Array Fold Logic

Przemyslaw Daca et al.

September 30, 2020

Overview

- Contributions of this paper.
- Array fold logic: syntax, semantics and utilities.
- Theoretical results.
- Tool and experimental results.

Contributions

- Define a new logic called AFL that can express interesting properties of counting over arrays.
- Show the satisfiability of AFL is **PSPACE**-complete and provide a decision procedure.
- Implement a tool AFOLDER that can solve some cases of program verification.

A Toy Example

Why “fold”? A concept in functional language which folds some functions over an array.

Example

```
min = max = a[0];
j = k = 0;
for(i=0;i<size(a);i++) {
  if(a[i]<min) { min=a[i]; j=1; }
  if(a[i]==min) j++;
}
for(i=0;i<size(a);i++) {
  if(a[i]>max) { max=a[i]; k=1; }
  if(a[i]==max) k++;
}
assert(j==k);
```

A Toy Example

$$
\begin{aligned}
&\exists\, min, max, i_1, i_2, j, k\ .\\
&0 \leq i_1 < |a| \ \wedge\ 0 \leq i_2 < |a| \wedge\\
&a[i_1] = min \ \wedge\ a[i_2] = max \wedge\\
&\forall i.(a[i] \geq min) \wedge\\
&\forall i.(a[i] \leq max) \wedge\\
&j = \big|\{i \mid a[i] = min\}\big| \ \wedge\\
&k = \big|\{i \mid a[i] = max\}\big| \ \wedge\\
&j = k
\end{aligned}
$$

This kind of formula is undecidable because we can reduce the Hilbert's 10th problem to it.

$$
\begin{gathered}
0 \leq i_1 < |a| \ \wedge\ 0 \leq i_2 < |a| \ \wedge\ a[i_1] = min \ \wedge\ a[i_2] = max \wedge\\
fold_a \binom{0}{0} \binom{\mathbf{e}{=}min \Rightarrow \mathbf{c}_1\mathbf{{+}{+}}}{\mathbf{e}{>}min \Rightarrow skip} = \binom{|a|}{j} \ \wedge\ fold_a \binom{0}{0} \binom{\mathbf{e}{=}max \Rightarrow \mathbf{c}_1\mathbf{{+}{+}}}{\mathbf{e}{<}max \Rightarrow skip} = \binom{|a|}{k} \ \wedge\ j = k
\end{gathered}
$$

Array Fold Logic: Syntax

Implicit Variables: $\{\mathbf{e}, \mathbf{c}_1, \ldots, \mathbf{c}_{m-1}, \mathbf{i}\} = FV^m$

- Array sort, $\mathtt{ASort}$
- Integer sort, $\mathtt{ISort}$
- Boolean sort, $\mathtt{BSort}$
- Integer vectors $\mathtt{VSort}^m$
- Functional constants $\mathtt{FSort}^m = \mathtt{VSort}^m \times \mathtt{ISort} \rightarrow \mathtt{VSort}^m$

Array Fold Logic: Syntax

$$
\begin{array}{lcl}
A & ::= & a \mid a\{T \leftarrow T\} \\
T & ::= & n \mid x \mid T + T \mid a[T] \mid |a| \\
B & ::= & a = b \mid T = T \mid T < T \mid \neg B \mid B \wedge B \mid V^m = V^m \\
V^m & ::= & \begin{pmatrix} T \\ \cdots \\ T \end{pmatrix} \mid \mathit{fold}_a\, V^m\, F^m \\
F^m & ::= & \begin{pmatrix} \mathit{grd} \Rightarrow \mathit{upd} \\ \cdots \\ \mathit{grd} \Rightarrow \mathit{upd} \end{pmatrix} \\
\mathit{grd} & ::= & \mathbf{e} \approx T \mid \mathbf{i} \approx T \mid \mathbf{c}_m \approx T \mid \mathbf{s} \approx n \mid \mathit{grd} \wedge \mathit{grd} \qquad\qquad (\approx\, \in \{>, <, =, \neq\}) \\
\mathit{upd} & ::= & \mathbf{c}_m \mathrel{+}= n \mid \mathbf{s} \leftarrow n \mid \mathit{skip} \mid \mathit{break} \mid \mathit{upd}\,;\mathit{upd}
\end{array}
$$

Given a set of function branches Br, we can define a control flow graph $G = \langle S, E, \gamma \rangle$.

- $E = \bigcup_{grd \Rightarrow upd \in Br} \{(s_1, s_2) \mid s_1 \models grd \wedge s_2 = ite(\mathbf{s} \leftarrow n \in upd, n, s_1)\}$
- γ is the labeling of edges with the set of formulas $\Phi(grd)$ and $\Phi(upd)$.

Requirement: edges in the same SCC of G update the counters in a monotonic way.

Array Fold Logic: Semantics

$\sigma = \langle \lambda, \mu \rangle$ where $\mu : Var_I \to \mathbb{Z}, \lambda : Var_A \to \mathbb{Z}^*$.
$\kappa = FV^m \to \mathbb{Z}^{m+1}$

$$
\begin{array}{lll}
1. & \left[\begin{pmatrix} t_1^1 \\ \cdots \\ t_m^1 \end{pmatrix} = \begin{pmatrix} t_1^2 \\ \cdots \\ t_m^2 \end{pmatrix} \right]^\sigma & \equiv \left([t_1^1]^\sigma = [t_1^2]^\sigma \right) \wedge \ldots \wedge \left([t_m^1]^\sigma = [t_m^2]^\sigma \right) \\
2. & [fold_a \; v \; f]^\sigma & \equiv [fold_a \; v \; f]^{\sigma,\kappa} \text{, where } \kappa(FV^m) = \binom{v}{0} \\
3. & [fold_a \; v \; f]^{\sigma,\kappa} & \equiv \text{if } ([\mathbf{i}]^\kappa < 0) \text{ or } ([\mathbf{i}]^\kappa \geq |a|) \text{ or } (false \in [f]^{\sigma,\kappa}) \text{ then } v \\
 & & \quad \text{else } [fold_a \; v' \; f]^{\sigma,\kappa'} \text{, where } \kappa'(FV^m) = \binom{v'}{\mathbf{s}'} = [f]^{\sigma,\kappa} \left(\kappa(FV^m) \right) \\
4. & [f]^{\sigma,\kappa} \begin{pmatrix} v_1 \\ \cdots \\ v_m \end{pmatrix} & \equiv \begin{pmatrix} v_1' \\ \cdots \\ v_m' \end{pmatrix} \text{, where } v_j' \equiv \text{ if } upd(v_j) \in [f]^{\sigma,\kappa} \text{ then } upd(v_j) \text{, else } v_j \\
5. & \left[\begin{pmatrix} grd_1 \Rightarrow upd_1 \\ \cdots \\ grd_m \Rightarrow upd_m \end{pmatrix} \right]^{\sigma,\kappa} & \equiv \{\mathbf{i}' = \mathbf{i}+1\} \cup [grd_1 \Rightarrow upd_1]^{\sigma,\kappa} \cup \ldots \cup [grd_m \Rightarrow upd_m]^{\sigma,\kappa} \\
6. & [grd \Rightarrow upd]^{\sigma,\kappa} & \equiv \text{if } [grd]^{\sigma,\kappa} = true \text{ then } [upd]^{\sigma,\kappa} \text{ else } \emptyset \\
7. & [\mathbf{e} \approx t]^{\sigma,\kappa} & \equiv [\mathbf{e}]^\kappa \approx [t]^\sigma \qquad \text{(similarly for } \mathbf{i} \approx T,\ \mathbf{c}_m \approx T,\ \mathbf{s} \approx n) \\
8. & [grd_1 \wedge grd_2]^{\sigma,\kappa} & \equiv [grd_1]^{\sigma,\kappa} \wedge [grd_2]^{\sigma,\kappa} \\
9. & [upd_1; upd_2]^{\sigma,\kappa} & \equiv [upd_1]^{\sigma,\kappa} \cup [upd_2]^{\sigma,\kappa} \\
10. & [\mathbf{c}_m \mathrel{+}= n]^{\sigma,\kappa} & \equiv \{\mathbf{c}_m' = \mathbf{c}_m + n\} \\
11. & [\mathbf{s} \leftarrow n]^{\sigma,\kappa} & \equiv \{\mathbf{s}' = n\} \\
12. & [skip]^{\sigma,\kappa} & \equiv \emptyset \\
13. & [break]^{\sigma,\kappa} & \equiv \{false\}
\end{array}
$$

Utility and Expressive Power

- Boundedness:

$$fold_a(0)(l \leq \mathbf{e} \leq u \Rightarrow skip) = |a|$$

- Partitioning:

$$fold_a(0)\begin{pmatrix} \mathbf{i}<p \wedge \mathbf{e}\leq a[p] \Rightarrow skip \\ \mathbf{i}\geq p \wedge \mathbf{e}\geq a[p] \Rightarrow skip \end{pmatrix} = |a|$$

- Periodic:

$$fold_a(0)\begin{pmatrix} \mathbf{s}=0 \wedge \mathbf{e}=0 \Rightarrow \mathbf{s}\leftarrow 1 \\ \mathbf{s}=1 \wedge \mathbf{e}=1 \Rightarrow \mathbf{s}\leftarrow 0 \end{pmatrix} = |a|$$

- Pumping($0^n 1^n$):

$$fold_a\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\begin{pmatrix} \mathbf{s}=0 \wedge \mathbf{e}=0 \Rightarrow \mathbf{c}_1\texttt{++} \\ \mathbf{s}=0 \wedge \mathbf{e}=1 \Rightarrow \mathbf{c}_2\texttt{++} \wedge \mathbf{s}\leftarrow 1 \\ \mathbf{s}=1 \wedge \mathbf{e}=1 \Rightarrow \mathbf{c}_2\texttt{++} \end{pmatrix} = \begin{pmatrix} |a| \\ n \\ n \end{pmatrix}$$

- Counting.
- Histogram.

Properties that require universal quantification over *several* index variables are inexpressive. e.g. Sortedness and ...

Utility and Expressive Power

- Loops.
- Conditional statements.

```
static size_t parse_table_header(uint8_t *a, size_t size, ...)
  size_t i=0, pipes=0;
```

$$\left\{\ i_0 = 0 \ \wedge\ p_0 = 0\ \right\}$$

```
  while (i < size && a[i] != '\n')
    if (a[i++] == '|') pipes++;
```

$$\left\{\ \binom{i_1}{p_1} = \mathit{fold}_a \binom{i_0}{p_0} \left(\substack{\mathbf{e}=P \,\Rightarrow\, \mathbf{c}_1\texttt{++} \\ \mathbf{e}\neq P \,\wedge\, \mathbf{e}\neq N \,\Rightarrow\, \mathit{skip}} \right)\ \right\}$$

```
  if (a[0] == '|') pipes--;
```

$$\left\{\ \binom{*}{p_2} = \mathit{fold}_a \binom{0}{p_1} \left(\mathbf{i} = 0 \ \wedge\ \mathbf{e} = P \Rightarrow \mathbf{c}_1\texttt{--} \right)\ \right\}$$

```
 i++;
 if (i < size && a[i] == '|') i++;
```

$$\left\{\ i_2 = i_1 + 1 \ \wedge\ i_3 = \mathit{fold}_a(i_2)\left(\mathbf{i} = i_2 \ \wedge\ \mathbf{e} = P \ \Rightarrow\ \mathit{skip}\right)\ \right\}$$

```
  end = i;
  while (end < size && a[end] != '\n') end++;
```

$$\left\{\ e_0 = i_3 \ \wedge\ e1 = \mathit{fold}_a(e_0)\left(\mathbf{e} \neq N \ \Rightarrow\ \mathit{skip}\right)\ \right\}$$

Theoretical Results: Complexity

Definition (symbolic k-counter machine)

An SMC is a tuple $\mathcal{M} = (\eta, X, Q, \delta, q^{init})$ where

- η is a vector of k counters.
- $\delta \subseteq Q \times \mathtt{CC}_k(X) \times \mathtt{IC}(X) \times Q \times \mathbb{Z}^k$ is the transition relation.

Translation from a functional constant f of $\mathtt{FSort}^m$ to an SCM.
Reversal bounded.

$$SCC_1 \to SCC_2 \to \cdots \to SCC_m$$

Parallel composition of SCMs.

Small model property

Lemma

There exists a constant $c \in \mathbb{N}$, such that an AFL formula Φ is satisfiable iff there exists a model σ it maps each variable in X to integer that $\leq 2^{|\Phi|^c}$ and array to sequence of $\leq 2^{|\phi|^c}$ where each integer of the array also lies in the bound.

Give fixed counter values.
Why we want reversal-bounded?

Theorem

The satisfiability problem of AFL is **PSPACE***-complete.*

Membership: NTM.
Hardness: DFA emptiness problem reduced to sat of AFL formula.

Undecidable Extension

Theorem

Array fold logic with $\exists^*\forall^*$ extension is undecidable.

Proof.

Reduction from Hilbert's Tenth Problem to the decidability of quantified AFL. $x = y \cdot z$.

$$|a| = x \ \wedge\ fold_a \binom{0}{0} \begin{pmatrix} \mathbf{e}{=}0 \Rightarrow \mathit{skip} \\ \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1\texttt{++} \end{pmatrix} = \binom{|a|}{z} \ \wedge$$

$$\forall j \,.\, 0 \leq j < |a| \implies fold_a \binom{j}{0} \begin{pmatrix} \mathbf{i} \leq j{+}y \wedge \mathbf{e}{=}0 \Rightarrow \mathit{skip} \\ \mathbf{i} \leq j{+}y \wedge \mathbf{e}{=}1 \Rightarrow \mathbf{c}_1\texttt{++} \end{pmatrix} = \binom{*}{1}$$

□

Decision Procedure

Idea: translate the AFL formula ϕ into a quantifier-free PA formula $\psi = \psi_n \wedge \psi_e \wedge \psi_l$.

- ψ_n is part of ϕ that does not contain fold.
- ψ_e is the reduction from the reachability problem of SMC to QFPA.
- ψ_l is the link formula used for linking some constraints between initial and final configuration in ψ_e.

Lemma

The complexity of satifiability of m-AFL for a fixed m is **NP***-complete.*

Experimental Results

Tool: AFOLDER implemented on C++ and uses Z3 for PA solving.
Benchmarks:

- Program RedCarpet project that is used for parsing the Markdown language.
- NUMA(non-uniform memory access) which include multi-thread and memory operations.
- Several cases in SV-COMP.
- Histogram examples.

Experimental Results

Table 3: Experimental results for AFOLDER.

Example	$\|\phi\|$	folds	MFPA	transl. time	solving time	result	array length
Markdown(1)	62	6	3	$< 1s$	$< 1s$	sat	8
Markdown(2)	69	7	4	$1s$	$< 1s$	sat	14
Markdown(3)	76	8	5	$1.3s$	79s	sat	17
perf_bench_numa(10)	93	10	1	< 1s	< 1s	sat	100
perf_bench_numa(20)	183	20	1	< 1s	< 1s	sat	100
perf_bench_numa(40)	363	40	1	< 1s	< 1s	sat	100
standard_minInArray	10	3	3	< 1s	< 1s	unsat	-
linear_sea.ch_true	13	3	3	< 1s	< 1s	unsat	-
array_call3	11	2	3	< 1s	< 1s	unsat	-
standard_sentinel	14	3	3	< 1s	< 1s	unsat	-
standard_find	11	3	3	< 1s	< 1s	unsat	-
standard_vararg	11	3	3	< 1s	< 1s	unsat	-
histogram(8)	58	8	8	< 1s	1.3s	sat	9
histogram(9)	65	9	9	< 1s	6.9s	sat	10
histogram(10)	72	10	10	2s	55s	sat	11
histogram(11)	79	11	11	8s	368s	sat	12
histogram_unsat(11)	80	11	11	9s	19s	unsat	-