Array Logic

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

07 April 2025

Outline

Array Logic (BM Ch 11, KS Ch 7)

- Two-sorted first-order logic. One sort is domain of integers, the other is domain of arrays (modelled as functions from integers to integers).
- Signature of the logic includes "array read" function: $read(a, i)$ or "$a[i]$" (value stored at position i in a),
- and "array write" function $write(a, i, v)$ or $a\langle i \triangleleft v\rangle$ (returns new array a' which coincides with a except at position i where it has value e).

Example formula

$$[(x < m) \wedge$$
$$(0 \leq i) \wedge \forall k(((0 \leq k) \wedge (k < i)) \implies (a[k] \leq m)) \wedge$$
$$a' = a\langle i \triangleleft x\rangle]$$
$$\implies \forall k(((0 \leq k) \wedge (k \leq i)) \implies (a'[k] \leq m)).$$

Application: Symbolic Execution of Array Programs

Illustrating symbolic execution for integer programs: Are there input values of x and y that lead to *error* being executed?

```
// input x, y
int z = 2 * y;
z = z + x;
if (x < y)
  if (z == 12)
    error();
  ...
...
```

Is

$$x_0 < y_0 \wedge 2y_0 + x_0 = 12$$

satisfiable?

Application: Symbolic Execution of Array Programs

Are there input arrays a, b and integers i_1, i_2, j, v_1 that lead to *error* being executed?

```
// input array a, i1, ...
...
if (i1 == j)
  ...
  if (i1 == i2)
    ...
  else if (a[j] == v1)
    b[j] := a[j];
    a[i1] := v1;
    a[i2] := v2;
    if (a[j] != b[j])
      error();
    ...
```

Is

$$i_1 = j \wedge i_1 \neq i_2 \wedge a_0[j] = v_1 \ \wedge$$
$$(a_0\langle i_1 \triangleleft v_1\rangle\langle i_2 \triangleleft v_2\rangle)[j] \neq (b_0\langle j \triangleleft a_0[j]\rangle)[j]$$

satisfiable?

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 1):

```
int m = -1;
for (i = 0; i < N; i++)
  if (m < a[i])
    m := a[i];
// assert for each k: (0 <= k < N) => a[k] <= m
```

Adequate loop invariant for this program?

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 1):

```
int m = -1;
for (i = 0; i < N; i++)
  if (m < a[i])
    m := a[i];
// assert for each k: (0 <= k < N) => a[k] <= m
```

Adequate loop invariant for this program?

$$\forall k((0 \leq k < i) \implies (a[k] \leq m))$$

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 1):

```
int m = -1;
for (i = 0; i < N; i++)
  if (m < a[i])
    m := a[i];
// assert for each k: (0 <= k < N) => a[k] <= m
```

Adequate loop invariant for this program?

$$\forall k((0 \leq k < i) \implies (a[k] \leq m))$$

One of the verification conditions: Is the formula $\forall a \forall N \forall m \forall i$:

$$\begin{array}{ll} [& \forall k((0 \leq k < i) \implies a[k] \leq m) \wedge \\ & (i < N) \wedge (i' = i + 1) \wedge m < a[i] \wedge m' = a[i] \,] \implies \\ \forall k & ((0 \leq k < i') \implies a[k] \leq m). \end{array}$$

valid?

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 2):

```
for (i = 0; i < N; i++)
  a[i] := 0;
// assert for each k: (0 <= k < N) => a[k] = 0
```

What is an adequate loop invariant for this program?

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 2):

```
for (i = 0; i < N; i++)
  a[i] := 0;
// assert for each k: (0 <= k < N) => a[k] = 0
```

What is an adequate loop invariant for this program?

$$\forall k((0 \leq k < i) \implies (a[k] = 0))$$

Application: Verifying Array Programs

Floyd-Hoare style verification of array programs (Example 2):

```
for (i = 0; i < N; i++)
  a[i] := 0;
// assert for each k: (0 <= k < N) => a[k] = 0
```

What is an adequate loop invariant for this program?

$$\forall k((0 \leq k < i) \implies (a[k] = 0))$$

One of the verification conditions: Is the formula $\forall a \forall N \forall i$:

$$[\forall k((0 \leq k < i) \implies (a[k] = 0)) \wedge (i < N) \wedge (i' = i + 1) \wedge a' = a\langle i \triangleleft 0 \rangle] \implies \forall k((0 \leq k < i')) \implies (a'[k] = 0)).$$

valid?

Basic Array Logic [BM Sec 9.5]

Two-Sorted First-Order Logic, with FO signature

$$\Sigma_A = (\cdot[\cdot], \cdot\langle \cdot \triangleleft \cdot \rangle)$$

- Array-Term (a): $a \mid a\langle t \triangleleft t \rangle$
- Value-Term (t): $x \mid a[t]$
- Atomic-Formula: Value-Term = Value-Term |
  ~~Array-Term = Array-Term~~
- Formula: Atomic-Formula $\mid \exists x(\ldots) \mid \exists a(\ldots) \mid$
  Boolean combination of Formulas

Interpreted in sorts integers ($\mathbb{Z}$) and arrays ($\mathbb{Z} \rightarrow \mathbb{Z}$).

Example $\mathrm{FO}(\Sigma_A)$ formula

$$[\forall i(a\langle k \triangleleft v \rangle[i] = a[i])] \implies a[k] = v.$$

Note: equality of array-terms $a = b$ is definable as $\forall i(a[i] = b[i])$.

General Array Logic [BM Sec 9.5]

FO Signature

$$\Sigma_A^{\mathbb{Z}} = (\cdot[\cdot], \cdot\langle\cdot \triangleleft \cdot\rangle), 0, 1, +, <)$$

- Variables $x, y, \ldots$ of sort Integers, and $a, b, \ldots$ of sort arrays.
- Array-Term (a): $b \mid a\langle t \triangleleft t\rangle$
- Value-Term (t): $0 \mid 1 \mid x \mid a[t] \mid t + t'$
- Atomic-Formula: Value-Term $<$ Value-Term $\mid$
  Value-Term $=$ Value-Term $\mid$
  ~~Array-Term $=$ Array-Term~~
- Formula: Atomic-Formula $\mid \exists x(\ldots) \mid \exists a(\ldots) \mid$
  Boolean combination of Formulas

Example $FO(\Sigma_A^{\mathbb{Z}})$ formula

$$\forall a \forall b \forall i \forall j (0 < i < j \implies a[i] \leq b[j])$$

General Array Logic is undecidable [Bradley, Manna, Sipma VMCAI 2006]

- A linear loop program is of the form

int $x_1, \ldots, x_n$;
$x_1, \ldots, x_n := c_1, \ldots, c_n$; // initialization
while ($x_1 \geq 0$) {
  if
    true -> x := $A_1 \cdot$ x;
    true -> x := $A_2 \cdot$ x;
    ...
    true -> x := $A_m \cdot$ x;
  fi
}

- A linear loop program terminates if all its non-deterministic executions terminate.
- Problem of deciding whether a linear loop program terminates is undecidable (no algorithm/decision-procedure can exist)
- Reduce termination of linear loop program to satisfiability of array logic.

Reduction

Given a linear loop program P, construct array logic formula φ_P with array variables $a_1, \ldots a_n$:

$$\exists a_1 \cdots \exists a_n \exists z \forall i \exists j \quad (\quad \begin{array}{l} a_1[z] \geq 0 \ \wedge \\ \bigwedge_{k=1}^{n} a_k[z] = c_i \ \wedge \\ \bigvee_{l=1}^{m} \rho_l(i,j) \ \wedge \\ a_1[j] \geq 0), \end{array}$$

where $\rho_l(i,j)$ is the formula:

$$\begin{array}{rcl} A_l(1,1) \cdot a_1[i] + \cdots + A_l(1,n) \cdot a_n[i] & = & a_1[j] \ \wedge \\ \cdots & & \wedge \\ A_l(n,1) \cdot a_1[i] + \cdots + A_l(n,n) \cdot a_n[i] & = & a_n[j]. \end{array}$$

φ_P says that program P has a non-terminating execution.

Quantifier-Free Basic Array Logic [BM Sec 9.5]

Consider array logic signature without arithmetic:

$$\Sigma_A = (\cdot[\cdot], \cdot\langle \cdot \triangleleft \cdot \rangle)$$

Consider quantifier-free formulas over Σ_A.

- Array-Term (a): $b \mid a\langle t \triangleleft t \rangle$
- Value-Term (t): $x \mid a[t]$
- Atomic-Formula: Value-Term $=$ Value-Term
- Formula: Boolean combination of Atomic-Formulas

Example $\mathrm{QF}(\Sigma_A)$ formula

$$i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a\langle i_1 \triangleleft v_1 \rangle \langle i_2 \triangleleft v_2 \rangle [j] \neq a[j]$$

Quantifier-Free Array Logic [BM Sec 9.5]

Reduce to EUF by using the "read-over-write" rule: Repeatedly replace $F(\cdots a\langle i \triangleleft v\rangle[j] \cdots)$ by

$$
\begin{array}{l}
(i = j) \wedge F(\cdots v \cdots) \vee \\
(i \neq j) \wedge F(\cdots a[j] \cdots).
\end{array}
$$

If no array writes then replace array variables a by functions f_a and array-reads $a[i]$ by $f_a(i)$ to get an EUF formula. Use decision procedure for EUF.

Example

Check satisfiability of

$$i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge a\langle i_1 \triangleleft v_1\rangle\langle i_2 \triangleleft v_2\rangle[j] \neq a[j]$$

$$\begin{aligned}
\equiv \quad & (i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge i_2 = j \wedge v_2 \neq a[j]) \vee \\
& (i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge i_2 \neq j \wedge a\langle i_1 \triangleleft v_1\rangle[j] \neq a[j]) \\
\equiv \quad & (i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge i_2 = j \wedge v_2 \neq a[j]) \vee \\
& (i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge i_2 \neq j \wedge i_1 = j \wedge v_1 \neq a[j]) \vee \\
& (i_1 = j \wedge i_1 \neq i_2 \wedge a[j] = v_1 \wedge i_2 \neq j \wedge i_1 \neq j \wedge a[j] \neq a[j]).
\end{aligned}$$

Check satisfiability using EUF procedure (like Shostak on each disjunct).

Exercise

Check satisfiability of

$$a[x] = v \land x \neq y \land a\langle y \lhd u \rangle[x] \neq v.$$

Array Property Formulas and Array Property Fragment [BM Sec 11.1]

Unfortunately, the use of universal quantification must be restricted to avoid undecidability (see Section 11.4 for further discussion). An **array property** is a Σ_A-formula of the form

$$\forall \bar{i}.\ F[\bar{i}] \ \rightarrow\ G[\bar{i}]$$

in which $\bar{i}$ is a list of variables, and $F[\bar{i}]$ and $G[\bar{i}]$ are the **index guard** and the **value constraint**, respectively. The index guard $F[\bar{i}]$ is any Σ_A-formula that is syntactically constructed according to the following grammar:

$$\begin{array}{rcl} \text{iguard} & \rightarrow & \text{iguard} \wedge \text{iguard} \mid \text{iguard} \vee \text{iguard} \mid \text{atom} \\ \text{atom} & \rightarrow & \text{var} = \text{var} \mid \mathit{evar} \neq \text{var} \mid \text{var} \neq \mathit{evar} \mid \top \\ \text{var} & \rightarrow & \mathit{evar} \mid \mathit{uvar} \end{array}$$

where *uvar* is any universally quantified index variable, and *evar* is any constant or unquantified (that is, implicitly existentially quantified) variable.

Additionally, a universally quantified index can occur in a value constraint $G[\bar{i}]$ only in a read $a[i]$, where a is an array term. The read cannot be nested; for example, $a[b[i]]$ is not allowed.

The array property fragment of T_A then consists of formulae that are Boolean combinations of quantifier-free Σ_A-formulae and array properties.

Array Property Fragment [BM Sec 11.1]

Array Property Formulas:

$$\forall \bar{i}(F(\bar{i}) \Rightarrow G(\bar{i}))$$

with above restrictions on index guard F and value constraint G.

Array Property Fragment:

Boolean combinations of

- Quantifier-Free Basic Array Formulas ($\mathrm{QF}(\Sigma_A)$).
- Array Property Formulas.

Reduction Procedure for $\mathrm{APF}(\Sigma_A)$

1. Put given formula F in Negation Normal Form (NNF)
2. Remove array writes (update terms) by replacing $F(a\langle i \triangleleft v\rangle)$ by
$$F(a') \;\wedge\; a'[i] = v \;\wedge\; \forall j (j \neq i \rightarrow a'[j] = a[j])$$
3. Remove existential quantification: Replace $F(\exists i G(i))$ by $F(G(j))$ for a fresh variable j. (Note that $\exists i$ can arise due to $\neg\forall i(\cdots)$ which is allowed in APF.)
4. Construct index set $\mathcal{I}$ containing
   - a fresh variable λ (representing all other positions in an array),
   - terms t such that a read $a[t]$ occurs in the formula and t is not a univ quantified var.
   - terms t (vars?) that occur in comparison with univ quantified var in index guards.
5. Replace universal quantification by finite conjunctions over $\mathcal{I}$.
6. Resulting formula F_6 is in $\mathrm{QF}(\Sigma_A)$. Decide satisfiability using algo for $\mathrm{QF}(\Sigma_A)$.

Example

Example 11.6 from BM

Example of APF Procedure

$$F : a\langle l \triangleleft v\rangle[k] = b[k] \wedge b[k] \neq v \wedge a[k] = v \wedge \forall i (i \neq l \rightarrow a[i] = b[i]).$$

Array Property Formulas with Arithmetic [Bradley, Manna, Sipma VMCAI 2006]

$T_{\mathsf{A}}^{\mathbb{Z}}$. An **array property** is again a $\Sigma_{\mathsf{A}}^{\mathbb{Z}}$-formula of the form

$$\forall \bar{i}.\ F[\bar{i}] \ \rightarrow\ G[\bar{i}]\ ,$$

where $\bar{i}$ is a list of integer variables, and $F[\bar{i}]$ and $G[\bar{i}]$ are the **index guard** and the **value constraint**, respectively. The form of an index guard is constrained according to the following grammar:

$$\begin{array}{rcl}
\mathsf{iguard} & \rightarrow & \mathsf{iguard} \wedge \mathsf{iguard} \mid \mathsf{iguard} \vee \mathsf{iguard} \mid \mathsf{atom} \\
\mathsf{atom} & \rightarrow & \mathsf{expr} \leq \mathsf{expr} \mid \mathsf{expr} = \mathsf{expr} \\
\mathsf{expr} & \rightarrow & uvar \mid \mathsf{pexpr} \\
\mathsf{pexpr} & \rightarrow & \mathsf{pexpr}' \\
\mathsf{pexpr}' & \rightarrow & \mathbb{Z} \mid \mathbb{Z} \cdot evar \mid \mathsf{pexpr}' + \mathsf{pexpr}'
\end{array}$$

where $uvar$ is any universally quantified integer variable, and $evar$ is any existentially quantified or free integer variable.

Array Property Fragment [Bradley, Manna, Sipma VMCAI 2006]

Consider the fragment of FO logic of the combined signatures $\Sigma_A = (\cdot[\cdot], \cdot\langle\cdot \triangleleft \cdot\rangle)$ and $\Sigma_{LA} = (+, -, <, 0, 1)$ consisting of: Boolean combinations of quantifier-free formulas over $\Sigma_A \cup \Sigma_{LA}$ and Array Property formulas.

Example APF formula

$$\begin{array}{l} l \leq k \leq u + 1 \, \wedge \\ a' = a\langle k \triangleleft 0\rangle \, \wedge \\ a'[k] \neq b'[k] \, \wedge \\ a'[u+1] = b[u+1] \, \wedge \\ \forall i((l \leq i \leq u) \implies a[i] = b[i]) \end{array}$$

Properties we can say in APF

- $\forall i(a[i] = b[i])$ (array equality)
- $\forall i((l \leq i \leq u) \implies a[i] = b[i])$ (bounded array equality)
- $\forall i((l \leq i \leq u) \implies 0 \leq a[i])$ (bounded universal property)
- $\forall i \forall j(i \leq j \implies a[i] \leq a[j])$ (increasing)

What we cannot say:

- $\forall i \forall j(i \neq j \implies a[i] \neq a[i])$ (distinct elements)
- $\forall i \forall j(i < j \implies a[i] < a[j])$ (strictly increasing)
- $\forall i(b[a[i]] = c[i])$

Reduction Algorithm

Algorithm 7.3.1: ARRAY-REDUCTION

Input: An array property formula ϕ_A in NNF
Output: A formula ϕ_{UF} in the index and element theories with uninterpreted functions

1. Apply the write rule to remove all array updates from ϕ_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 ϕ_{UF};
5. **return** ϕ_{UF};

Reduces $\mathrm{APF}(\Sigma_A^{\mathbb{Z}})$ to EUF+LA, which can be shown to be decidable using

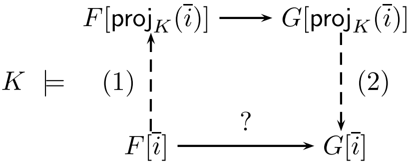
Example Reduction

Example

$$\forall i((l \leq i \leq u) \implies a[i] = b[i]) \land$$
$$\neg(\forall i((l \leq i \leq u + 1) \implies (a\langle(u+1) \triangleleft b[u+1]\rangle[i] = b[i])$$

Proof of Correctness [BM]

Let $K = J[\vec{i} \mapsto \vec{v}]$.



Overview

$\mathrm{QF}(\Sigma_A)$	Decidable	Reduce to EUF
$\mathrm{FO}(\Sigma_A)$	?	
$\mathrm{QF}(\Sigma_A^{\mathbb{Z}})$	Decidable	Nelson-Oppen on $\mathrm{QF}(\Sigma_A)$+LIA
$\mathrm{FO}(\Sigma_A^{\mathbb{Z}})$	Undecidable	Reduction from linear loop progs.
$\mathrm{APF}(\Sigma_A)$	Decidable	Reduce to $\mathrm{QF}(\Sigma_A)$
$\mathrm{APF}(\Sigma_A^{\mathbb{Z}})$	Decidable	Reduce to $\mathrm{QF}(\Sigma_A^{\mathbb{Z}})$