a \approx_{\mathcal{I}} c \wedge c \overset{*}{\approx}_{\mathcal{I}'} b) \rightarrow a \overset{*}{\approx}_{\mathcal{I} \cup \mathcal{I}'} b$$

Definition 6 ($\overset{*}{\approx}$) *Let $\overset{*}{\approx}$ be the binary relation defined by*

$$a \overset{*}{\approx} b \quad \textit{iff} \quad \exists\, \mathcal{I}\,.\, a \overset{*}{\approx}_{\mathcal{I}} b$$

The context will help distinguish $\overset{*}{\approx}\_$ and $\overset{*}{\approx}$. Note that $\overset{*}{\approx}$ is an equivalence relation.

Definition 7 (chains) *A chain of applications of a ternary symbol R like $\approx\_$ or $\rightarrow\_$, called an R-chain, is defined to be a conjunction of the form $(a_1\ R_{\mathcal{I}_1}\ a_2) \wedge (a_2\ R_{\mathcal{I}_2}\ a_3) \wedge \ldots \wedge (a_{n-1}\ R_{\mathcal{I}_{n-1}}\ a_n)$, with $n \geq 2$.*

- *The chain is denoted $(a_1\ R_{\mathcal{I}_1}\ a_2\ R_{\mathcal{I}_2}\ \ldots\ R_{\mathcal{I}_{n-1}}\ a_n)$.*
- *n is the* length *of the chain.*
- *The* union along the chain *is defined to be $\bigcup_{1 \leq j < n} \mathcal{I}_j$.*
- *The chain is said to be* from x to y *iff $a_1 \equiv x$ and $a_n \equiv y$.*

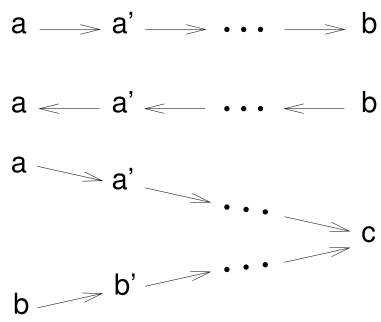

Figure 2. Standard forms for $\approx\_$-chains

Lemma 2 (standard form for chains) *Suppose $a \overset{*}{\approx}_{\mathcal{I}} b$, with $\mathcal{I} \neq \emptyset$. Then one of the following is true:*

i. *there is a $\rightarrow\_$-chain from a to b or from b to a, where the union along the chain is $\mathcal{I}$*

ii. *for some c, there is a $\rightarrow\_$-chain from a to c and another from b to c, where the union of the unions along the two chains is $\mathcal{I}$.*

Figure 2 shows the possibilities.

Proof Let C be a $\approx\_$-chain $a_1 \approx_{\mathcal{I}_1} \ldots \approx_{\mathcal{I}_{n-1}} a_n$ from a to b, with $\mathcal{I} = \bigcup_{1 \leq i \leq n-1} \mathcal{I}_i$. Assume C is of minimal length of all such chains. For every i with $1 \leq i \leq n-1$, let $\leftrightarrow_i$ be either $\rightarrow_{\mathcal{I}_i}$ or $\leftarrow_{\mathcal{I}_i}$, and suppose we have $a_1 \leftrightarrow_i \ldots \leftrightarrow_{n-1} a_n$. It is easy to prove that if this latter chain is not of one of the forms described in (i) and (ii), there must be an i with $1 < i \leq n-1$ such that $\leftrightarrow_{i-1}$ is $\leftarrow_{\mathcal{I}_{i-1}}$ and $\leftrightarrow_i$ is $\rightarrow_{\mathcal{I}_i}$. So we have $a_{i-1} \leftarrow_{\mathcal{I}_{i-1}} a_i \rightarrow_{\mathcal{I}_i} a_{i+1}$. So both $a_i =_{\mathcal{I}_{i-1}} a_{i-1}$ and $a_i =_{\mathcal{I}_i} a_{i+1}$ are in Γ. It must be the case that both $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ are non-empty, since otherwise (subst) would apply to replace the left hand side of one of those equations by the right hand side of the other. No rules can apply, since Γ is normal. Since both $\mathcal{I}_{i-1}$ and $\mathcal{I}_i$ are non-empty, (trans) would be applicable, unless the conditions described in Section 3.3 for preventing non-termination were keeping it from being applied. This implies that either $a_{i-1} =_{\mathcal{I}_{i-1} \cup \mathcal{I}_i} a_{i+1}$ or $a_{i+1} =_{\mathcal{I}_{i-1} \cup \mathcal{I}_i} a_{i-1}$ is in Γ, since a_1 and a_2 must be their own normal forms as determined by the core congruence closure algorithm. Hence, we have $a_{i-1} \approx_{\mathcal{I}_{i-1}\cup\mathcal{I}_i} a_{i+1}$. So the chain $a_1 \approx_{\mathcal{I}_1} \ldots a_{i-1} \approx_{\mathcal{I}_{i-1}\cup\mathcal{I}_i} a_{i+1} \ldots \approx_{\mathcal{I}_{n-1}} a_n$, whose union is $\mathcal{I}$, has smaller length than C. This contradicts the assumption that C is of minimal length of such chains. □

Now an interpretation, given as a function $[\![\_]\!]$ from the constant and function symbols of Γ to their interpretations, is defined. $[\![\_]\!]$ is defined to map every constant symbol a of basic type I to a itself. $[\![\_]\!]$ will map array constants to functions. To satisfy extensionality, functions that give the same value for every input are required to be equal. First let $\perp_C$ be a new symbol not occurring in Γ, for every $\overset{*}{\approx}$-equivalence class C. Define $[\![read]\!]$ to be the operation of function application, except that when it is given $\perp_C$, it may just return $\perp_C$. Intuitively, for an array constant a, $[\![a]\!]$ will be a function mapping all but a finite number of inputs to a default value $\perp_C$. Formally, suppose a is in $\overset{*}{\approx}$-equivalence class C. Define $[\![a]\!]$ to be the function that returns $\perp_C$ for every input, except those assigned values by the following:

Definition 8 (interpretation of array constants)

for every constant symbol b of the same type as a,
for every set $\mathcal{I}$ such that $a \overset{*}{\approx}_{\mathcal{I}} b$,
for every index constant i not appearing in $\mathcal{I}$,
if $read(b,i) = x \in \Gamma$ for some x, then
the value of $[\![a]\!]$ for input $[\![i]\!]$ is defined to be $[\![x]\!]$.

Notice that the body of Definition 8 may specify the value for $[\![a]\!]$ on input i more than once. So for $[\![\,]\!]$ to be well-defined, if the value of $[\![a]\!]$ on input i is specified to be $[\![x_1]\!]$ and $[\![x_2]\!]$, we need $[\![x_1]\!] = [\![x_2]\!]$. So if $a \overset{*}{\approx}_{\mathcal{I}} b$ and $a \overset{*}{\approx}_{\mathcal{I}'} c$ with i not in $\mathcal{I}$ and not in $\mathcal{I}'$, then for $[\![\,]\!]$ to be well-defined, it must be the case that if $read(b,i) = x_1, read(c,i) = x_2 \in \Gamma$, then $[\![x_1]\!] = [\![x_2]\!]$. Since the conditions $a \overset{*}{\approx}_{\mathcal{I}} b$, $a \overset{*}{\approx}_{\mathcal{I}'} c$, i not in $\mathcal{I}$, and i not in $\mathcal{I}'$ together imply $b \overset{*}{\approx}_{\mathcal{I}\cup\mathcal{I}'} c$ and i not in $\mathcal{I}\cup\mathcal{I}'$, the following lemma suffices to prove that $[\![\,]\!]$ is indeed well-defined.

Lemma 3 (well-definedness of $[\![\,]\!]$) *If $a \overset{*}{\approx}_{\mathcal{I}} b$, i not in $\mathcal{I}$, and $read(a,i) = x_1, read(b,i) = x_2 \in \Gamma$, then $x_1 \equiv x_2$.*

The proof of this lemma relies on the following sub-lemma.

Lemma 4 (certain reads equal along chains) *Suppose $a_1, \ldots, a_n$, and i are such that $a_1 \to_{\mathcal{I}_1} \ldots \to_{\mathcal{I}_{n-1}} a_n$ for some $\mathcal{I}_1, \ldots, \mathcal{I}_{n-1}$, where i is not in $\bigcup_{1\leq j\leq n-1} \mathcal{I}_j$. Suppose there is a constant x such that $read(a_1,i) = x \in \Gamma$. Then $read(a_n,i) = x \in \Gamma$.*

Proof The proof is by induction on n. The base case is trivial. For the induction case, suppose $read(a_1,i) = x \in \Gamma$. Since Γ is normal, no rules can apply. So we must have $\mathcal{I}_1 \neq \emptyset$, since otherwise (subst) would apply with $a_1 = a_2$ and $read(a_1,i)$. Furthermore, since (partial-eq) cannot apply, it must be the case that the conditions of Section 3.3 for preventing non-termination are what is prohibiting its application with $a_1 =_{\mathcal{I}_1} a_2$ and $read(a,i)$. In particular, it must be the case that $read(a_2,i)$ is already known to be equal to $read(a_1,i)$. The other possibility, namely that i is known to be equal to an element of $\mathcal{I}$, is excluded because i is not in $\mathcal{I}$ by hypothesis, and correctness of the core congruence closure algorithm would require i to appear in $\mathcal{I}$ in a normal goal if i were known to be equal to an element of $\mathcal{I}$. For $read(a_1,i)$ and $read(a_2,i)$ to have the same normal form with respect to the core congruence closure algorithm, we must have $read(a_2,i) = x \in \Gamma$; this follows from the definition of convenient normal form. Now the induction hypothesis may be applied to conclude that $read(a_n,i) = x \in \Gamma$. □

Proof (of Lemma 3) Suppose $a \overset{*}{\approx}_{\mathcal{I}} b$ and suppose $\mathcal{I} \neq \emptyset$. Then by Lemma 2, there is either a $\to_{\_}$-chain from a to b or from b to a, or there is a constant c such that there is a $\to_{\_}$-chain from a to c and another from b to c. By Lemma 4, in the first case either $read(b,i) = x_1 \in \Gamma$ or $read(a,i) = x_2 \in \Gamma$, and in the second, $read(c,i) = x_1, read(c,i) = x_2 \in \Gamma$. Since Γ is normal, for all x, y, and z, $read(x,i) = y, read(x,i) = z \in \Gamma$ implies $y \equiv z$, since otherwise (subst) would apply. So in either case, $x_1 \equiv x_2$. If $\mathcal{I} = \emptyset$, then it must be the case that $a \equiv b$, since $read(a,i)$ and $read(b,i)$ are both in Γ; otherwise, (subst) would apply. But again, $read(a,i) = x, read(a,i) = y \in \Gamma$ implies that $x \equiv y$. □

Lemma 5 (correctness of the constructed model) *The model constructed in the previous section satisfies every formula of the goal Γ in convenient normal form.*

Proof Consider the types (I), (II), and (III) of formulas from the list in section 4.1; recall that goals in convenient normal form consist of formulas of just these types.

Case I: $\boldsymbol{read}(x,y) = z$ Since x is an array constant, $x \approx_{\emptyset} x$, and so the construction of Definition 8 will assign the value that function $[\![x]\!]$ takes on argument $[\![y]\!]$ to be $[\![z]\!]$. Hence $[\![read(x,y)]\!] = [\![z]\!]$.

Case II: $x \neq y$ Since all disequations in Γ are between index expressions, x and y must be index constants. Hence, $[\![x]\!] = x$ and $[\![y]\!] = y$, by construction. If $x \equiv y$, then the goal would not be normal, because (ax) would apply. So the interpretation satisfies $x \neq y$.

Case III: $x =_{\mathcal{I}} y$ It must be shown that for every index constant not in $[\![\mathcal{I}]\!]$, $[\![x]\!]$ and $[\![y]\!]$ give the same value. $[\![x]\!]$ and $[\![y]\!]$ have the same default value since they are in the same $\overset{*}{\approx}$-equivalence class. For those index constants i not in $\mathcal{I}$ that appear in a formula of the form $read(y, i) = z \in \Gamma$, they store the same values, by Definition 8. □

From the fact that a model has been constructed for a normal goal, the main result now follows.

Theorem 2 (completeness) *The satisfiability procedure for **Arr** is complete.*

5. Complexity analysis

Observe that each application of (w-elim) or (partial-eq) leads to one new subgoal for each element of the indexing set $\mathcal{I}$ in the rule. The size of $\mathcal{I}$ is easily seen to be bounded by the size N of the original goal Γ. So any deduction from Γ may be viewed as a tree with branching factor no more than N. It is not hard to show, in fact, that N is an upper bound on the number of branching nodes in the tree, so there are at most $O(N^N) = O(2^{N \lg N})$ branches. Each branch can be shown to be of polynomial length, so the algorithm runs in worst-case exponential time.

Theorem 3 (NP-completeness) *The problem of testing a conjunction of literals for satisfiability in **Arr** is NP-complete.*

Proof Downey and Sethi showed that a subproblem of the problem decided here is NP-hard [4]. To show that the problem is in NP, observe that the size of the model constructed in the previous section for a goal Γ in convenient normal form is polynomial in the size of Γ. The conversion of a normal goal to convenient normal form incurs at most a polynomial expansion of the goal. So the size of the model constructed is polynomial in the size of the normal goal. Hence a model can be nondeterministically guessed in polynomial time. Checking whether or not a conjunction of literals is satisfied by a model can be done deterministically in polynomial time. So satisfiability of a conjunction of literals can be checked nondeterministically in polynomial time. □

6. Extensions

In this section, several extensions to the refutation procedure for **Arr** are considered. Due to lack of space, correctness proofs are omitted.

6.1. Propagating all entailed equations

Full incorporation of the satisfiability procedure into the framework for cooperating procedures of [2] requires that the procedure can discover all equations between terms occurring in a satisfiable goal that are entailed by that goal. The procedure for **Arr** always does this for index terms but not always for array terms. If the rules of Figure 3 are added to phase 2, however, it can be shown that if t and t' are array terms in a normal goal that are entailed to be equal, then $t \overset{*}{\approx}_{\emptyset} t'$.

$$\text{(trans2)} \quad \frac{\Gamma,\ a =_{\mathcal{I}} b,\ b =_{\mathcal{I}'} c,\ a =_{\mathcal{I} \cup \mathcal{I}'} c}{\Gamma,\ a =_{\mathcal{I}} b,\ b =_{\mathcal{I}'} c} \quad \text{where } \mathcal{I} \neq \emptyset \text{ and } \mathcal{I}' \neq \emptyset$$

$$\text{(patch)} \quad \frac{\Gamma,\ \neg\phi,\ a =_{i,\,\mathcal{I}} b \qquad \Gamma,\ \phi,\ a =_{\mathcal{I}} b}{\Gamma,\ a =_{i,\,\mathcal{I}} b} \quad \text{where } \phi \text{ is } read(a, i) = read(b, i)$$

Figure 3. Rules to propagate entailed equations

6.2. Propagating properly entailed disjunctions

Definition 9 (proper entailment of disjunctions) *A disjunction that is entailed when neither of its disjuncts is entailed is said to be* properly entailed.

Incorporating the procedure into the framework of [2] also requires it to have the following property. Let ϕ and ψ be equations whose sides appear in goal Γ. If the procedure reports Γ satisfiable, then Γ cannot properly entail $\phi \vee \psi$. The original procedure for **Arr** does not have this property; an example is the normal goal $a =_{\{i\}} b, a =_{\{j\}} b, read(b, i) = v, read(b, j) = v'$, which entails $i = j \vee a = b$ but neither $i = j$ nor $a = b$. It can be proved, however, that the modified procedure of section 6.1 does have this property.

6.3. Allowing constant arrays

Constant arrays are arrays that store a single value for all indices. The language is extended with function symbols $const_\tau$ for each value sort τ, and the following axiom schema is added:

$$\forall\, x : \tau\, .\, \forall\, i : I\, .\, read(const(x), i) \quad = \quad x$$

The procedure of section 6.1 is modified to obtain a procedure for this extended theory by adding the rules of Figure 4. (const-elim1) is added to both phases, and (const-symm) and (const-elim2) are added to phase 2. To ensure that the conclusion of (const-elim2) entails its premise, the simplifying assumption is made that the interpretation of the type I of indices is infinite. With this modified procedure, goals that are normal with respect to phase 2 may fail to be normal with respect to phase 1. For example, the applications of *const* in the goal $const(write(a, i, v)) = const(b)$

are removed using (const-elim2) in phase 2, but this adds the equation $write(a, i, v) = b$ to the goal, which could be analyzed with the (w-elim) rule of phase 1. So it is necessary to repeat the phases.

$$\text{(const-elim1)} \quad \frac{\Gamma[x]}{\Gamma[read(const(x), i)]}$$

$$\text{(const-symm)} \quad \frac{\Gamma,\ a =_{\mathcal{I}} const(x)}{\Gamma,\ const(x) =_{\mathcal{I}} a}$$

where a is not of the form $const(y)$

$$\text{(const-elim2)} \quad \frac{\Gamma,\ x = y}{\Gamma,\ const(x) =_{\mathcal{I}} const(y)}$$

Figure 4. Rules to treat constant arrays

7. Conclusion

A refutation procedure for an extensional theory of multi-dimensional arrays has been presented and proved correct. The theory **Arr** decided essentially subsumes all previously decided array theories. The procedure is suitable for incorporation into a framework for cooperating decision procedures.

8. Acknowledgements

We thank the anonymous reviewers for their very helpful criticism. The first author was supported during part of this work by a National Science Foundation Graduate Fellowship. Support was also provided in part by NSF contract CCR-9806889-002 and ARPA/AirForce contract F33615-00-C-1693. This paper does not necessarily reflect the position or the policy of the U.S. Government; no official endorsement should be inferred.

References

[1] L. Bachmair and A. Tiwari. Abstract Congruence Closure and Specializations. In D. McAllester, editor, *17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 64–78. Springer-Verlag, 2000.

[2] C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In D. McAllester, editor, *17th International Conference on Computer Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000.

[3] G. Collins and D. Syme. A Theory of Finite Maps. In *Conference on Higher Order Logic Theorem Proving and its Applications*, 1995.

[4] P. Downey and R. Sethi. Assignment Commands with Array References. *Journal of the ACM*, 25(4):652–666, Oct. 1978.

[5] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[6] D. Kaplan. Some Completeness Results in the Mathematical Theory of Computation. *Journal of the ACM*, 15(1):124–34, Jan. 1968.

[7] J. Levitt. *Formal Verification Techniques for Digital Systems*. PhD thesis, Stanford University, 1999.

[8] J. McCarthy. Towards a Mathematical Science of Computation. In *IFIP Congress 62*, 1962.

[9] G. Nelson. Techniques for Program Verification. Technical Report CSL-81-10, Xerox PARC, June 1981.

[10] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[11] S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, 1992.

[12] H. Ruess. Private communication. 2000.

[13] R. Shostak. Deciding combinations of theories. *Journal of the Association for Computing Machinery*, 31(1):1–12, 1984.

[14] A. Stump, D. Dill, J. Giesl, and C. Barrett. On a Very Simple Abstract Higher-Order Congruence Closure Algorithm. *In preparation*, 2000. Available from http://verify.stanford.edu/˜ stump/.

[15] N. Suzuki and D. Jefferson. Verification Decidability of Presburger Array Programs. In *Proceedings of a Conference on Theoretical Computer Science*, 1977. University of Waterloo, Waterloo, Ontario, Canada.