VERIFYING ARRAY-MANIPULATING PROGRAMS WITH MAX-STRATEGY ITERATION

Arijit Shaw

June 12, 2019

Master's Thesis presentation, CMI

Introduction

```
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.

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

Introduction

```
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

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

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

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.

[1] Cousot P, Cousot R, Logozzo F.: A parametric segmentation functor for fully automatic and scalable array content analysis. ACM SIGPLAN Notices. 2011

Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Techanical Issues for Extension to Arrays

An Abstract Domain for Arrays

A Strategy Iteration Algorithm

Outline

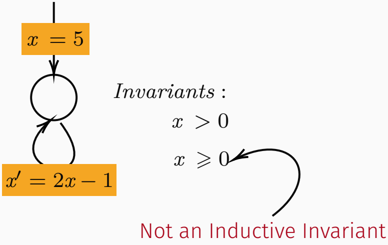
Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Techanical Issues for Extension to Arrays

- An Abstract Domain for Arrays

A Strategy Iteration Algorithm



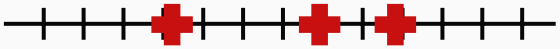
Inductive invariants :

- holds initially
- if it holds, holds at next iteration

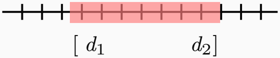
Interval Domain

$$d_1 \leq x_1 \leq d_2$$

Concrete Domain



Abstract Domain

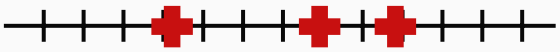


Abstract Domain and Templates

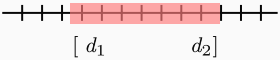
Interval Domain

$$d_1 \leq x_1 \leq d_2$$

Concrete Domain



Abstract Domain



Templates

To capture more complicated structures.

$$d_1 \leq x_1 - x_2 \leq d_2$$

$$x_1 + x_2 \leq d_3$$

Template Domain

$$-d_2 \leq x_1 - x_2 \leq d_1$$

$$x_1 + x_2 \leq d_3$$

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

$$\mathbf{T} \quad . \quad \mathbf{x} \quad \leq \quad \mathbf{d}$$

Template Domain

$$-d_2 \leq x_1 - x_2 \leq d_1$$

$$x_1 + x_2 \leq d_3$$

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

$$\mathbf{T} \quad . \quad \mathbf{x} \quad \leq \quad \mathbf{d}$$

Interval Domain as Templates:

$$-d_2 \leq x_1 \leq d_1$$

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} x \end{pmatrix} \leq \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

- Search for inductive invariants is second order logic problem :

$$\exists_2 Inv.\forall x, x' (Init(x) \implies Inv(x)) \wedge (Inv(x) \wedge Trans(x, x')) \implies Inv(x'))$$

TSIS: Reducing Second-order Search to First-Order Search

- Search for inductive invariants is second order logic problem :

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

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

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

TSIS: Reducing Second-order Search to First-Order Search

- Search for inductive invariants is second order logic problem :

$$\exists_2 Inv. \forall x, x' (Init(x) \implies Inv(x)) \wedge (Inv(x) \wedge Trans(x, x')) \implies Inv(x'))$$

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

$$\exists \delta. \forall x, x' (Init(x) \implies T(x, \delta)) \wedge (T(x, \delta) \wedge Trans(x, x')) \implies T(x', \delta))$$

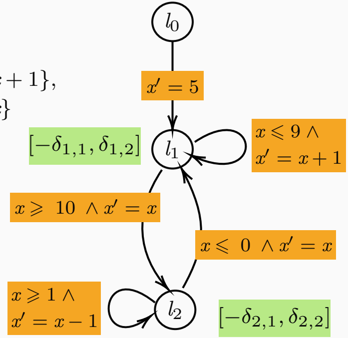
- Remove existential quatifier by iteratively checking the formula using some solver:

$$\forall x, x' (Init(x) \implies T(x, \delta)) \wedge (T(x, \delta) \wedge Trans(x, x')) \implies T(x', \delta))$$

Template Invariant as Fixed-Point Solution to Domain Equations

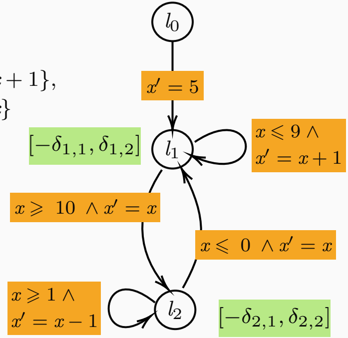
$$\forall x, x' (Init(x) \implies T(x, \delta)) \wedge (T(x, \delta) \wedge Trans(x, x')) \implies T(x', \delta))$$

$$\delta_{1,2} = max \begin{cases} -\infty \\ sup\{x' | x \leq \delta_{0,1} \wedge -x \leq -\delta_{0,2} \wedge x' = 5\}, \\ sup\{x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x \leq 9 \wedge x' = x + 1\}, \\ sup\{x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x \leq 0 \wedge x' = x\} \end{cases}$$



$$\forall x, x' (Init(x) \implies T(x, \delta)) \wedge (T(x, \delta) \wedge Trans(x, x')) \implies T(x', \delta))$$

$$\delta_{1,2} = max \begin{cases} -\infty \\ sup\{x' | x \leq \delta_{0,1} \wedge -x \leq -\delta_{0,2} \wedge x' = 5\}, \\ sup\{x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x \leq 9 \wedge x' = x + 1\}, \\ sup\{x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x \leq 0 \wedge x' = x\} \end{cases}$$



$$\delta_{1,1} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{0,1} \wedge -x \leq -\delta_{0,2} \wedge x' = 5\}, \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x \leq 9 \wedge x' = x + 1\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x \leq 0 \wedge x' = x\} \end{cases}$$

Strategies!

$$\delta_{0,1} = \infty$$

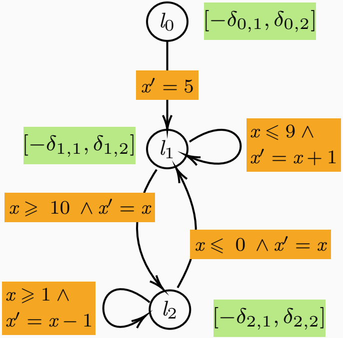
$$\delta_{0,2} = \infty$$

$$\delta_{1,1} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{0,1} \wedge -x \leq -\delta_{0,2} \wedge x' = 5\}, \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x' \leq 9 \wedge x' = x+1\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x' \leq 0 \wedge x' = x\} \end{cases}$$

$$\delta_{1,2} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{0,1} \wedge -x \leq -\delta_{0,2} \wedge x' = 5\}, \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x' \leq 9 \wedge x' = x+1\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x' \leq 0 \wedge x' = x\} \end{cases}$$

$$\delta_{2,1} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x \geq 10 \wedge x' = x\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x \geq 0 \wedge x' = x-1\} \end{cases}$$

$$\delta_{2,2} = max \begin{cases} -\infty \\ sup\{x' | x \leq \delta_{1,1} \wedge -x \leq -\delta_{1,2} \wedge x \geq 10 \wedge x' = x\}, \\ sup\{x' | x \leq \delta_{2,1} \wedge -x \leq -\delta_{2,2} \wedge x \geq 0 \wedge x' = x-1\} \end{cases}$$



$$\delta_{0,1} = \infty$$

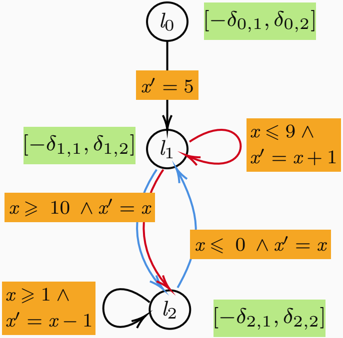
$$\delta_{0,2} = \infty$$

$$\delta_{1,1} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5\}, \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x' \leq 9 \wedge x' = x+1\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' \leq 0 \wedge x' = x\} \end{cases}$$

$$\delta_{1,2} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5\}, \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x' \leq 9 \wedge x' = x+1\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' \leq 0 \wedge x' = x\} \end{cases}$$

$$\delta_{2,1} = max \begin{cases} -\infty \\ sup\{-x' | x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x \geq 10 \wedge x' = x\}, \\ sup\{-x' | x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x \geq 0 \wedge x' = x-1\} \end{cases}$$

$$\delta_{2,2} = max \begin{cases} -\infty \\ sup\{x' | x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x \geq 10 \wedge x' = x\}, \\ sup\{x' | x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x \geq 0 \wedge x' = x-1\} \end{cases}$$



Outline

Template Shaped Invariant Synthesis

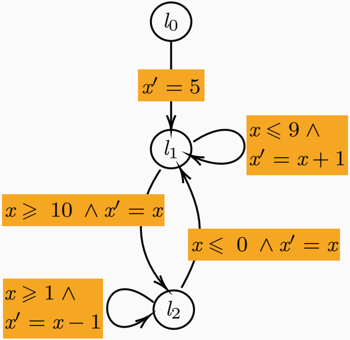
Strategy Iteration algorithm for Invariant Synthesis

Techanical Issues for Extension to Arrays

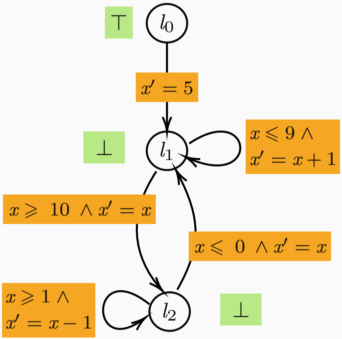
An Abstract Domain for Arrays

A Strategy Iteration Algorithm

- Programs modeled as control flow graph (CFG).

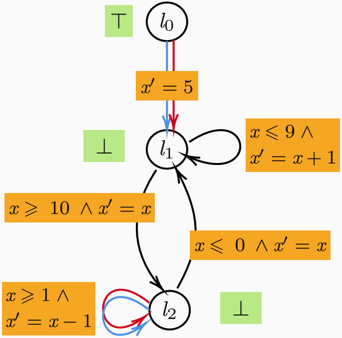


- Programs modeled as control flow graph (CFG).
- Inititalize Abstract values.

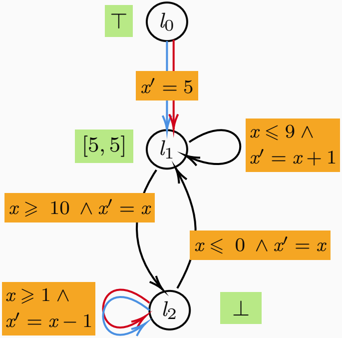


Strategy Iteration: A Method to Derive Fixed Points

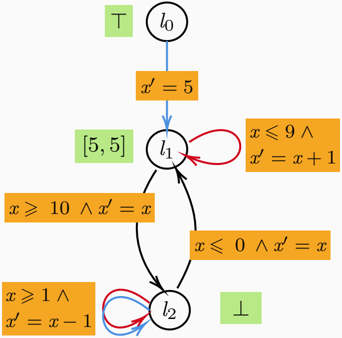
- Programs modeled as control flow graph (CFG).
- Inititalize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.



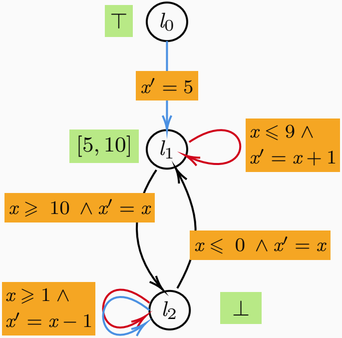
- Programs modeled as control flow graph (CFG).
- Inititalize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.



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

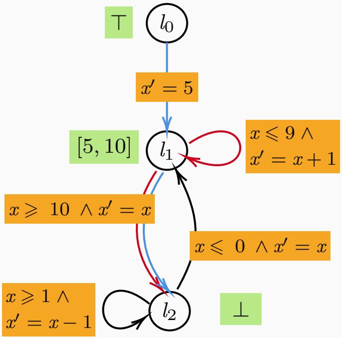


- Programs modeled as control flow graph (CFG).
- Inititalize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.



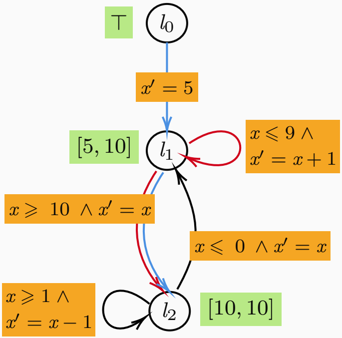
Strategy Iteration: A Method to Derive Fixed Points

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

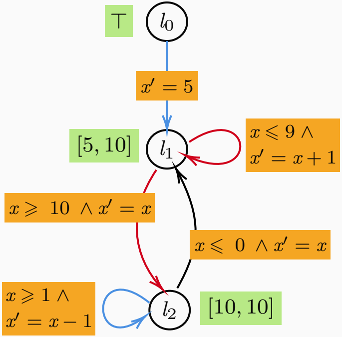


Strategy Iteration: A Method to Derive Fixed Points

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

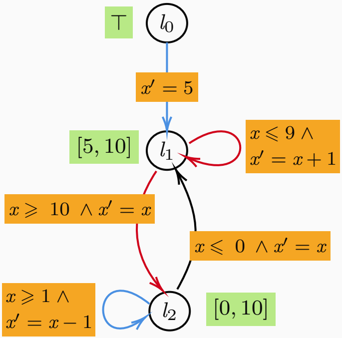


- Programs modeled as control flow graph (CFG).
- Inititalize Abstract values.
- Choose strategies one by one
- Until a fixedpoint is reached.



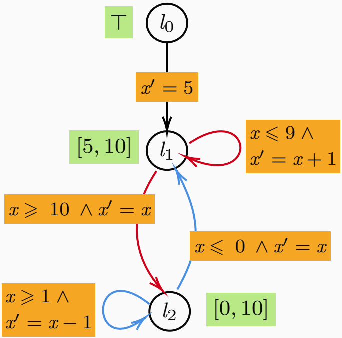
Strategy Iteration: A Method to Derive Fixed Points

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



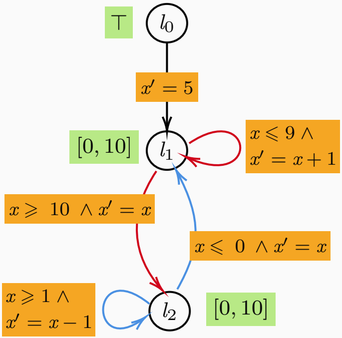
Strategy Iteration: A Method to Derive Fixed Points

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



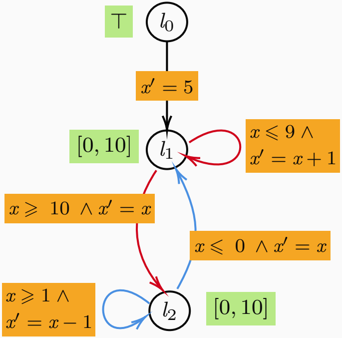
Strategy Iteration: A Method to Derive Fixed Points

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



Guarantee :

- Termination for finite systems
- Soundness : always returns a correct fixed-point;
- Optimality: Returns lfp if transition for polyhedral template if transition is monotonic.

Can we do this for Arrays too?

Outline

Template Shaped Invariant Synthesis

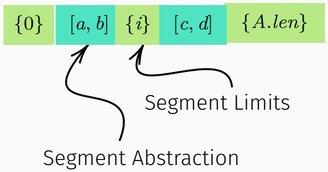
Strategy Iteration algorithm for Invariant Synthesis

Techanical Issues for Extension to Arrays

An Abstract Domain for Arrays

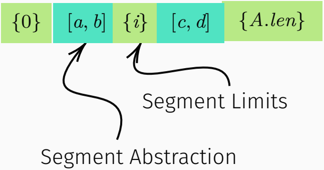
A Strategy Iteration Algorithm

An Array Domain



Cousot P, Cousot R, Logozzo F.: A parametric segmentation functor for fully automatic and scalable array content analysis. ACM SIGPLAN Notices. 2011

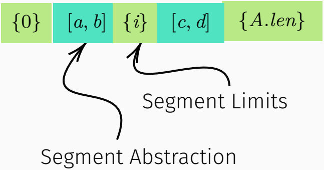
An Array Domain



$$\forall j.(0 \le j < i \implies a \le A[j] \le b) \land (i \le j < A.len \implies c \le A[j] \le d)$$

$$0 \le i \land i \le A.len$$

Cousot P, Cousot R, Logozzo F.: A parametric segmentation functor for fully automatic and scalable array content analysis. ACM SIGPLAN Notices. 2011



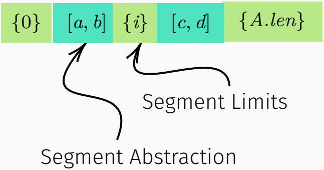
$$\forall j.(0 \leq j < i \implies a \leq A[j] \leq b) \wedge (i \leq j < A.len \implies c \leq A[j] \leq d)$$

$$0 \leq i \wedge i \leq A.len$$

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

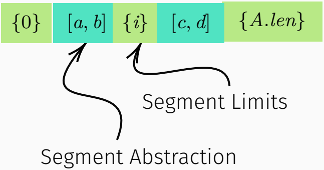
- Number of Segments
- Segment Limits
- Segment Abstractions

Cousot P, Cousot R, Logozzo F.: A parametric segmentation functor for fully automatic and scalable array content analysis. ACM SIGPLAN Notices. 2011



Given :

- Number of Segments
- Segment Limits

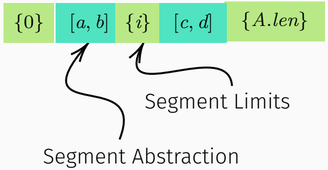


Given :

- Number of Segments
- Segment Limits

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

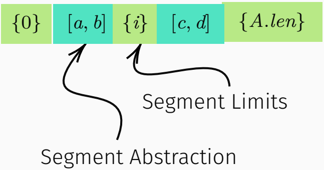
Use Max SI to get these bounds



Given :

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

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



$$\forall A, A' (Init(A) \implies Inv(A)) \land (Inv(A) \land Trans(A, A')) \implies Inv(A'))$$

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

Outline

Template Shaped Invariant Synthesis

Strategy Iteration algorithm for Invariant Synthesis

Techanical Issues for Extension to Arrays

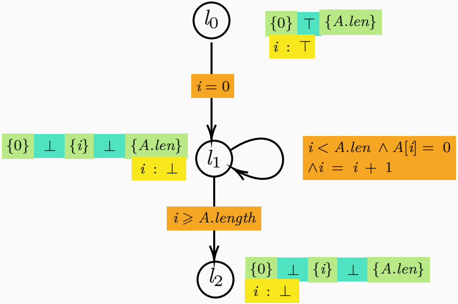
- An Abstract Domain for Arrays

A Strategy Iteration Algorithm

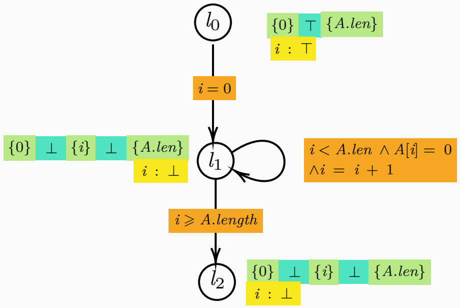
```
int[] A;
int i = 0;
while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}
assert(__CPROVER_forall
        {unsigned int j;
         !(j < A.Length) || A
              [j] = 0}
          );
```

Max-SI in arrays

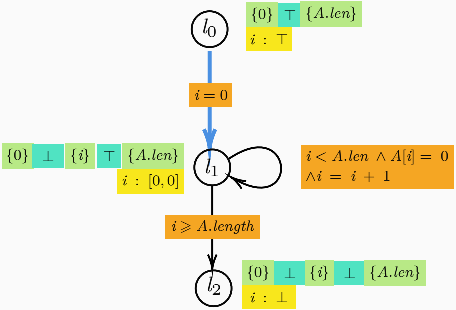
```
int[] A;
int i = 0;
while (i < A.Length) {
    A[i] = 0;
    i = i + 1;
}
assert(__CPROVER_forall
        {unsigned int j;
         !(j < A.Length) || A
              [j] = 0}
          );
```



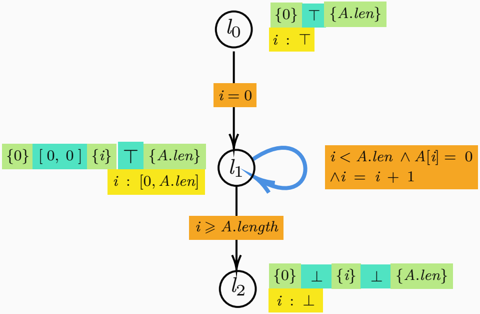
In Array Segmentation Domain

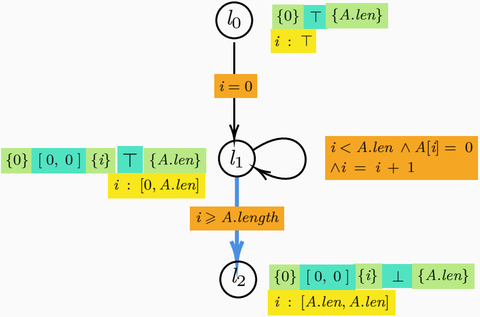


In Array Segmentation Domain



In Array Segmentation Domain




l_0
$\{0\}$ $\top$ $\{A.len\}$
$i : \top$
$i = 0$
l_1
$\{0\}$ $[0, 0]$ $\{i\}$ $\top$ $\{A.len\}$
$i : [0, A.len]$
$i < A.len \wedge A[i] = 0$
$\wedge i = i + 1$
$i \geqslant A.length$
l_2
$\{0\}$ $[0, 0]$ $\{i\}$ $\bot$ $\{A.len\}$
$i : [A.len, A.len]$

Intuition

Approach works well for problems with :

- Loop with a counter.
- Therefore initialization …
- …Copying

```c
#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;
}
```

Domain needed for this:

$a_1 \; - \; a_2 \; :$

$\{0\}$	$[0,0]$	$\{i\}$	$\top$	$\{A.len\}$

What if we introduce more number of Segments

```
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 :

$\forall i.((i < n - i) \implies A[i] = 0 \wedge$
$(i \geq n - i) \implies A[i] = 1)$

Domain needed for this:

$\{0\}$ $[0, 0]$ $\{i\}$ $\top$ $\{n - i - 1\}$ $[1, 1]$ $\{n\}$

What if we introduce more powerful domain.e.g., conditional with given predicates

```
int n = 10, i = 0, k = 5;
int[] A = new int[n];
while  (i < n) {
        if (i < k){
                A[i] = 0;
        }
        else {
                A[i] = -16;
        }
        i = i + 1:
}
```

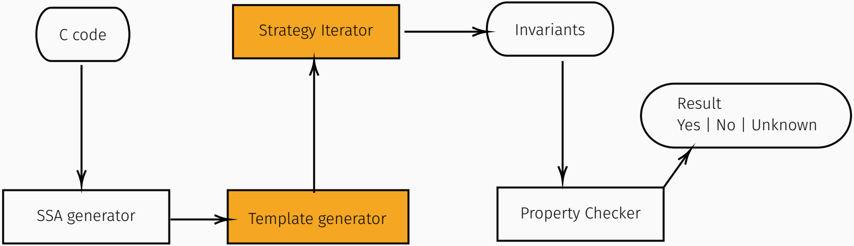
Loop invariant :

$\forall j.((j < i) \implies A[j] = 0 \wedge$
$(j \geq n - i - 1) \implies A[j] = 1)$

Domain needed for this:

$\{0\}$ $\begin{array}{l} j < k \implies [0,0] \\ j \geqslant k \implies [-16,-16] \end{array}$ $\{i\}$ $\begin{array}{l} j < k \implies \bot \\ j \geqslant k \implies \bot \end{array}$ $\{A.len\}$

2LS


C code
Strategy Iterator
Invariants
Result
Yes | No | Unknown
SSA generator
Template generator
Property Checker

Conclusion

- ✓ 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.

Extra Slides

What Others Do!

Array Smashing

Array Exploding

What Others Do!

Array Smashing

Array Exploding

Array Partitioning

What Others Do!

Array Smashing

Array Exploding

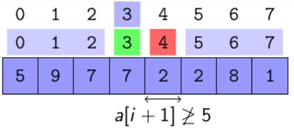
Array Partitioning

- **Tiling :** Find a relation between LoopCounter and Indices.
- **Cell Morphing :** Abstract a of array type into a couple $(k, ak = a[k])$.
  Array programs $\rightarrow$ array-free Horn clauses $\rightarrow$ SMT-solver

Tiling

- **Tile** : LoopCounter $\times$ Indices $\rightarrow$ {**tt,ff**} for loop L.
- **Theorem :** If Tile satisfies some properties and if Pre $\rightarrow$ 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+1] = A[i] + 1;
            A[i] = A[i-1];
        }
    } 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 $\rightarrow$ array-free Horn clauses $\rightarrow$ 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 $I^\sharp$ 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.**