Arrays

Chapter 7

Decision Procedures
An Algorithmic Point of View

D.Kroening       O.Strichman

Revision 1.0
Outline

1 Introduction
   - Definition
   - Basic Operations
   - Syntax
   - Semantics
   - Example

2 Arrays as Uninterpreted Functions

3 A Reduction Algorithm for Array Logic
   - Array Properties
   - A Reduction Algorithm
Arrays are an important data structure:

- “Native” implementation in most processor architectures
- Offered by most programming languages
- $O(1)$ index operation
  E.g., all data structures in Minisat are based on arrays
- Hardware: memories
Formalization

- Mapping from an index type to an element type

- \( T_I \): index type
- \( T_E \): element type
- \( T_A = (T_I \rightarrow T_E) \): array type

- Assumption: there are relations

\[
=_I \subseteq (T_I \times T_I) \quad \text{and} \quad =_E \subseteq (T_E \times T_E)
\]

The subscript is omitted if the type of the operands is clear.

- The theories used to reason about the indices and the elements are called index theory and element theory, respectively.
Basic Operations

Let $a \in T_A$ denote an array.

There are two basic operations on arrays:

1. **Reading**: $a[i]$ is the value of the element that has index $i$

2. **Writing**: the array $a$ where element $i$ has been replaced by $e$ is denoted by $a\{i \leftarrow e\}$
What theory is suitable for the indices?

- Index logic should permit existential and universal quantification:
  
  - “there exists an array element that is zero”
  - “all elements of the array are greater than zero”

- Example: *Presburger arithmetic*, i.e., linear arithmetic over integers with quantification
More About the Index Theory

What theory is suitable for the indices?

- Index logic should permit existential and universal quantification:
  - “there exists an array element that is zero”
  - “all elements of the array are greater than zero”

- Example: *Presburger arithmetic*, i.e., linear arithmetic over integers with quantification

\[ n \]-dimensional arrays:
For \( n \geq 2 \), add \( T_A(n - 1) \) to the element type of \( T_A(n) \).
A Very General Definition of Array Logic

Syntax defined by extending the syntactic rules for the index logic and the element logic

- $atom_I$: atom in the index logic
- $atom_E$: atom in the element logic
- $term_I$: term in the index logic
- $term_E$: term in the element logic
Syntax

\[
\begin{align*}
\text{atom} & : \ atom_I \ | \ atom_E \ | \ \neg \text{atom} \ | \ atom \land atom \ | \\
& \quad \forall \ array-identifier \cdot \ text{atom} \\
\text{term}_A & : \ array-identifier \ | \ \text{term}_A\{term_I \leftarrow term_E\} \\
\text{term}_E & : \ \text{term}_A[term_I]
\end{align*}
\]

Equality between arrays \(a_1\) and \(a_2\): write as \(\forall i. \ a_1[i] = a_2[i]\)
Main axiom:

Axiom (Read-over-write Axiom)

\[ \forall a \in T_a. \forall e \in T_e. \forall i, j \in T_i. \]

\[ a\{i \leftarrow e\}[j] = \begin{cases} 
    e & : \ i = j \\
    a[j] & : \ \text{otherwise}
\end{cases} \]
Program Verification Example I

1 a: array 0..99 of integer;
2 i: integer;

3 for i:=0 to 99 do
4    /* ∀x ∈ N₀. x < i → a[x] = 0 */
5    a[i]:=0;
6    /* ∀x ∈ N₀. x ≤ i → a[x] = 0 */
7    done;
8    /* ∀x ∈ N₀. x ≤ 99 → a[x] = 0 */
Main step of the correctness argument:
invariant in line 7 is maintained by the assignment in line 6

Verification condition:

\[(\forall x \in \mathbb{N}_0. \ x < i \rightarrow a[x] = 0) \land a' = a\{i \leftarrow 0\} \rightarrow (\forall x \in \mathbb{N}_0. \ x \leq i \rightarrow a'[x] = 0)\]
Q: Is this logic decidable?
Decidability

Q: Is this logic decidable?

A: No, even if the combination of the index logic and the element logic is decidable
Arrays as Uninterpreted Functions

Fragment: no quantification over arrays
Arrays as Uninterpreted Functions

Fragment: no quantification over arrays

Arrays are functions! (From indices to elements)
Arrays as Uninterpreted Functions

Fragment: no quantification over arrays

Arrays are functions! (From indices to elements)

Idea: use procedures for uninterpreted functions!
Example

\[(i = j \land a[j] = 'z') \rightarrow a[i] = 'z'\]

'z': read as an integer number
Example

\[(i = j \land a[j] = 'z') \rightarrow a[i] = 'z'\]

'z': read as an integer number

\(F_a: \) uninterpreted function introduced for the array \(a:\)

\[(i = j \land F_a(j) = 'z') \rightarrow F_a(i) = 'z'\]
Example

\[(i = j \land F_a(j) = 'z') \rightarrow F_a(i) = 'z'\]

Apply Bryant’s reduction:

\[(i = j \land F_1^* = 'z') \rightarrow F_2^* = 'z'\]

where

\[
F_1^* = f_1 \quad \text{and} \quad F_2^* = \begin{cases} f_1 : & i = j \\ f_2 : & \text{otherwise} \end{cases}
\]

Prove this using a decision procedure for equality logic.
Array Updates

What about $a\{i \leftarrow e\}$?
Array Updates

What about \( a\{i \leftarrow e\} \)?

1. Replace \( a\{i \leftarrow e\} \) by a fresh variable \( a' \) of array type

2. Add two constraints:
   a) \( a'[i] = e \) for the value that is written,
   b) \( \forall j \neq i. \ a'[j] = a[j] \) for the values that are unchanged.

   Compare to the read-over-write axiom!

This is called the write rule.
Array Updates: Example I

Transform

\[ a\{i \leftarrow e\}[i] \geq e \]

into:

\[ a'[i] = e \implies a'[i] \geq e \]
Transform

\[ a[0] = 10 \rightarrow a\{1 \leftarrow 20\}[0] = 10 \]

into:

\[(a[0] = 10 \land a'[1] = 20 \land (\forall j \neq 1. a'[j] = a[j])) \rightarrow a'[0] = 10 \]
Array Updates: Example II

Transform
\[ a[0] = 10 \rightarrow a\{1 \leftarrow 20\}[0] = 10 \]

into:
\[
( a[0] = 10 \land a'[1] = 20 \land (\forall j \neq 1. \ a'[j] = a[j])) \rightarrow a'[0] = 10
\]

and then replace \( a, a' \):
\[
( F_a(0) = 10 \land F_{a'}(1) = 20 \land (\forall j \neq 1. \ F_{a'}(j) = F_a(j))) \rightarrow F_{a'}(0) = 10
\]
Transform

\[ a[0] = 10 \rightarrow a\{1 \leftarrow 20\}[0] = 10 \]

into:

\[(a[0] = 10 \land a'[1] = 20 \land (\forall j \neq 1. a'[j] = a[j])) \rightarrow a'[0] = 10 \]

and then replace \(a, a'\):

\[(F_a(0) = 10 \land F_a'(1) = 20 \land (\forall j \neq 1. F_a'(j) = F_a(j))) \rightarrow F_a'(0) = 10 \]

Q: Is this decidable in general?
Say Presburger plus uninterpreted functions?
Now: restricted class of array logic formulas in order to obtain decidability.
We consider formulas that are Boolean combinations of array properties.

Definition (array property)

A formula is an array property iff if it is of the form

$$\forall i_1, \ldots, i_k \in T_I. \phi_I(i_1, \ldots, i_k) \rightarrow \phi_V(i_1, \ldots, i_k),$$

and satisfies the following conditions:

1. The predicate $\phi_I$ must be an index guard.
2. The index variables $i_1, \ldots, i_k$ can only be used in array read expressions of the form $a[i_j]$.

The predicate $\phi_V$ is called the value constraint.
### Definition (Index Guard)

A formula is an *index guard* iff it follows the grammar

\[
\text{iguard} : \quad \text{iguard} \land \text{iguard} \mid \text{iguard} \lor \text{iguard} \mid \\
\quad \text{iterm} \leq \text{iterm} \mid \text{iterm} = \text{iterm} \\
\text{iterm} : \quad i_1 \mid \ldots \mid i_k \mid \text{term} \\
\text{term} : \quad \text{integer-constant} \mid \\
\quad \text{integer-constant} \cdot \text{index-identifier} \mid \\
\quad \text{term} + \text{term}
\]

The "*index-identifier*" used in "*term*" must not be one of \(i_1, \ldots, i_k\).
The extensionality rule defines the equality of two arrays $a_1$ and $a_2$ as element-wise equality. Extensionality is an array property:

$$\forall i. \ a_1[i] = a_2[i]$$
The extensionality rule defines the equality of two arrays $a_1$ and $a_2$ as element-wise equality. Extensionality is an array property:

$$\forall i. a_1[i] = a_2[i]$$

How about the array update?

$$a' = a\{ i \leftarrow 0 \}$$

Is this an array property as well?
An array update expression can be replaced by adding two constraints:

\[ a'[i] = 0 \land \forall j \neq i. a'[j] = a[j] \]

The first conjunct is obviously an array property.
An array update expression can be replaced by adding two constraints:

\[ a'[i] = 0 \land \forall j \neq i. \ a'[j] = a[j] \]

The first conjunct is obviously an array property.

The second conjunct can be rewritten as

\[ \forall j. \ (j \leq i - 1 \lor i + 1 \leq j) \longrightarrow a'[j] = a[j] \]
Algorithm

Input: Array property formula $\phi_A$ in NNF
Output: Formula $\phi_{UF}$

1. Apply the write rule to remove all array updates from $\phi_A$.
2. Replace all existential quantifications of the form $\exists i \in T_I. P(i)$ by $P(j)$, where $j$ is a fresh variable.
3. Replace all universal quantifications of the form $\forall i \in T_I. P(i)$ by
   \[
   \bigwedge_{i \in I(\phi)} P(i)
   \]
4. Replace the array read operators by uninterpreted functions and obtain $\phi_{UF}$;
5. return $\phi_{UF}$;
\( \mathcal{I}(\phi) \) denotes the index expressions that \( i \) might possibly be equal to.

Theorem: This set contains the following elements:

1. All expressions used as an array index in \( \phi \) that are not quantified variables.
2. All expressions used inside index guards in \( \phi \) that are not quantified variables.
3. If \( \phi \) contains none of the above, \( \mathcal{I}(\phi) \) is \( \{0\} \) in order to obtain a nonempty set of index expressions.
We prove validity of

$$(\forall x \in \mathbb{N}_0. x < i \rightarrow a[x] = 0) \land a' = a \{ i \leftarrow 0 \} \rightarrow (\forall x \in \mathbb{N}_0. x \leq i \rightarrow a'[x] = 0).$$

That is, we check satisfiability of

$$(\forall x \in \mathbb{N}_0. x < i \rightarrow a[x] = 0) \land a' = a \{ i \leftarrow 0 \} \land (\exists x \in \mathbb{N}_0. x \leq i \land a'[x] \neq 0).$$
Example

Apply write rule:

\[
(\forall x \in \mathbb{N}_0. \ x < i \rightarrow a[x] = 0) \\
\land \ a'[i] = 0 \land \forall j \neq i. \ a'[j] = a[j] \\
\land \ (\exists x \in \mathbb{N}_0. \ x \leq i \land \ a'[x] \neq 0).
\]

Instantiate existential quantifier with a new variable \( z \in \mathbb{N}_0 \):

\[
(\forall x \in \mathbb{N}_0. \ x < i \rightarrow a[x] = 0) \\
\land \ a'[i] = 0 \land \forall j \neq i. \ a'[j] = a[j] \\
\land \ z \leq i \land \ a'[z] \neq 0).
\]
The set $\mathcal{I}$ for our example is $\{i, z\}$.
Replace the two universal quantifications as follows:

\[
(i < i \rightarrow a[i] = 0) \land (z < i \rightarrow a[z] = 0) \\
\land \ a'[i] = 0 \land (i \neq i \rightarrow a'[i] = a[i]) \land (z \neq i \rightarrow a'[z] = a[z]) \\
\land \ z \leq i \land a'[z] \neq 0).
\]

Remove the trivially satisfied conjuncts to obtain

\[
(z < i \rightarrow a[z] = 0) \\
\land \ a'[i] = 0 \land (z \neq i \rightarrow a'[z] = a[z]) \\
\land \ z \leq i \land a'[z] \neq 0).
\]
Replace the arrays by uninterpreted functions:

\[
(z < i \implies F_a(z) = 0) \\
\land \quad F_a'(i) = 0 \land (z \neq i \implies F_a'(z) = F_a(z)) \\
\land \quad z \leq i \land F_a'(z) \neq 0)
\]

By distinguishing the three cases \( z < i, \ z = i, \) and \( z > i, \) it is easy to see that this formula is unsatisfiable.