VERIFYING ARRAY-MANIPULATING PROGRAMS WITH MAX-STRATEGY ITERATION

Arijit Shaw
June 12, 2019

Master’s Thesis presentation, CMI
int[] A;
int i = 0;
while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}
assert(__CPROVER_forall
    {unsigned int j;
    !(j < A.Length) || A[j] = 0}):

Property to satisfy:
All elements are initialized.

\[ \forall k. 0 \leq k < A.length \implies a[k] = 0 \]
int[] A;
int i = 0;

while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}

assert(__CPROVER_forall {
    unsigned int j;
    !(j < A.Length) || A[j] = 0
});

Property to satisfy:
All elements are initialized.

\[ \forall k. 0 \leq k < A.length \implies a[k] = 0 \]

Loop invariant:
\[ \forall k. 0 \leq k < i \implies a[k] = 0 \]
Distinctions of Array Invariants

- Invariants are usually quantified over indices
• Invariants are usually quantified over indices

• Index set is partitioned into segments with all elements in a segment constrained in a particular way
- Invariants are usually quantified over indices

- Index set is partitioned into segments with all elements in a segment constrained in a particular way
  - 2 segments for the current example
Distinctions of Array Invariants

- Invariants are usually quantified over indices
- Index set is partitioned into segments with all elements in a segment constrained in a particular way
  - 2 segments for the current example
- Of course, there can be variations from the above pattern
Thesis Objectives

- Understanding how synthesis of Arrays invariants\(^1\) works in extensions to Abstract Interpretation.

- Extend standard Strategy Iteration algorithm for deriving scalar invariants by using some of those ideas
  - For a restricted class of array programs

- Develop an algorithm and a design architecture to implement it within 2LS.

Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Technical Issues for Extension to Arrays
   An Abstract Domain for Arrays

A Strategy Iteration Algorithm
Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Technical Issues for Extension to Arrays
  An Abstract Domain for Arrays

A Strategy Iteration Algorithm
Inductive invariants:

- holds initially
- if it holds, holds at next iteration
Abstract Domain and Templates

Interval Domain

\[ d_1 \leq x_1 \leq d_2 \]

Concrete Domain

Abstract Domain

\[ [d_1, d_2] \]
Abstract Domain and Templates

Interval Domain
\[ d_1 \leq x_1 \leq d_2 \]

Concrete Domain

Abstract Domain
\[ [ d_1 \quad d_2 ] \]

Templates
To capture more complicated structures.

\[ d_1 \leq x_1 - x_2 \leq d_2 \]

\[ x_1 + x_2 \leq d_3 \]
\[-d_2 \leq x_1 - x_2 \leq d_1\]
\[x_1 + x_2 \leq d_3\]

\[
\begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}
\]

\[T \cdot x \leq d\]
\[-d_2 \leq x_1 - x_2 \leq d_1\]
\[x_1 + x_2 \leq d_3\]

\[
\begin{pmatrix}
1 & -1 \\
-1 & 1 \\
1 & 1
\end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}
\]

\[T \cdot x \leq d\]

Interval Domain as Templates:
\[-d_2 \leq x_1 \leq d_1\]
\[
\begin{pmatrix} 1 \\ -1 \end{pmatrix} \cdot (x) \leq \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}
\]
Search for inductive invariants is second order logic problem:

\[ \exists_2 Inv. \forall x, x' (Init(x) \implies Inv(x)) \land (Inv(x) \land Trans(x, x')) \implies Inv(x') \]
· Search for inductive invariants is second order logic problem:

\[ \exists_2 \forall x, x' (Init(x) \implies Inv(x)) \land (Inv(x) \land \text{Trans}(x, x')) \implies Inv(x') \]

· Reduce the problem to a first order logic search using templates:

\[ \exists \delta. \forall x, x' (Init(x) \implies T(x, \delta)) \land (T(x, \delta) \land \text{Trans}(x, x')) \implies T(x', \delta) \]
Search for inductive invariants is a second order logic problem:

\[ \exists_2 Inv. \forall x, x' (Init(x) \implies Inv(x)) \land (Inv(x) \land Trans(x, x')) \implies Inv(x') \]

Reduce the problem to a first order logic search using templates:

\[ \exists \delta. \forall x, x' (Init(x) \implies T(x, \delta)) \land (T(x, \delta) \land Trans(x, x')) \implies T(x', \delta) \]

Remove existential quantifier by iteratively checking the formula using some solver:

\[ \forall x, x' (Init(x) \implies T(x, \delta)) \land (T(x, \delta) \land Trans(x, x')) \implies T(x', \delta) \]
Template Invariant as Fixed-Point Solution to Domain Equations

\[ \forall x, x' (\text{Init}(x) \implies T(x, \delta)) \land (T(x, \delta) \land \text{Trans}(x, x')) \implies T(x', \delta) \]

\[ \delta_{1,2} = \max \left\{ -\infty, \sup\{x'\mid x \leq \delta_{0,1} \land -x \leq -\delta_{0,2} \land x' = 5\}, \sup\{x'\mid x \leq \delta_{1,1} \land -x \leq -\delta_{1,2} \land x \leq 9 \land x' = x + 1\}, \sup\{x'\mid x \leq \delta_{2,1} \land -x \leq -\delta_{2,2} \land x \leq 0 \land x' = x\} \right\} \]
Template Invariant as Fixed-Point Solution to Domain Equations

\[ \forall x, x' \ (Init(x) \implies T(x, \delta)) \land (T(x, \delta) \land Trans(x, x')) \implies T(x', \delta) \]

\[ \delta_{1, 2} = \max \left\{ -\infty, \sup\{x' \mid x \leq \delta_{0, 1} \land -x \leq -\delta_{0, 2} \land x' = 5\}, \sup\{x' \mid x \leq \delta_{1, 1} \land -x \leq -\delta_{1, 2} \land x \leq 9 \land x' = x + 1\}, \sup\{x' \mid x \leq \delta_{2, 1} \land -x \leq -\delta_{2, 2} \land x \leq 0 \land x' = x\} \right\} \]

\[ \delta_{1, 1} = \max \left\{ -\infty, \sup\{-x' \mid x \leq \delta_{0, 1} \land -x \leq -\delta_{0, 2} \land x' = 5\}, \sup\{-x' \mid x \leq \delta_{1, 1} \land -x \leq -\delta_{1, 2} \land x \leq 9 \land x' = x + 1\}, \sup\{-x' \mid x \leq \delta_{2, 1} \land -x \leq -\delta_{2, 2} \land x \leq 0 \land x' = x\} \right\} \]
Strategies!

\[
\delta_{0,1} = \infty
\]

\[
\delta_{0,2} = \infty
\]

\[
\delta_{1,1} = \max \left\{ -\infty \right. \\
\sup \left\{ -x' \mid x \leq \delta_{0,1} \land -x \leq -\delta_{0,2} \land x' = 5 \right\}, \\
\sup \left\{ -x' \mid x \leq \delta_{1,1} \land -x \leq -\delta_{1,2} \land x' \leq 9 \land x' = x + 1 \right\}, \\
\sup \left\{ -x' \mid x \leq \delta_{2,1} \land -x \leq -\delta_{2,2} \land x' \leq 0 \land x' = x \right\}
\]

\[
\delta_{1,2} = \max \left\{ -\infty \right. \\
\sup \left\{ -x' \mid x \leq \delta_{0,1} \land -x \leq -\delta_{0,2} \land x' = 5 \right\}, \\
\sup \left\{ -x' \mid x \leq \delta_{1,1} \land -x \leq -\delta_{1,2} \land x' \leq 9 \land x' = x + 1 \right\}, \\
\sup \left\{ -x' \mid x \leq \delta_{2,1} \land -x \leq -\delta_{2,2} \land x' \leq 0 \land x' = x \right\}
\]

\[
\delta_{2,1} = \max \left\{ -\infty \right. \\
\sup \left\{ -x' \mid x \leq \delta_{1,1} \land -x \leq -\delta_{1,2} \land x \geq 10 \land x' = x \right\}, \\
\sup \left\{ -x' \mid x \leq \delta_{2,1} \land -x \leq -\delta_{2,2} \land x \geq 0 \land x' = x - 1 \right\}
\]

\[
\delta_{2,2} = \max \left\{ -\infty \right. \\
\sup \left\{ x' \mid x \leq \delta_{1,1} \land -x \leq -\delta_{1,2} \land x \geq 10 \land x' = x \right\}, \\
\sup \left\{ x' \mid x \leq \delta_{2,1} \land -x \leq -\delta_{2,2} \land x \geq 0 \land x' = x - 1 \right\}
\]
\[\delta_{0,1} = \infty\]
\[\delta_{0,2} = \infty\]

\[\delta_{1,1} = \max\left\{-\infty \mid x \leq \delta_{0,1} \land -x \leq \delta_{0,2} \land x' = 5, \right.\]
\[\left. \sup\{-x' \mid x \leq \delta_{1,1} \land -x \leq \delta_{1,2} \land x' \leq 9 \land x' = x + 1\}, \right.\]
\[\left. \sup\{-x' \mid x \leq \delta_{2,1} \land -x \leq \delta_{2,2} \land x' \leq 0 \land x' = x\}\right\}

\[\delta_{1,2} = \max\left\{-\infty \mid x \leq \delta_{0,1} \land -x \leq \delta_{0,2} \land x' = 5, \right.\]
\[\left. \sup\{-x' \mid x \leq \delta_{1,1} \land -x \leq \delta_{1,2} \land x' \leq 9 \land x' = x + 1\}, \right.\]
\[\left. \sup\{-x' \mid x \leq \delta_{2,1} \land -x \leq \delta_{2,2} \land x' \leq 0 \land x' = x\}\right\}

\[\delta_{2,1} = \max\left\{-\infty \mid x \leq \delta_{1,1} \land -x \leq \delta_{1,2} \land x \geq 10 \land x' = x\right\}, \sup\{-x' \mid x \leq \delta_{2,1} \land -x \leq \delta_{2,2} \land x \geq 0 \land x' = x - 1\}

\[\delta_{2,2} = \max\left\{-\infty \mid x \leq \delta_{1,1} \land -x \leq \delta_{1,2} \land x \geq 10 \land x' = x\right\}, \sup\{-x' \mid x \leq \delta_{2,1} \land -x \leq \delta_{2,2} \land x \geq 0 \land x' = x - 1\}\]
Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Technical Issues for Extension to Arrays
  An Abstract Domain for Arrays

A Strategy Iteration Algorithm
· Programs modeled as control flow graph (CFG).
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.

\[
\begin{align*}
x' &= 5 \\
x' &= x + 1 \\
x' &= x - 1 \\
x' &= x \\
\end{align*}
\]
· Programs modeled as control flow graph (CFG).
· Initialize Abstract values.
· Choose strategies one by one
· Until a fixedpoint is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.

\[
\begin{align*}
x' &= 5 \\
x' &= x + 1 \\
x' &= x - 1 \\
\end{align*}
\]
• Programs modeled as control flow graph (CFG).
• Initialize Abstract values.
• Choose strategies one by one
• Until a fixedpoint is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixed point is reached.
· Programs modeled as control flow graph (CFG).
· Initialize Abstract values.
· Choose strategies one by one
· Until a fixedpoint is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixed point is reached.
Programs modeled as control flow graph (CFG).

- Initialize Abstract values.
- Choose strategies one by one
- Until a fixed point is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixed point is reached.
· Programs modeled as control flow graph (CFG).
· Initialize Abstract values.
· Choose strategies one by one
· Until a fixedpoint is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.
Strategy Iteration: A Method to Derive Fixed Points

- Programs modeled as control flow graph (CFG).
- Initialize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.

\[
x' = 5 \\
x \leq 9 \land x' = x + 1 \\
x \geq 10 \land x' = x \\
x \leq 0 \land x' = x \\
x \geq 1 \land x' = x - 1
\]

Guarantee:

- Termination for finite systems
- Soundness: always returns a correct fixed-point;
- Optimality: Returns \( \text{lfp} \) if transition for polyhedral template if transition is monotonic.
Can we do this for Arrays too?
Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Technical Issues for Extension to Arrays
   An Abstract Domain for Arrays

A Strategy Iteration Algorithm
To find an optimal fixed point over this domain, we want to decide:

- Number of Segments
- Segment Limits
- Segment Abstractions

To find an optimal fixedpoint over this domain, we want to decide:

- Number of Segments
- Segment Limits
- Segment Abstractions


\[
\forall j. (0 \leq j < i \implies a \leq A[j] \leq b) \land (i \leq j < A.len \implies c \leq A[j] \leq d) \\
0 \leq i \land i \leq A.len
\]
To find a optimal fixedpoint over this domain, we want to decide:

- Number of Segments
- Segment Limits
- Segment Abstractions

Getting an Invariant with Array Domain

![Diagram of array segments]

**Given:**

- Number of Segments
- Segment Limits
**Given:**

- Number of Segments
- Segment Limits

Segment Abstractions: **Use an abstract domain.**

Use Max SI to get these bounds
Getting anInvariant with Array Domain

Segment Limits
Segment Abstraction

Given:

- Number of Segments: Use 2.
- Segment Limits: Linear expression over Loop Counter

Segment Abstractions: Use an abstract domain.

Use Max SI to get these bounds
\[
\forall A, A' (\text{Init}(A) \implies \text{Inv}(A)) \land (\text{Inv}(A) \land \text{Trans}(A, A')) \implies \text{Inv}(A')
\]

\[
\text{Inv}(A) = \forall j.(0 \leq j < i \implies a \leq A[j] \leq b) \land (i \leq j < A\text{.len} \implies c \leq A[j] \leq d)
\]
Outline

Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Technical Issues for Extension to Arrays
  An Abstract Domain for Arrays

A Strategy Iteration Algorithm
```c
int[] A;
int i = 0;
while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}
assert(__CPROVER_forall {
    unsigned int j;
    !(j < A.Length) || A
    [j] = 0
});
```
int[] A;
int i = 0;
while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}
assert(__CPROVER_forall {
    unsigned int j;
    !(j < A.Length) \lor A[j] = 0
});
In Array Segmentation Domain

\[
\begin{align*}
  i = 0 & \quad \text{[}0\} \perp \{i\} \perp \{A.len\} \\
  i < A.len \land A[i] = 0 & \land i = i + 1 \\
  i \geq A.length & \quad \text{[}0\} \perp \{i\} \perp \{A.len\} \\
  i = i + 1 & \quad \text{[}0\} \perp \{i\} \perp \{A.len\}
\end{align*}
\]
In Array Segmentation Domain

\[ l_0 \]

\[ \{0\} \perp \{i\} \perp \{A.len\} \]

\[ i = 0 \]

\[ \{0\} \perp \{i\} \perp \{A.len\} \]

\[ i : \top \]

\[ l_1 \]

\[ i < A.len \land A[i] = 0 \land i = i + 1 \]

\[ i \geq A.length \]

\[ l_2 \]

\[ \{0\} \perp \{i\} \perp \{A.len\} \]

\[ i : \bot \]
In Array Segmentation Domain

\[
\begin{align*}
  l_0 & \xrightarrow{i = 0} l_1, \\
  l_1 & \xrightarrow{i \geq A.length} l_2
\end{align*}
\]

- \(l_0\): \(\{0\} \uparrow \{A.len\}\) \\
  \(i : \top\)

- \(l_1\): \(\{0\} \uparrow [0, 0] \uparrow \{i\} \uparrow \{A.len\}\) \\
  \(i : [0, A.len]\)

- \(l_2\): \(\{0\} \perp \{i\} \perp \{A.len\}\) \\
  \(i : \bot\)

- \(i < A.len \land A[i] = 0 \land i = i + 1\)
In Array Segmentation Domain

\begin{align*}
\{0\} & \top \{A.\text{len}\} \\
i & : \top \\
\{0\} \leftarrow [0, 0] \leftarrow \{i\} \top \{A.\text{len}\} \\
i & : [0, A.\text{len}] \\
\{0\} \leftarrow [0, 0] \leftarrow \{i\} \top \{A.\text{len}\} \\
i & : [A.\text{len}, A.\text{len}] \\
i \geq A.\text{length} & \\
i < A.\text{len} \land A[i] = 0 \land i = i + 1
\end{align*}
Approach works well for problems with:

- Loop with a counter.
- Therefore initialization ...
- ...Copying
# define N 100000

int main( ) {
    int a1[N], a2[N], a, i, x;
    for ( i = 0 ; i < N ; i++ ) {
        a2[i] = a1[i];
    }
    for ( x = 0 ; x < N ; x++ ) {
        __VERIFIER_assert(a1[x] == a2[x]);
    }
    return 0;
}
What if we introduce more number of Segments

```java
int n = 10, i = 0;
int[] A = new int[n];
while (i < n - i) {
    A[i] = 0;
    A[n-i] = 1;
    i = i + 1;
}
```

Loop invariant:
\[
\forall i. ((i < n - i) \implies A[i] = 0 \land (i \geq n - i) \implies A[i] = 1)
\]

Domain needed for this:
\[
\{0\} \ [0, 0] \ {i} \ [n - i - 1] \ [1, 1] \ {n}\]
What if we introduce more powerful domain e.g., conditional with given predicates

```
1  int n = 10, i = 0, k = 5;
2  int[] A = new int[n];
3  while (i < n) {
4    if (i < k){
5      A[i] = 0;
6    }
7    else {
8      A[i] = -16;
9    }
10   i = i + 1;
11 }
```

Loop invariant:
\[ \forall j. ((j < i) \implies A[j] = 0 \land (j \geq n - i - 1) \implies A[j] = 1) \]

Domain needed for this:
\[
\begin{align*}
\{0\} & : j < k \implies [0, 0]  \quad \{\text{A.len}\} & : j < k \implies \bot \\
\{i\} & : j \geq k \implies [-16, -16] & \{\text{A.len}\} & : j \geq k \implies \bot
\end{align*}
\]
2LS

- C code
- SSA generator
- Template generator
- Strategy Iterator
- Invariants
- Property Checker
- Result: Yes | No | Unknown
Understanding current approach existing in Abstract Interpretation.

Extend existing scalar SI algorithm for arrays.

... Developing a design architecture to implement it within 2LS.
Future Work

- Generating Number of Array Segments.
- Generating Array Bound Parameters.
  - Maybe with Syntax Guided Synthesis.
Array Smashing

Array Exploding
What Others Do!

- Array Smashing
- Array Exploding
- Array Partitioning
Array Smashing  Array Exploding

Array Partitioning

- **Tiling**: Find a relation between LoopCounter and Indices.

- **Cell Morphing**: Abstract a of array type into a couple ($k, ak = a[k]$).

  Array programs → array-free Horn clauses → SMT-solver
• **Tile**: LoopCounter × Indices → \{tt,ff\} for loop \(L\).

• **Theorem**: If Tile satisfies some properties and if Pre → Inv holds then the Hoare triple \(\{Pre\}L\{Post\}\) holds for a tile.

• Put tiles to SMT solver to check whether these properties hold.

• Challenge: **Finding the right tile**.

---

**void foo(int A[], int N) {**
for (int i = 0; i < N; i++) {
if(!(i==0 || i==N-1)) {
   if (A[i] < 5) {
      A[i] = A[i-1];
   }
} else {
   A[i] = 5;
}
} else {  
   A[i] = 5;
}   
} assert(for k in 0..N-1, A[k]>=5);**

---

Source: Supratik Chakraborty, Ashutosh Gupta, and Divyesh Unadkat. *Verifying array manipulating programs by tiling.*
Cell morphing

- Array programs → array-free
  Horn clauses → SMT-solver
- Abstract a of array type into a couple \((k, ak = a[k])\)
- To each program point attach, instead of a set \(I\) of concrete states \((x_1, \ldots, x_m, a)\), a set \(I^\#\) of abstract states \((x_1, \ldots, x_m, k, ak)\).

Source: David Monniaux and Laure Gonnord. Cell morphing: from array programs to array-free horn clauses.