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

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:

- **Reading**: $a[i]$ is the value of the element that has index $i$
- **Writing**: the array $a$ where element $i$ has been replaced by $e$ is denoted by $a(i \leftarrow e)$

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

$n$-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

- \textit{atom}_I: atom in the index logic
- \textit{atom}_E: atom in the element logic
- \textit{term}_I: term in the index logic
- \textit{term}_E: term in the element logic

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

Program Verification Example I

1. \(a: \text{array } 0..99 \text{ of integer;}
2. i: \text{integer;}
3. for i:=0 to 99 do
4.   /* \(\forall \ x \in \mathbb{N}_0. \ x < i \ \square \rightarrow \ a[x] = 0\) */
5.   a[i]:=0;
6.   /* \(\forall \ x \in \mathbb{N}_0. \ x \leq i \ \square \rightarrow \ a[x] = 0\) */
7. done;
8. /* \(\forall \ x \in \mathbb{N}_0. \ x \leq 99 \ \square \rightarrow \ a[x] = 0\) */

Program Verification Example II

Main step of the correctness argument:

Verification condition:

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

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 are functions! (From indices to elements)

Idea: use procedures for uninterpreted functions!

Decision Procedures – Arrays 13

Example

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

'z': read as an integer number

Fa: uninterpreted function introduced for the array a:

\[(i = j \land Fa(j) = 'z') \rightarrow Fa(i) = 'z'

Decision Procedures – Arrays 14

Array Updates

What about \(a\{i ← □ e\}\)?

- Replace \(a\{i ← □ e\}\) by a fresh variable \(a'\) of array type
- 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.

Prove this using a decision procedure for equality logic.

Array Updates: Example I

Transform

\[a\{i ← □ e\}[i] ≥ e\]

into:

\[a'[i] = e \rightarrow a'[i] ≥ e\]

Array Updates: Example II

Transform

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

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':

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

Q: Is this decidable in general?
Say Presburger plus uninterpreted functions?
**Decision Procedures – Arrays**

**Array Properties**

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 it is of the form

\[ \forall i_1, \ldots, i_k \in T_I, \phi_i(t_1, \ldots, t_k) \rightarrow \phi_V(t_1, \ldots, t_k), \]

and satisfies the following conditions:

1. The predicate \( \phi_V \) 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 if follows the grammar

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

The “index-identifier” used in “term” must not be one of \( i_1, \ldots, i_k \).

**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 \( P(i) \), where \( i \) 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 \( \phi_{UF} \);

**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?

\[ a' = a(i \leftarrow 0) \]

Is this an array property as well?

**Array Properties: Array Update**

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

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

The first conjunct is obviously an array property.

The second conjunct can be rewritten as

\[ \forall j, (j \leq i - 1 \lor i + 1 \leq j) \rightarrow a'[j] = a[j] \]

**The Set \( I(\phi) \)**

\( I(\phi) \) 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, \( I(\phi) \) is \( \{0\} \) in order to obtain a nonempty set of index expressions.
Example

We prove validity of
\[
(\forall x \in \mathbb{N}_0. x < i \implies a[x] = 0) \\
\land \quad a' = a[i \mapsto 0] \\
\implies (\forall x \in \mathbb{N}_0. x \leq i \implies a'[x] = 0).
\]

That is, we check satisfiability of
\[
(\forall x \in \mathbb{N}_0. x < i \implies a[x] = 0) \\
\land \quad a' = a[i \mapsto 0] \\
\land \quad (\exists x \in \mathbb{N}_0. x \leq i \land a'[x] \neq 0).
\]

Decision Procedures – Arrays 25

Example

Apply write rule:
\[
(\forall x \in \mathbb{N}_0. x < i \implies a[x] = 0) \\
\land \quad a'[i] = 0 \\
\land \quad (\forall j \neq i. a'[j] = a[j]) \\
\land \quad z \leq i \land a'[z] \neq 0.
\]

Instantiate existential quantifier with a new variable \(z \in \mathbb{N}_0\):
\[
(\forall x \in \mathbb{N}_0. x < i \implies a[x] = 0) \\
\land \quad a'[i] = 0 \\
\land \quad (\forall j \neq i. a'[j] = a[j]) \\
\land \quad z \leq i \land a'[z] \neq 0.
\]

Decision Procedures – Arrays 26

Example

The set \(I\) for our example is \(\{i, z\}\).
Replace the two universal quantifications as follows:
\[
(i < i \implies a[i] = 0) \\
\land \quad z < i \implies a'[z] = 0) \\
\land \quad z \neq i \implies a'[z] = a[z]) \\
\land \quad z \leq i \land a'[z] \neq 0.
\]

Remove the trivially satisfied conjuncts to obtain
\[
(z < i \implies a[z] = 0) \\
\land \quad a'[i] = 0 \land (z \neq i \implies a'[z] = a[z]) \\
\land \quad z \leq i \land a'[z] \neq 0.
\]

Decision Procedures – Arrays 27

Example

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.

Decision Procedures – Arrays 28