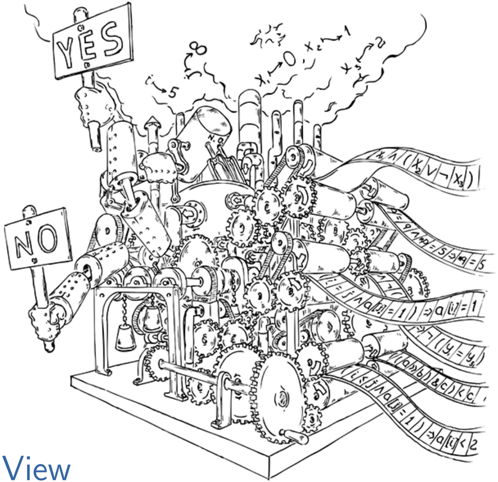
# Arrays

## 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

# Motivation

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

- Mapping from an *index type* to an *element type*

- $T_I$: index type
- $T_E$: element type
- $T_A = (T_I \longrightarrow T_E)$: 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.

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 *Writing*: the array $a$ where element $i$ has been replaced by $e$ is denoted by $a\{i \longleftarrow e\}$

# 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

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

$$
\begin{array}{rcl}
atom & : & atom_I \mid atom_E \mid \neg atom \mid atom \wedge atom \mid \\
 & & \forall\, \textit{array-identifier}\, .\ atom \\
term_A & : & \textit{array-identifier} \mid term_A\{term_I \longleftarrow term_E\} \\
term_E & : & term_A\,[\,term_I\,]
\end{array}
$$

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)**

$$\forall a \in T_A.\ \forall e \in T_E.\ \forall i, j \in T_I.$$

$$a\{i \longleftarrow e\}[j] = \begin{cases} e & : \quad i = j \\ a[j] & : \quad \textit{otherwise}\,. \end{cases}$$

# Program Verification Example I

```
a:  array 0..99 of integer;
i:  integer;

for i:=0 to 99 do
        /* $\forall x \in \mathbb{N}_0.\ x < i \longrightarrow a[x] = 0$ */
        a[i]:=0;
        /* $\forall x \in \mathbb{N}_0.\ x \leq i \longrightarrow a[x] = 0$ */
done;
/* $\forall x \in \mathbb{N}_0.\ x \leq 99 \longrightarrow a[x] = 0$ */
```

Main step of the correctness argument:
invariant in line 7 is maintained by the assignment in line 6

Verification condition:

$$
\begin{array}{rl}
 & (\forall x \in \mathbb{N}_0.\ x < i \longrightarrow \mathsf{a}[x] = 0) \\
\wedge & \mathsf{a}' = \mathsf{a}\{i \longleftarrow 0\} \\
\longrightarrow & (\forall x \in \mathbb{N}_0.\ x \leq i \longrightarrow \mathsf{a}'[x] = 0)
\end{array}
$$

Q: Is this logic decidable?

Q: Is this logic decidable?

A: No, even if the combination of the index logic and the element logic is decidable

Fragment: no quantification over arrays

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

$$(i = j \wedge a[j] = \texttt{'z'}) \longrightarrow a[i] = \texttt{'z'}$$

`'z'`: read as an integer number

# Example

$$(i = j \land a[j] = \texttt{'z'}) \longrightarrow a[i] = \texttt{'z'}$$

`'z'`: read as an integer number

$F_a$: uninterpreted function introduced for the array $a$:

$$(i = j \land F_a(j) = \texttt{'z'}) \longrightarrow F_a(i) = \texttt{'z'}$$

# Example

$$(i = j \wedge F_a(j) = \texttt{'z'}) \longrightarrow F_a(i) = \texttt{'z'}$$

Apply Bryant's reduction:

$$(i = j \wedge F_1^* = \texttt{'z'}) \longrightarrow F_2^* = \texttt{'z'}$$

where

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

Prove this using a decision procedure for equality logic.

# Array Updates

What about $a\{i \longleftarrow e\}$?

What about $a\{i \longleftarrow e\}$?

1. Replace $a\{i \longleftarrow e\}$ 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

$$a\{i \longleftarrow e\}[i] \geq e$$

into:

$$a'[i] = e \longrightarrow a'[i] \geq e$$

# Array Updates: Example II

Transform

$$a[0] = 10 \longrightarrow a\{1 \longleftarrow 20\}[0] = 10$$

into:

$$(a[0] = 10 \wedge a'[1] = 20 \wedge (\forall j \neq 1.\ a'[j] = a[j])) \longrightarrow a'[0] = 10$$

# Array Updates: Example II

Transform

$$a[0] = 10 \longrightarrow a\{1 \longleftarrow 20\}[0] = 10$$

into:

$$(a[0] = 10 \wedge a'[1] = 20 \wedge (\forall j \neq 1.\ a'[j] = a[j])) \longrightarrow a'[0] = 10$$

and then replace $a$, $a'$:

$$(F_a(0) = 10 \wedge F_{a'}(1) = 20 \wedge (\forall j \neq 1.\ F_{a'}(j) = F_a(j))) \longrightarrow F_{a'}(0) = 10$$

# Array Updates: Example II

Transform

$$a[0] = 10 \longrightarrow a\{1 \longleftarrow 20\}[0] = 10$$

into:

$$(a[0] = 10 \wedge a'[1] = 20 \wedge (\forall j \neq 1.\ a'[j] = a[j])) \longrightarrow a'[0] = 10$$

and then replace $a$, $a'$:

$$(F_a(0) = 10 \wedge F_{a'}(1) = 20 \wedge (\forall j \neq 1.\ F_{a'}(j) = F_a(j))) \longrightarrow F_{a'}(0) = 10$$

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

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

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*.

# Index Guards

## Definition (Index Guard)

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

$$
\begin{array}{rcl}
iguard & : & iguard \wedge iguard \,|\, iguard \vee iguard \,| \\
 & & iterm \leq iterm \,|\, iterm = iterm \\
iterm & : & i_1 \,|\, \ldots \,|\, i_k \,|\, term \\
term & : & \textit{integer-constant} \,| \\
 & & \textit{integer-constant} \cdot \textit{index-identifier} \,| \\
 & & term + term
\end{array}
$$

The "*index-identifier*" used in "$term$" must not be one of $i_1, \ldots, i_k$.

# Array Properties: Example

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

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

# Array Properties: Example

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?

$$\mathsf{a}' = \mathsf{a}\{i \longleftarrow 0\}$$

Is this an array property as well?

# Array Properties: Array Update

An array update expression can be replaced by adding two constraints:

$$a'[i] = 0 \quad \wedge \quad \forall j \neq i.\ a'[j] = a[j]$$

The first conjunct is obviously an array property.

An array update expression can be replaced by adding two constraints:

$$a'[i] = 0 \quad \wedge \quad \forall j \neq i.\ a'[j] = a[j]$$

The first conjunct is obviously an array property.

The second conjunct can be rewritten as

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

# 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
$$\bigwedge_{i \in \mathcal{I}(\phi)} P(i)\ .$$
4. Replace the array read operators by uninterpreted functions and obtain $\phi_{UF}$;
5. **return** $\phi_{UF}$;

# The Set I

$\mathcal{I}(\phi)$ 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.

# Example

We prove validity of

$$\begin{array}{ll} & (\forall x \in \mathbb{N}_0.\ x < i \longrightarrow \mathsf{a}[x] = 0) \\ \wedge & \mathsf{a}' = \mathsf{a}\{i \longleftarrow 0\} \\ \longrightarrow & (\forall x \in \mathbb{N}_0.\ x \leq i \longrightarrow \mathsf{a}'[x] = 0)\ . \end{array}$$

That is, we check satisfiability of

$$\begin{array}{ll} & (\forall x \in \mathbb{N}_0.\ x < i \longrightarrow \mathsf{a}[x] = 0) \\ \wedge & \mathsf{a}' = \mathsf{a}\{i \longleftarrow 0\} \\ \wedge & (\exists x \in \mathbb{N}_0.\ x \leq i \wedge \mathsf{a}'[x] \neq 0)\ . \end{array}$$

# Example

Apply write rule:

$$
\begin{array}{ll}
 & (\forall x \in \mathbb{N}_0.\ x < i \longrightarrow \mathsf{a}[x] = 0) \\
\wedge & \mathsf{a}'[i] = 0 \wedge \forall j \neq i.\ \mathsf{a}'[j] = \mathsf{a}[j] \\
\wedge & (\exists x \in \mathbb{N}_0.\ x \leq i \wedge \mathsf{a}'[x] \neq 0)\ .
\end{array}
$$

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

$$
\begin{array}{ll}
 & (\forall x \in \mathbb{N}_0.\ x < i \longrightarrow \mathsf{a}[x] = 0) \\
\wedge & \mathsf{a}'[i] = 0 \wedge \forall j \neq i.\ \mathsf{a}'[j] = \mathsf{a}[j] \\
\wedge & z \leq i \wedge \mathsf{a}'[z] \neq 0)\ .
\end{array}
$$

# Example

The set $\mathcal{I}$ for our example is $\{i, z\}$.
Replace the two universal quantifications as follows:

$$\begin{array}{ll} & (i < i \longrightarrow \mathsf{a}[i] = 0) \wedge (z < i \longrightarrow \mathsf{a}[z] = 0) \\ \wedge & \mathsf{a}'[i] = 0 \wedge (i \neq i \longrightarrow \mathsf{a}'[i] = \mathsf{a}[i]) \wedge (z \neq i \longrightarrow \mathsf{a}'[z] = \mathsf{a}[z]) \\ \wedge & z \leq i \wedge \mathsf{a}'[z] \neq 0) \ . \end{array}$$

Remove the trivially satisfied conjuncts to obtain

$$\begin{array}{ll} & (z < i \longrightarrow \mathsf{a}[z] = 0) \\ \wedge & \mathsf{a}'[i] = 0 \wedge (z \neq i \longrightarrow \mathsf{a}'[z] = \mathsf{a}[z]) \\ \wedge & z \leq i \wedge \mathsf{a}'[z] \neq 0) \ . \end{array}$$

# Example

Replace the arrays by uninterpreted functions:

$$
\begin{array}{ll}
 & (z < i \longrightarrow F_a(z) = 0) \\
\wedge & F_{a'}(i) = 0 \wedge (z \neq i \longrightarrow F_{a'}(z) = F_a(z)) \\
\wedge & z \leq i \wedge F_{a'}(z) \neq 0) \; .
\end{array}
$$

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