Motivation

Arrays are important data structures
- "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

Outline

- Introduction
  - Definition
  - Basic Operations
  - Syntax Semantics
- Arrays as Uninterpreted Functions
- A Reduction Algorithm for Array Logic
  - Array Properties
  - A Reduction Algorithm

Formalization

- Mapping from index type to element type
  - $T_I$: index type
  - $T_E$: element type
  - $T_A = (T_I \rightarrow T_E)$: array type
- Assumptions:
  - equality relations exist
    - $=I \subseteq (T_I \times T_I)$ and $=E \subseteq (T_E \times T_E)$
- The theories used to reason about the indices and the elements are called the index theory and element theory, respectively.
Basic Operations

Let $a \in T_A$ denote an array.

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- **Writing**: the array $a$ where element $i$ has been replaced by $e$ is denoted by $a[i \leftarrow e]$

On the Index & Element Theories

- Index logic should permit existential and universal quantification:
  - “there exists an array element that is zero”
  - “the elements of the array are sorted” (requires the element logic to have a $<$-operation)
  - Example: Presburger arithmetic, i.e., linear arithmetic over integers with quantification
- For $n$-dimensional arrays:
  - $T_A(1) = (T_I \rightarrow T_E)$
  - For $n \geq 2$, $T_A(n) = (T_I \rightarrow (T_E \cup T_A(n-1)))$

A Very General 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

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

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

Main step of the correctness argument:
the invariant is maintained by the body of the loop

\[ \text{a}_1 := \text{a}_0 \{ i \leftarrow 0 \} ; \]
\[ \text{i}_1 := \text{i}_0 + 1 ; \]
\[ \text{a}_1 \{ \text{i} \leftarrow \text{i}_1 \} \]

Decidability

\[ \text{Is this logic decidable?} \]

\[ \text{No} \]

- Even if the combination of the index logic and the element logic are!
Arrays as Uninterpreted Functions

- Fragment: no quantification over arrays
- Arrays are functions
  - (From indices to elements)
- Idea: use procedures for deciding uninterpreted functions!

Example

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

“z”: read as an integer number

\[F_a: \text{introduced for the array } a:\]

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

Array Updates

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

The write rule:
- Replace with a fresh variable \(a'\) of array type
- Assert
  - For the value that is written: \(a'[i] = e\)
  - For the values that are unchanged:
    \[\forall j : j \neq i \rightarrow a'[j] = a[j]\]
Array Updates: Example

Transform:
\[
(a'[i] \leftarrow e)[i] \geq e
\]

Into:
\[
(a'[i] = e \land (\forall j . j \neq i \rightarrow a'[j] = a[j])) \rightarrow \\
    a'[i] \geq e
\]

Array Updates: Example (cont.)

Transform:
\[
((\forall j . a[j] > 0) \land (a'[i] = a[i]+1) \land \\
    (\forall j . j \neq i \rightarrow a'[j] = a[j])) \rightarrow \\
    (\forall j . a'[j] > 0)
\]

Into:
\[
((\forall j . F_a[j] > 0) \land (F_a[i] = F_a[i]+1) \land \\
    (\forall j . j \neq i \rightarrow F_a'[j] = F_a'[j])) \rightarrow \\
    (\forall j . F_a'[j] > 0)
\]

Arrays as Uninterpreted Functions

Transform: no quantification over arrays

Reduces to combinations of the index logic znc uninterpreted functions, assuming index logic allows quantification

But combination of Presburger arithmetic and uninterpreted functions undecidable 😞
Array Properties

For decidability, we consider boolean combinations of array properties.

Definition (array property)
A formula is an array property iff it is of the form
\[ \forall i_1, \ldots, i_k \in T_I. \phi_I(i_1, \ldots, i_k) \longrightarrow \phi_V(i_1, \ldots, i_k), \]
and satisfies the following conditions:
- The predicate \( \phi_I \) must be an index guard.
- 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.

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

- What about array update?
  \[ a' = a \{ i \leftarrow 0 \} \]

- Array update can be replaced by two expressions:
  \[ a'[i] = 0 \land \forall j. (j \leq i - 1 \lor i + 1 \leq j) \rightarrow a'[j] = a[j] \]
Algorithm
Input: Array property formula $\varphi_A$ in NNF
Output: Formula $\varphi_{UF}$

1. Apply write rule to remove all array updates from $\varphi_A$.
2. Replace existential quantifications of the form $\exists i \in T_i. P(i)$ by $P(j)$, where $j$ is a fresh variable.
3. Replace universal quantifications of the form $\forall i \in T_i. P(i)$ by $\bigwedge_{i \in \mathcal{I}(\varphi)} P(i)$
4. Replace array read operations by uninterpreted functions to obtain $\varphi_{UF}$
5. return $\varphi_{UF}$

Example
We prove validity of:

$$((\forall x . x < i \rightarrow a[x] = 0) \land a'[i] = 0) \rightarrow (\forall x . x \leq i \rightarrow a'[x] = 0)$$

That is, we check satisfiability of:

$$((\forall x . x < i \rightarrow a[x] = 0) \land a'[i] = 0) \land$$

$$(\exists x . x \leq i \land a'[x] \neq 0)$$

Algorithm

- Where
  - $\mathcal{I}(\varphi)$ denotes the index expressions in $\varphi$ that the quantified variable $i$ might possibly be equal to.
  - $\varphi$ denotes the transformed $\varphi$, after step 2
- $\mathcal{I}(\varphi)$ contains:
  - All index expressions used as an array index in $\varphi$ that are not quantified variables.
  - All index expressions used inside index guards in $\varphi$ that are not quantified variables.
  - If $\varphi$ contains none of the above, then $\mathcal{I}(\varphi)$ is $\{0\}$ in order to obtain a nonempty set of index expressions.

Example
Apply write rule:

$$((\forall x . x < i \rightarrow a[x] = 0) \land a'[i] = 0 \land$$

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

$$((\exists x . x \leq i \land a'[x] \neq 0)$$

Replace existential quantifier with fresh variable $z$:

$$((\forall x . x < i \rightarrow a[x] = 0) \land a'[i] = 0 \land$$

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

$$(\exists x . x \leq i \land a'[x] \neq 0) \land$$

$$(\exists z . z \leq i \land a'[z] \neq 0)$$
Example (cont.)

The set $I(\varphi)$ for our example is $\{i, z, i - 1, i + 1\}$.
Replace the two universal quantifications:

\[
(\forall a \forall z \bullet (i < a[i] = 0) \land (z < a[z] = 0) \land
\]

\[
(\forall a \forall z \bullet (i - 1 < a[i - 1] = 0) \land (i + 1 < a[i + 1] = 0) \land
\]

\[
a'[i] = 0 \land
\]

\[
((i \leq i - 1 \lor i + 1 \leq i) \rightarrow a'[i] = a[i]) \land
\]

\[
((z \leq i - 1 \lor i + 1 \leq z) \rightarrow a'[z] = a[z]) \land
\]

\[
((i - 1 \leq i - 1 \lor i + 1 \leq i - 1) \rightarrow a'[i - 1] = a[i - 1]) \land
\]

\[
((i + 1 \leq i - 1 \lor i + 1 \leq i + 1) \rightarrow a'[i + 1] = a[i + 1]) \land
\]

\[
z \leq i \land a'[z] \neq 0
\]

Example (cont.)

Remove the trivially satisfied conjuncts to obtain:

\[
(z < i \rightarrow a[i] = 0) \land a[i - 1] = 0 \land a'[i] = 0 \land
\]

\[
((z \leq i - 1 \lor i + 1 \leq z) \rightarrow a'[z] = a[z]) \land
\]

\[
a'[i - 1] = a[i - 1] \land a'[i + 1] = a[i + 1] \land
\]

\[
z \leq i \land a'[z] \neq 0
\]

Replace the arrays by uninterpreted functions:

\[
(z < i \rightarrow F_a[z] = 0) \land F_a[i - 1] = 0 \land F_a'[i] = 0 \land
\]

\[
((z \leq i - 1 \lor i + 1 \leq z) \rightarrow F_a'[z] = F_a[z]) \land
\]

\[
F_a'[i - 1] = F_a[i - 1] \land F_a'[i + 1] = F_a[i + 1] \land
\]

\[
z \leq i \land F_a'[z] \neq 0
\]

Example (cont.)

In case $z < i$:

\[
(z < i \rightarrow F_a[z] = 0) \land F_a[i - 1] = 0 \land F_a'[i] = 0 \land
\]

\[
((z \leq i - 1 \lor i + 1 \leq z) \rightarrow F_a'[z] = F_a[z]) \land
\]

\[
F_a'[i - 1] = F_a[i - 1] \land F_a'[i + 1] = F_a[i + 1] \land
\]

\[
z \leq i \land F_a'[z] \neq 0
\]

The VC becomes:

\[
F_a'[z] = 0 \land F_a[i - 1] = 0 \land F_a'[i] = 0 \land
\]

\[
F_a'[z] = F_a[z] \land F_a'[i - 1] = F_a[i - 1] \land
\]

\[
F_a'[i + 1] = F_a[i + 1] \land F_a'[z] \neq 0
\]

Example (cont.)

In case $z = i$:

\[
(z < i \rightarrow F_a[z] = 0) \land F_a[i - 1] = 0 \land F_a'[i] = 0 \land
\]

\[
((z \leq i - 1 \lor i + 1 \leq z) \rightarrow F_a'[z] = F_a[z]) \land
\]

\[
F_a'[i - 1] = F_a[i - 1] \land F_a'[i + 1] = F_a[i + 1] \land
\]

\[
z \leq i \land F_a'[z] \neq 0
\]

The VC becomes:

\[
F_a'[i - 1] = 0 \land F_a[i] = 0 \land F_a'[i - 1] = F_a[i - 1] \land
\]

\[
F_a'[i + 1] = F_a[i + 1] \land F_a'[z] \neq 0
\]
Example (cont.)

In case $z > i$:

$$(z < i \rightarrow F_a[z] = 0) \land F_a[i - 1] = 0 \land F_a[i] = 0 \land$$

$$(z \leq i - 1 \lor i + 1 \leq z) \rightarrow F_a[z] = F_a[z]) \land$$

$$F_a[i - 1] = F_a[i - 1] \land F_a[i + 1] = F_a[i + 1] \land$$

$$z \leq i \land F_a[z] \neq 0$$

The VC becomes:

false