Automaton-based Array Initialization Analysis

Đurica Nikolić and Fausto Spoto

Dipartimento di Informatica, University of Verona (Italy)

March 9th, 2012

Motivating Example - from the CubeWallpaper Android program by Google

```
private void readModel(String prefix) {
  String[] p = ....
  int numpoints = p.length;
  this.mOriginal = new ThreeDPoint[numpoints];
  this.mRotated = new ThreeDPoint[numpoints];
  for (int i = 0; i < numpoints; i++) {
    this.mOriginal[i] = new ThreeDPoint();
    this.mRotated[i] = new ThreeDPoint();
    String[] coord = p[i].split(" ");
    this.mOriginal[i].x=Float.valueOf(coord[0]);
    this.mOriginal[i].y=Float.valueOf(coord[1]);
    this.mOriginal[i].z=Float.valueOf(coord[2]);
  }
  [point *]
}
```

Motivating Example - from the CubeWallpaper Android program by Google

```
private void readModel(String prefix) {
  String[] p = ....
  int numpoints = p.length;
  this.mOriginal = new ThreeDPoint[numpoints];
  this.mRotated = new ThreeDPoint[numpoints];
  for (int i = 0; i < numpoints; i++) {
    this.mOriginal[i] = new ThreeDPoint();
    this.mRotated[i] = new ThreeDPoint();
    String[] coord = p[i].split(" ");
    this.mOriginal[i].x=Float.valueOf(coord[0]);
    this.mOriginal[i].y=Float.valueOf(coord[1]);
    this.mOriginal[i].z=Float.valueOf(coord[2]);
  }
  [point *]
}
```

MOTIVATING EXAMPLE - from the CubeWallpaper Android program by Google

```
private void readModel(String prefix) {
  String[] p = ....
  int numpoints = p.length;
  this.mOriginal = new ThreeDPoint[numpoints];
  this.mRotated = new ThreeDPoint[numpoints];
  for (int i = 0; i < numpoints; i++) {
    this.mOriginal[i] = new ThreeDPoint();
    this.mRotated[i] = new ThreeDPoint();
    String[] coord = p[i].split(" ");
    this.mOriginal[i].x=Float.valueOf(coord[0]);
    this.mOriginal[i].y=Float.valueOf(coord[1]);
    this.mOriginal[i].z=Float.valueOf(coord[2]);
  }
  [point *]
}
```

mOriginal and *mRotated* are COMPLETELY INITIALIZED at [point *]

Motivating Example - from the CubeWallpaper Android program by Google

```
private void readModel(String prefix) {
  String[] p = ....
  int numpoints = p.length;
  this.mOriginal = new ThreeDPoint[numpoints];
  this.mRotated = new ThreeDPoint[numpoints];
  for (int i = 0; i < numpoints; i++) {
    this.mOriginal[i] = new ThreeDPoint();
    this.mRotated[i] = new ThreeDPoint();
    String[] coord = p[i].split(" ");
    this.mOriginal[i].x=Float.valueOf(coord[0]);
    this.mOriginal[i].y=Float.valueOf(coord[1]);
    this.mOriginal[i].z=Float.valueOf(coord[2]);
  }
  [point *]
}
```

Our nullness analysis used to consider elements of *mOriginal* and *mRotated* POTENTIALLY NULL at [point *]

mOriginal and *mRotated* are COMPLETELY INITIALIZED at [point *]

Related Work

- Complete initialization of arrays to some value is **UNDECIDABLE**
- **STATIC ANALYSIS** can often help
- the majority of existing techniques **ARE NOT AUTOMATIC**
- the most precise existing approach is [CousotCousotLogozzo2011]
  - our approach is **SIMPLER** and **FASTER**, but **LESS PRECISE**
  - unlike theirs, our approach has been **FORMALLY PROVEN CORRECT**

GOAL: DEFINE, FORMALLY PROVE CORRECT AND IMPLEMENT AN ARRAY INITIALIZATION ANALYSIS FOR A JAVA-LIKE LANGUAGE

1. define SYNTAX and OPERATIONAL SEMANTICS of a Java-like language
2. define an ABSTRACT INTERPRETATION of the operational semantics
3. PROVE 1 and 2 related by a correctness relation
4. provide a STATIC ANALYSIS ALGORITHM
5. EXPERIMENTAL EVALUATION of the approach
6. extension of analysis to MULTI-DIMENSIONAL ARRAYS

Syntax of a simple imperative language

E	::=	n	INTEGER	ARITHMETIC EXPRESSIONS
	|	x	VARIABLE	
	|	$x.length$	ARRAY LENGTH	
	|	x[E]	ARRAY ELEMENT	
	|	E $\oplus$ E	$\oplus \in \{+, -, *, \div, \%\}$	
B	::=	true	TRUTH	BOOLEAN EXPRESSIONS
	|	false	FALSITY	
	|	¬B	NEGATION	
	|	$E \otimes E$	$\otimes \in \{<, \leq, =\}$	
	|	$B \oslash B$	$\oslash \in \{\wedge, \vee\}$	
A	::=	B	TEST	ACTIONS
	|	x := E	VARIABLE ASSIGNMENT	
	|	x[E] := E	ARRAY ELEMENT ASSIGNMENT	
	|	x := `new` t[E]	CREATION OF AN ARRAY	
C	::=	L_1 : A → L_2;	COMMAND	

Example

A Java loop

```
1. i = 0;
2. if (b<5) {
3.  a[i] = 2;
  }
4. while (i< a.length) {
5.  a[i] = 3;
6.  i++;
  }
7. ...
```

Corresponding transition system

Example

A Java loop

```
1. i = 0;
2. if (b<5) {
3.  a[i] = 2;
  }
4. while (i< a.length) {
5.  a[i] = 3;
6.  i++;
  }
7. ...
```

Corresponding transition system

$C_0 \quad 1: \quad i := 0 \rightarrow 2;$

Example

A Java loop

```
1. i = 0;
2. if (b<5) {
3.  a[i] = 2;
  }
4. while (i< a.length) {
5.  a[i] = 3;
6.  i++;
  }
7. ...
```

Corresponding transition system

$C_0 \quad 1 : \quad i := 0 \rightarrow 2;$

$C_1 \quad 2 : \quad b < 5 \rightarrow 3;$

$C_2 \quad 2 : \quad \neg(b < 5) \rightarrow 4;$

Example

A Java loop

```
1. i = 0;
2. if (b<5) {
3.  a[i] = 2;
  }
4. while (i< a.length) {
5.  a[i] = 3;
6.  i++;
  }
7. ...
```

Corresponding transition system

C_0 $1 :$ $i := 0 \rightarrow 2;$

C_1 $2 :$ $b < 5 \rightarrow 3;$

C_2 $2 :$ $\neg(b < 5) \rightarrow 4;$

C_3 $3 :$ $a[i] := 2 \rightarrow 4;$

C_4 $4 :$ $i < a.length \rightarrow 5;$

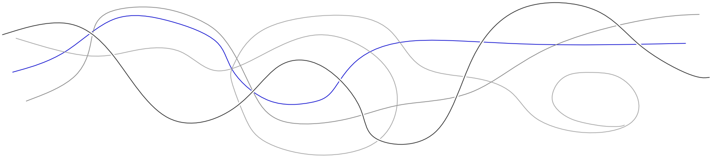
C_5 $4 :$ $\neg(i < a.length) \rightarrow 7;$

C_6 $5 :$ $a[i] := 3 \rightarrow 6;$

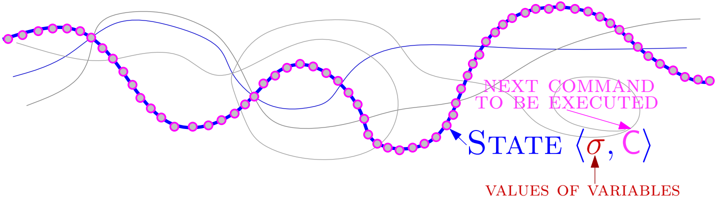
C_7 $6 :$ $i := i + 1 \rightarrow 4;$

C_8 $7 :$ $\cdots$

Trace-based Operational Semantics

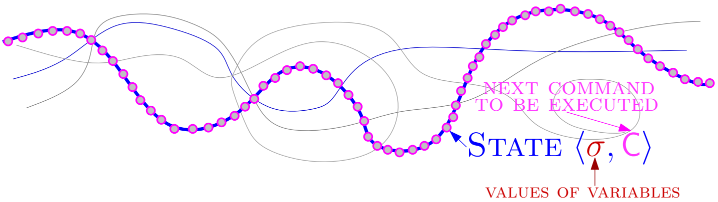


Trace-based Operational Semantics



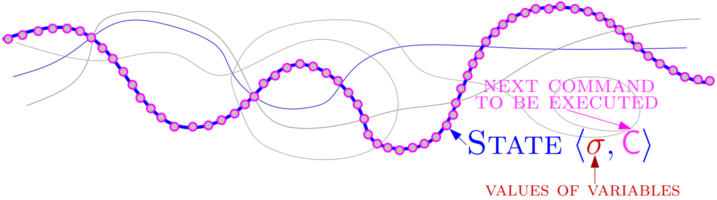
- States describe the current **configuration of the system**

Trace-based Operational Semantics



- States describe the current **CONFIGURATION OF THE SYSTEM**
- Operational semantics at C_n: **@C_n - ALL OF THE TRACES THAT REACH C_n**

TRACE-BASED OPERATIONAL SEMANTICS



- States describe the current **CONFIGURATION OF THE SYSTEM**
- Operational semantics at C_n: **@C_n - ALL OF THE TRACES THAT REACH C_n**
- Problem: **INFINITE TRACES**

ABSTRACT INTERPRETATION

- To reduce the size of the system, we use ABSTRACT INTERPRETATION
- General idea: ABSTRACT AWAY IRRELEVANT INFORMATION
- CONCRETE STATE: $\langle \sigma, C \rangle$ vs. ABSTRACT STATE: $\langle \sout{\sigma}, \alpha(C) \rangle$,
  i.e., *abstract states are obtained from concrete states by removing variables' values and some irrelevant commands*

ABSTRACT INTERPRETATION

- To reduce the size of the system, we use ABSTRACT INTERPRETATION
- General idea: ABSTRACT AWAY IRRELEVANT INFORMATION
- CONCRETE STATE: $\langle \sigma, \mathrm{C} \rangle$ vs. ABSTRACT STATE: $\langle \not\sigma, \alpha(\mathrm{C}) \rangle$,
  i.e., *abstract states are obtained from concrete states by removing variables' values and some irrelevant commands*
- How do we determine RELEVANT COMMANDS?

How do we determine relevant commands?

Array of interest: *a*

Index variable: *i*

$$
\begin{array}{l}
\ldots \\
\mathbf{i} = \mathbf{0}; \\
\ldots \\
\text{while } \mathbf{i} < \mathbf{a.length} \text{ do} \{ \\
\quad \ldots \\
\quad \mathbf{a}[\mathbf{i}] = \cdots; \\
\quad \ldots \\
\quad \mathbf{i}++; \\
\quad \ldots \\
\} \\
\ldots
\end{array}
$$

How do we determine relevant commands?

Array of interest: *a*
Index variable: *i*

$\ldots$
$\mathbf{i} = \mathbf{0};$
$\ldots$
while $\mathbf{i} < \mathbf{a.length}$ do$\{$
  $\ldots$
  $\mathbf{a}[\mathbf{i}] = \cdots;$
  $\ldots$
  $\mathbf{i}++;$
  $\ldots$
$\}$
$\ldots$

Abstraction: 0
Index variable *i* **initialized to** 0

How do we determine relevant commands?

Array of interest: *a*
Index variable: *i*

$$
\begin{array}{l}
\ldots \\
\mathbf{i} = \mathbf{0}; \\
\ldots \\
\text{while } \mathbf{i} < \mathbf{a.length} \text{ do}\{ \\
\quad \ldots \\
\quad \mathbf{a}[\mathbf{i}] = \cdots; \\
\quad \ldots \\
\quad \mathbf{i}++; \\
\quad \ldots \\
\} \\
\ldots
\end{array}
$$

Abstraction: $=$
Array element $a[i]$ **initialized**

How do we determine relevant commands?

Array of interest: *a*
Index variable: *i*

$$
\begin{array}{l}
\ldots \\
\mathbf{i} = \mathbf{0}; \\
\ldots \\
\text{while } \mathbf{i} < \mathbf{a.length} \text{ do} \{ \\
\quad \ldots \\
\quad \mathbf{a}[\mathbf{i}] = \cdots; \\
\quad \ldots \\
\quad \mathbf{i}++; \\
\quad \ldots \\
\} \\
\ldots
\end{array}
$$

Abstraction: $+$
Index variable *i* incremented by 1

How do we determine relevant commands?

Array of interest: *a*
Index variable: *i*

$$\ldots$$
$$\mathbf{i} = \mathbf{0};$$
$$\ldots$$
while $\mathbf{i} < \mathbf{a.length}$ do$\{$
$$\quad\ldots$$
$$\quad\mathbf{a}[\mathbf{i}] = \cdots;$$
$$\quad\ldots$$
$$\quad\mathbf{i}++;$$
$$\quad\ldots$$
$$\}$$
$$\ldots$$

Abstraction: $\geq$
Index variable compared with *a.length*

Abstract Interpretation

- To reduce the size of the system, we use ABSTRACT INTERPRETATION
- General idea: ABSTRACT AWAY IRRELEVANT INFORMATION
- CONCRETE STATE: $\langle \sigma, C \rangle$ vs. ABSTRACT STATE: $\langle \not\sigma, \alpha(C) \rangle$, i.e., *abstract states are obtained from concrete states by removing variables' values and some irrelevant commands*
- How do we determine RELEVANT COMMANDS?

$$\alpha(\langle L_1 : A \to L_2; \rangle) = \begin{cases} 0 & \text{if A is } i := 0 \\ + & \text{if A is } i := i + 1 \\ \geq & \text{if A is } \neg(i < a.length) \\ = & \text{if A is } a[i] = E \\ \mathcal{I} & \text{otherwise, if } mod(A) \cap \{a, i\} = \varnothing \\ \mathcal{R} & \text{otherwise,} \end{cases}$$

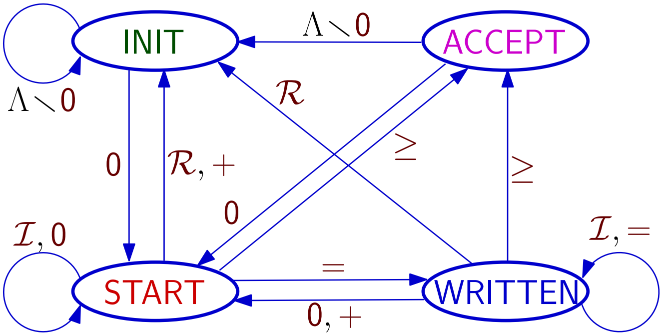
Abstraction of traces

We abstract traces by **regular expressions** on $\Lambda = \{0, =, +, \geq, \mathcal{I}, \mathcal{R}\}$

$$
\begin{aligned}
\text{INIT} &\cong \Lambda^* \\
\text{START} &\cong \Lambda^* 0 \mathcal{I}^* ((=\mathcal{I}^*)^+ + \mathcal{I}^*)^* \\
\text{WRITTEN} &\cong \Lambda^* 0 \mathcal{I}^* ((=\mathcal{I}^*)^+ + \mathcal{I}^*)^* (=\mathcal{I}^*)^+ \\
\text{ACCEPT} &\cong \Lambda^* 0 \mathcal{I}^* ((=\mathcal{I}^*)^+ + \mathcal{I}^*)^* (=\mathcal{I}^*)^* \geq
\end{aligned}
$$

Abstract Semantics: Finite-State Deterministic Automaton

- **ALPHABET** $\Lambda = \{0, +, =, \geq, \mathcal{I}, \mathcal{R}\}$
- **STATES** $S = \{\mathrm{INIT}, \mathrm{START}, \mathrm{WRITTEN}, \mathrm{ACCEPT}\}$;
- **TRANSITION FUNCTION** $\delta : S \times \Lambda \to S$: given states $\mathrm{p}, \mathrm{q} \in S$ and $\lambda \in \Lambda$, if the automaton has a transition from p to q labelled by λ, then $\delta(\mathrm{p}, \lambda) = \mathrm{q}$.



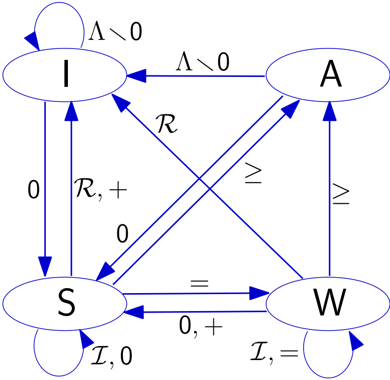
THE STATIC ANALYSIS ALGRITHM: ARRAYINIT

Algorithm for a specific pairs $\langle a, i \rangle$

for all $\mathsf{C} \in \mathbb{C}$ **do**
    $\varphi(\mathsf{C}) := \varnothing;$
end for
$\mathsf{ws} := [\langle \mathsf{C}_{init}, \mathsf{INIT} \rangle];$
$\varphi(\mathsf{C}_{init}) := \{\mathsf{INIT}\};$
while $(!\mathsf{ws.isEmpty}())$ **do**
    $\langle \mathsf{C}, \sigma^\sharp \rangle := \mathsf{ws.pop}();$
    for all C_1 such that $\mathsf{suc}(\mathsf{C}) = \mathsf{ini}(\mathsf{C}_1)$ **do**
        $\sigma_1^\sharp := \delta(\sigma^\sharp, s(\mathsf{C}));$
        if $(\sigma_1^\sharp \notin \varphi(\mathsf{C}_1))$ **then**
            $\mathsf{ws.push}(\langle \mathsf{C}_1, \sigma_1^\sharp \rangle);$
            $\varphi(\mathsf{C}_1) := \varphi(\mathsf{C}_1) \cup \{\sigma_1^\sharp\};$
        end if
    end for
end while

EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

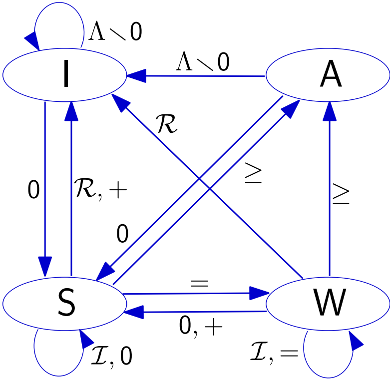
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



Example of application of the ArrayInit algorithm

Commands

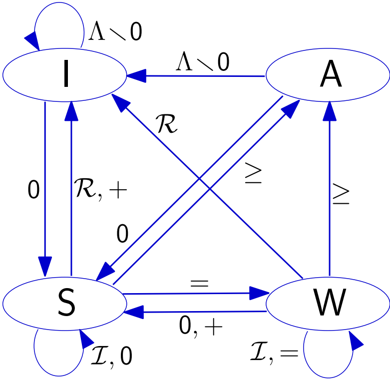
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



Example of application of the ArrayInit algorithm

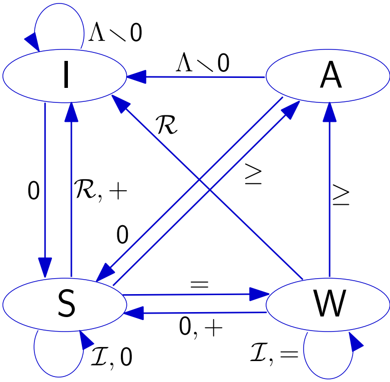
Reachable states

I	C_0	1	:	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2	:	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2	:	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3	:	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4	:	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4	:	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5	:	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6	:	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7	:	$\cdots$	



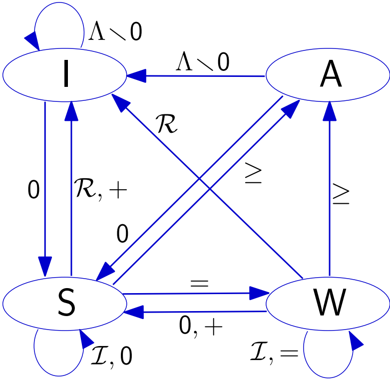
Example of application of the ArrayInit algorithm

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\emptyset$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\emptyset$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\emptyset$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\emptyset$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\emptyset$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\emptyset$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\emptyset$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

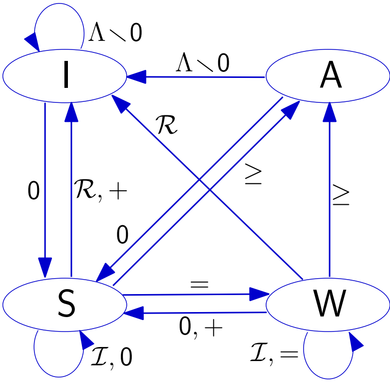
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

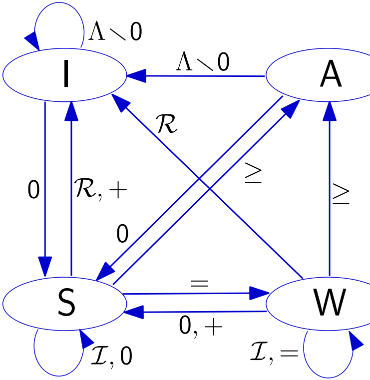
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	

ABSTRACTION OF COMMAND: 0



EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

I	C_0	1 :	$i := 0$	→ 2 ;
∅	C_1	2 :	$b < 5$	→ 3 ;
∅	C_2	2 :	$\neg(b < 5)$	→ 4 ;
∅	C_3	3 :	$a[i] := 2$	→ 4 ;
∅	C_4	4 :	$i < a.length$	→ 5 ;
∅	C_5	4 :	$\neg(i < a.length)$	→ 7 ;
∅	C_6	5 :	$a[i] := 3$	→ 6 ;
∅	C_7	6 :	$i := i + 1$	→ 4 ;
	C_8	7 :	$\cdots$	

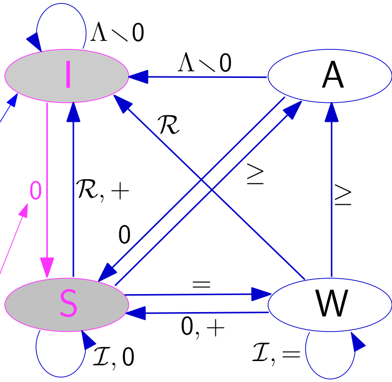


ABSTRACTION OF COMMAND: 0

CURRENT STATE: I

Example of application of the ArrayInit algorithm

I	C_0	1	:	$i := 0$	$\rightarrow$ 2 ;
∅	C_1	2	:	$b < 5$	$\rightarrow$ 3 ;
∅	C_2	2	:	$\neg(b < 5)$	$\rightarrow$ 4 ;
∅	C_3	3	:	$a[i] := 2$	$\rightarrow$ 4 ;
∅	C_4	4	:	$i < a.length$	$\rightarrow$ 5 ;
∅	C_5	4	:	$\neg(i < a.length)$	$\rightarrow$ 7 ;
∅	C_6	5	:	$a[i] := 3$	$\rightarrow$ 6 ;
∅	C_7	6	:	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7	:	$\cdots$	

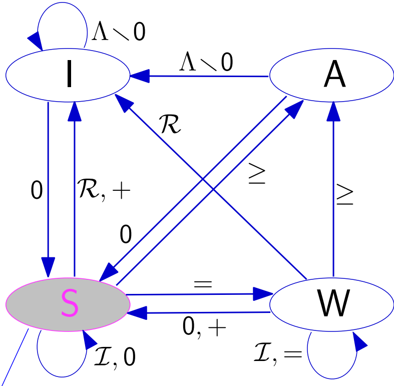


Abstraction of command: 0

Current state: I

Example of application of the ArrayInit algorithm

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



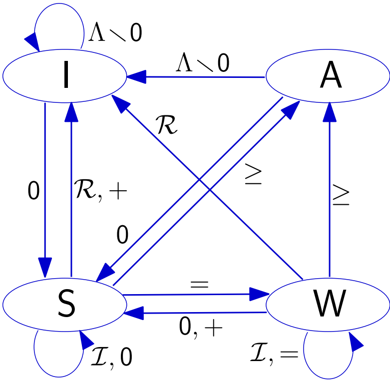
Abstraction of command: 0

Current state: I   Next state: S

Example of application of the ArrayInit algorithm

Next command(s)

I	C_0	1	:	$i := 0$	$\rightarrow$ 2 ;
$\varnothing$	C_1	2	:	$b < 5$	$\rightarrow$ 3 ;
$\varnothing$	C_2	2	:	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3	:	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4	:	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4	:	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5	:	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6	:	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7	:	$\cdots$	

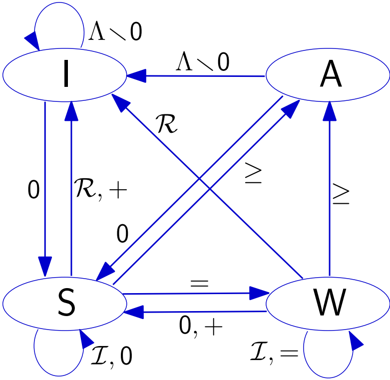


Abstraction of command: 0

Current state: I   Next state: S

EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

I	C_0	1	:	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2	:	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2	:	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3	:	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4	:	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4	:	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5	:	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6	:	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7	:	$\cdots$	

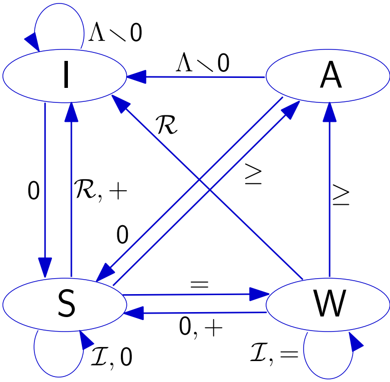


ABSTRACTION OF COMMAND: 0

CURRENT STATE: I   NEXT STATE: S

Example of application of the ArrayInit algorithm

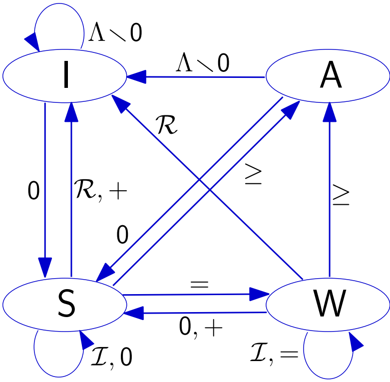
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



Working set: $\langle C_2, S \rangle$, $\langle C_1, S \rangle$

Example of application of the ArrayInit algorithm

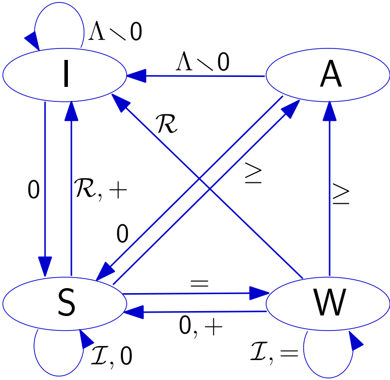
I	C_0	1 :	$i := 0$	$\rightarrow$	2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$	3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$	4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$	4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$	5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$	7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$	6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$	4 ;
	C_8	7 :	$\cdots$		



Working set: $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

Example of application of the ArrayInit algorithm

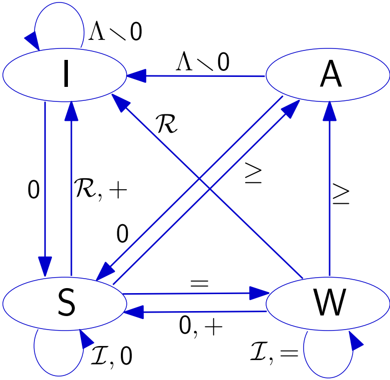
I	C_0	1	:	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2	:	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2	:	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3	:	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4	:	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4	:	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5	:	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6	:	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7	:	$\cdots$	



Working set: $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	

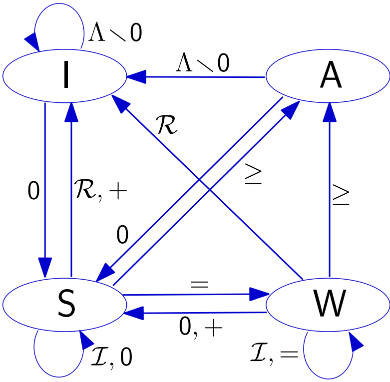


ABSTRACTION OF COMMAND: $\mathcal{I}$

WORKING SET: $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

Example of application of the ArrayInit algorithm

I	C_0	1 :	$i := 0$	$\rightarrow$	2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$	3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$	4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$	4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$	5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$	7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$	6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$	4 ;
	C_8	7 :	$\cdots$		



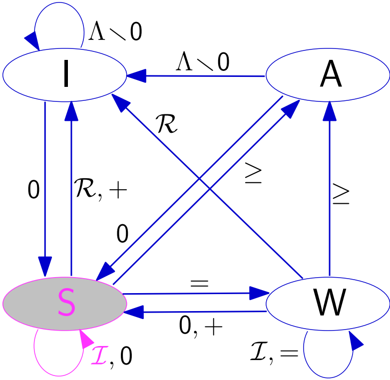
Abstraction of command: $\mathcal{I}$

Current state: S

Working set: $\langle C_2, S \rangle$, $\langle C_1, S \rangle$

EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



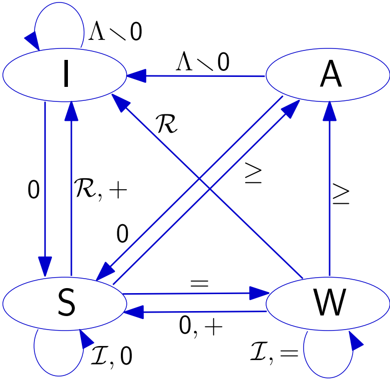
ABSTRACTION OF COMMAND: $\mathcal{I}$

CURRENT STATE: S NEXT STATE: S

WORKING SET: $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

Example of application of the ArrayInit algorithm

I	C_0	1 :	$i := 0$	$\rightarrow$	2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$	3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$	4 ;
$\varnothing$	C_3	3 :	$a[i] := 2$	$\rightarrow$	4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$	5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$	7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$	6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$	4 ;
	C_8	7 :	$\cdots$		



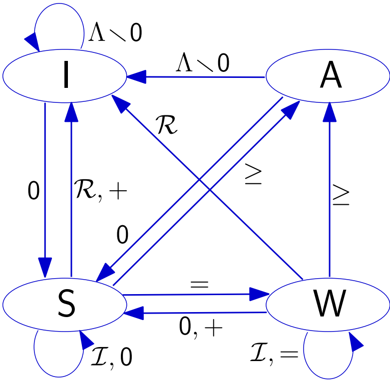
Abstraction of command: $\mathcal{I}$

Current state: S Next state: S

Working set: $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

Example of application of the ArrayInit algorithm

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
S	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\varnothing$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\varnothing$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\varnothing$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\varnothing$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



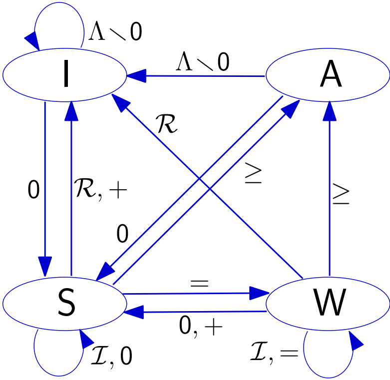
Abstraction of command: $\mathcal{I}$

Current state: S Next state: S

Working set: $\langle C_2, S \rangle$, $\langle C_1, S \rangle$

Example of application of the ArrayInit algorithm

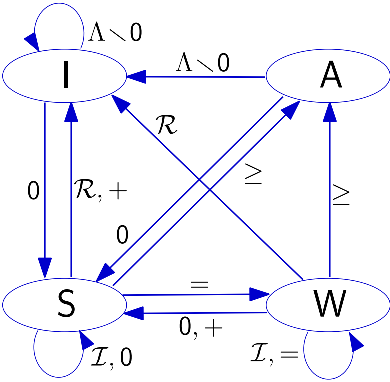
I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
S	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
$\emptyset$	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
$\emptyset$	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
$\emptyset$	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
$\emptyset$	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
	C_8	7 :	$\cdots$	



Working set: $\langle C_3, S\rangle$, $\langle C_2, S\rangle$, $\langle C_1, S\rangle$

Example of application of the ArrayInit algorithm

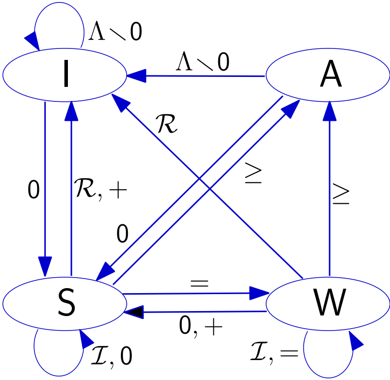
I	C_0	1 :	$i := 0$	→	2 ;
S	C_1	2 :	$b < 5$	→	3 ;
S	C_2	2 :	$\neg(b < 5)$	→	4 ;
S	C_3	3 :	$a[i] := 2$	→	4 ;
∅	C_4	4 :	$i < a.length$	→	5 ;
∅	C_5	4 :	$\neg(i < a.length)$	→	7 ;
∅	C_6	5 :	$a[i] := 3$	→	6 ;
∅	C_7	6 :	$i := i + 1$	→	4 ;
	C_8	7 :	$\cdots$		



Working set: $\langle C_3, S \rangle$, $\langle C_2, S \rangle$

EXAMPLE OF APPLICATION OF THE ARRAYINIT ALGORITHM

I	C_0	1 :	$i := 0$	$\rightarrow$ 2 ;
S	C_1	2 :	$b < 5$	$\rightarrow$ 3 ;
S	C_2	2 :	$\neg(b < 5)$	$\rightarrow$ 4 ;
S	C_3	3 :	$a[i] := 2$	$\rightarrow$ 4 ;
S, W	C_4	4 :	$i < a.length$	$\rightarrow$ 5 ;
S, W	C_5	4 :	$\neg(i < a.length)$	$\rightarrow$ 7 ;
S, W	C_6	5 :	$a[i] := 3$	$\rightarrow$ 6 ;
W	C_7	6 :	$i := i + 1$	$\rightarrow$ 4 ;
A	C_8	7 :	$\cdots$	



WORKING SET: $\varnothing$ - FIXPOINT REACHED

Soundness of the ArrayInit algorithm

Lemma

At the end of ArrayInit, $@C \subseteq \cup\varphi(C)$ holds for each command C.

Theorem

Consider a program *P*, variables *a* (ARRAY) and *i* (INDEX) and the automaton for *a* and *i*. At the end of the ArrayInit algorithm, for each command C such that $\varphi(C) = \{ACCEPT\}$, C IS A POINT OF *P* WHERE ALL ELEMENTS OF *a* HAVE BEEN INITIALIZED BY A LOOP WITH INDEX *i*.

JULIA - A STATIC ANALYZER FOR JAVA AND ANDROID

- Julia analyzer FINDS BUGS IN JAVA AND ANDROID PROGRAMS well BEFORE THEY ARE RUN. It is a SEMANTICAL TOOL, based on abstract interpretation, which checks all possible executions of a software and finds all possible bugs, inside the categories considered by the tool.
- Julia is very simple to use: it requires one or more JAR FILES making up the software, and it performs the following analyses:

- NULLNESS: is there a pointer dereferenced before being initialized?

JULIA - A STATIC ANALYZER FOR JAVA AND ANDROID

- Julia analyzer **FINDS BUGS IN JAVA AND ANDROID PROGRAMS** well **BEFORE THEY ARE RUN**. It is a **SEMANTICAL TOOL**, based on abstract interpretation, which checks all possible executions of a software and finds all possible bugs, inside the categories considered by the tool.
- Julia is very simple to use: it requires one or more **JAR FILES** making up the software, and it performs the following analyses:

- **NULLNESS**: is there a pointer dereferenced before being initialized?

ARRAYINIT has been implemented inside Julia.

Experimental Evaluation

Name	LOC	Total LOC	ArrayInit Total	ArrayInit Detected	ArrayInit Time	Total Time
AbdTest	489	56334	1	1	2.36	121.73
AccelerometerPlay	306	46854	1	1	0.35	71.99
CubeWallpaper	370	25654	3	3	0.12	28.51
HoneycombGallery	948	71501	1	0	1.06	157.85
TicTacToe	607	59040	3	3	0.70	102.65
Snake	420	57075	1	0	0.36	117.49
Real3D	1228	74384	2	2	1.06	177.95
ChimeTimer	4095	95781	9	7	0.80	383.45
Dazzle	4376	100271	4	1	1.02	394.44
OnWatch	9746	113368	10	6	2.91	525.15
Tricorder	10410	106100	17	11	1.01	467.58
TestAppv2	377	58365	1	1	0.38	102.34
TxWthr	2024	74441	7	1	0.42	179.78
JFlex	7681	40872	7	6	1.35	72.46
nti	2372	13098	4	4	0.09	13.55
plume	8587	43302	24	21	1.19	113.07

Experimental Evaluation

NAME	LOC	TOTAL LOC	ARRAYINIT TOTAL	ARRAYINIT DETECTED	ARRAYINIT TIME	TOTAL TIME
AbdTest	489	56334	1	1	2.36	121.73
AccelerometerPlay	306	46854	1	1	0.35	71.99
CubeWallpaper	370	25654	3	3	0.12	28.51
HoneycombGallery	948	71501	1	0	1.06	157.85
TicTacToe	607	59040	3	3	0.70	102.65
Snake	420	57075	1	0	0.36	117.49
Real3D	1228	74384	2	2	1.06	177.95
ChimeTimer	4095	95781	9	7	0.80	383.45
Dazzle	4376	100271	4	1	1.02	394.44
OnWatch	9746	113368	10	6	2.91	525.15
Tricorder	10410	106100	17	11	1.01	467.58
TestAppv2	377	58365	1	1	0.38	102.34
TxWthr	2024	74441	7	1	0.42	179.78
JFlex	7681	40872	7	6	1.35	72.46
nti	2372	13098	4	4	0.09	13.55
plume	8587	43302	24	21	1.19	113.07

Experimental Evaluation

NAME	LOC	TOTAL LOC	ARRAYINIT TOTAL	ARRAYINIT DETECTED	ARRAYINIT TIME	TOTAL TIME
AbdTest	489	56334	1	1	2.36	121.73
AccelerometerPlay	306	46854	1	1	0.35	71.99
CubeWallpaper	370	25654	3	3	0.12	28.51
HoneycombGallery	948	71501	1	0	1.06	157.85
TicTacToe	607	59040	3	3	0.70	102.65
Snake	420	57075	1	0	0.36	117.49
Real3D	1228	74384	2	2	1.06	177.95
ChimeTimer	4095	95781	9	7	0.80	383.45
Dazzle	4376	100271	4	1	1.02	394.44
OnWatch	9746	113368	10	6	2.91	525.15
Tricorder	10410	106100	17	11	1.01	467.58
TestAppv2	377	58365	1	1	0.38	102.34
TxWthr	2024	74441	7	1	0.42	179.78
JFlex	7681	40872	7	6	1.35	72.46
nti	2372	13098	4	4	0.09	13.55
plume	8587	43302	24	21	1.19	113.07

Experimental Evaluation

Name	LOC	Total LOC	ArrayInit Total	ArrayInit Detected	ArrayInit Time	Total Time
AbdTest	489	56334	1	1	2.36	121.73
AccelerometerPlay	306	46854	1	1	0.35	71.99
CubeWallpaper	370	25654	3	3	0.12	28.51
HoneycombGallery	948	71501	1	0	1.06	157.85
TicTacToe	607	59040	3	3	0.70	102.65
Snake	420	57075	1	0	0.36	117.49
Real3D	1228	74384	2	2	1.06	177.95
ChimeTimer	4095	95781	9	7	0.80	383.45
Dazzle	4376	100271	4	1	1.02	394.44
OnWatch	9746	113368	10	6	2.91	525.15
Tricorder	10410	106100	17	11	1.01	467.58
TestAppv2	377	58365	1	1	0.38	102.34
TxWthr	2024	74441	7	1	0.42	179.78
JFlex	7681	40872	7	6	1.35	72.46
nti	2372	13098	4	4	0.09	13.55
plume	8587	43302	24	21	1.19	113.07

ArrayInit increases the total time by only 0.47%.

Experimental Evaluation

NAME	Nullness	
	ArrayInit	~~ArrayInit~~
AccelerometerPlay	3	6
ChimeTimer	33	36
CubeWallpaper	0	3
TicTacToe	0	2
JFlex	57	65
OnWatch	82	85
Real3D	19	19
Tricorder	107	121
TxWthr	48	49
nti	15	15
plume	57	59

Experimental Evaluation

NAME	Nullness	
	ArrayInit	~~ArrayInit~~
AccelerometerPlay	3	6
ChimeTimer	33	36
CubeWallpaper	0	3
TicTacToe	0	2
JFlex	57	65
OnWatch	82	85
Real3D	19	19
Tricorder	107	121
TxWthr	48	49
nti	15	15
plume	57	59

ArrayInit improves the precision of the Nullness Analysis by 8.48% on the average.

Goal: define, formally prove correct and implement an array initialization analysis for a Java-like language

1. define SYNTAX and OPERATIONAL SEMANTICS of a Java-like language
2. define an ABSTRACT INTERPRETATION of the operational semantics
3. PROVE 1 and 2 related by a correctness relation
4. provide a STATIC ANALYSIS ALGORITHM
5. EXPERIMENTAL EVALUATION of the approach
6. extension of analysis to MULTI-DIMENSIONAL ARRAYS

GOAL: DEFINE, FORMALLY PROVE CORRECT AND IMPLEMENT AN ARRAY INITIALIZATION ANALYSIS FOR A JAVA-LIKE LANGUAGE

1. define SYNTAX and OPERATIONAL SEMANTICS of a Java-like language DONE
2. define an ABSTRACT INTERPRETATION of the operational semantics DONE
3. PROVE 1 and 2 related by a correctness relation DONE
4. provide a STATIC ANALYSIS ALGORITHM DONE
5. EXPERIMENTAL EVALUATION of the approach DONE
6. extension of analysis to MULTI-DIMENSIONAL ARRAYS (submitted for publication) DONE

Thank You