A Parametric Segmentation Functor for Fully Automatic and Scalable Array Content Analysis

Honza Horáček

Masaryk University

Brno, Czech Republic

6th November 2017

Introduction

- `FunArray`: a static analyzer for arrays.
  - Fast, must scale well.
  - For very large programs (.NET libraries).
- Objectives
  1. Reduce false alarms in imprecise array content analysis.
  2. Allow automatically proving user provided **non relational** pre/post conditions and assertions.
- Approach
  - Abstract interpretation.

A Motivating Example

```
1  public Random(int Seed) {
2   Contract.Requires(Seed != Int32.MinValue);
3   int num2 = 161803398 - Math.Abs(Seed);
4   this.SeedArray = new int[56];
5   this.SeedArray[55] = num2;
6   int num3 = 1;
7
8   // Loop 1
9   for (int i = 1; i < 55; i++) {
10   int index = (21 * i) % 55;
11   this.SeedArray[index] = num3; // (*)
12   num3 = num2 - num3;
13   if (num3 < 0) num3 += 2147483647;
14    num2 = this.SeedArray[index];
15  }
16
17  Contract.Assert(Contract.Forall( // (**)
18  0,this.SeedArray.Length - 1, i => a[i] >= -1));
19  .
```

FunArray's approach

1. Divide the array into possibly empty segments delimited by a set of segment bounds.
2. Abstract content of each segment uniformly.
   - Allow combining with user-provided abstractions.

Contribution

1. The array segmentation is automatically and semantically inferred during the analysis.
2. The combinatorial explosion in the handing of disjunctions is avoided by considering symbolic segment bounds as well as possibly empty segments.
3. The relations between array indexes and array elements can be inferred.
4. The precision/cost ratio can be tuned using a functor abstract domain.

Automatic Segmentation: Example

```
        void Init(int[] A) {
/* 0: */  int i = 0;
/* 1: */  while /* 2: */ (i < A.Length) {
/* 3: */   A[i] = 0;
/* 4: */   i = i + 1;
/* 5: */  }
/* 6: */ }
```

Automatic Segmentation: Notation

```
A = [5, 5, 8, 8, ?]:
[ A: {0} 5 {2} 8 {4} T {A.length} ]
```

Automatic Segmentation: The Analysis

```
1 p0 = [ A: {0} T {A.Length}? ]
```

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`
4. `p4 = p3[A[i]=0] = [ A: {0 i} 0 {1 i+1} T {A.Length}? ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`
4. `p4 = p3[A[i]=0] = [ A: {0 i} 0 {1 i+1} T {A.Length}? ]`
5. `p5 = p4[i=i+1] = [ A: {0 i-1} 0 {1 i} T {A.Length}? ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`
4. `p4 = p3[A[i]=0] = [ A: {0 i} 0 {1 i+1} T {A.Length}? ]`
5. `p5 = p4[i=i+1] = [ A: {0 i-1} 0 {1 i} T {A.Length}? ]`
6. `p2 = p1 U p5 = [ A: {0} 0 {i}? T {A.Length}? ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`
4. `p4 = p3[A[i]=0] = [ A: {0 i} 0 {1 i+1} T {A.Length}? ]`
5. `p5 = p4[i=i+1] = [ A: {0 i-1} 0 {1 i} T {A.Length}? ]`
6. `p2 = p1 U p5 = [ A: {0} 0 {i}? T {A.Length}? ]`
7. `p2 = p1 U p5 = [ A: {0} 0 {i}? T {A.Length}? ]`

Automatic Segmentation: The Analysis

1. `p0 = [ A: {0} T {A.Length}? ]`
2. `p2 = p1 = p0[i=0] = [ A: {0 i} T {A.Length}? ]`
3. `p3 = p2[i<n] = [ A: {0 i} T {A.Length} ]`
4. `p4 = p3[A[i]=0] = [ A: {0 i} 0 {1 i+1} T {A.Length}? ]`
5. `p5 = p4[i=i+1] = [ A: {0 i-1} 0 {1 i} T {A.Length}? ]`
6. `p2 = p1 U p5 = [ A: {0} 0 {i}? T {A.Length}? ]`
7. `p2 = p1 U p5 = [ A: {0} 0 {i}? T {A.Length}? ]`
8. `p6 = p2[i>=A.Length] = [ A: {0} 0 {A.Length,i}? ]`

Abstraction

- Main idea: different abstractions for different things!
  1. Variable and expression abstract domain. (**R**, **E**)
  2. Segment bounds abstract domain. (**B(E)**)
  3. Array element abstract domain. (**A**)
- FunArray abstract domain functor: **S(B(E), A, R)**.

Example of FunArray Abstract Domain Instantiation

```
int n = 10, i = 0;
int[] A = new int[n];

/* 1: */ while /* 2: */ (i < n) {
/* 3: */   A[i] = 0;
/* 4: */   i = i + 1:
/* 5: */   A[i] = -16;
/* 6: */   i = i + 1:
/* 7: */ }
/* 8: */
```

First attempt

- Reduced product of parity and intervals where pairs of a parity and an interval denote the conjunction of both properties.
- Used both for variables and array elements.

First attempt

- Reduced product of parity and intervals where pairs of a parity and an interval denote the conjunction of both properties.
- Used both for variables and array elements.

```
p1 = [ A: <{0 i} (T, [-oo,+oo]) {n 10}> ]
     [ i: (e, [0,0]) n: (e, [10,10]) ]
p2 = [ A: <{0} (e, [-16,0]) {i}? (T, [-oo,+oo]) {n 10}?> ]
     [ i: (e, [0,10]) n: (e, [10,10]) ]
p8 = [ A: <{0} (e, [-16,0]) {n 10 i}> ]
     [ i: (e, [10,10]) n: (e, [10,10]) ]
```

Second attempt

- Reduced cardinal power of intervals by parity.

```
p1 = [ A: <{0 i} (o -> [-oo,+oo],e -> [-oo,+oo]) {n 10}> ]
     [ i: (e, [0,0]) n: (e, [10,10]) ]
p2 = [ A: <{0} (o -> [-16,-16],e -> [0,0]) {i}?
           (o -> [-oo,+oo],e -> [-oo,+oo]) {n 10}?> ]
     [ i: (e, [0,10]) n: (e, [10,10]) ]
p8 = [ A: <{0} (o -> [-16,-16],e -> [0,0]) {n 10 i}> ]
     [ i: (e, [10,10]) n: (e, [10,10]) ]
```

Second attempt

- Reduced cardinal power of intervals by parity.

```
p1 = [ A: <{0 i} (o -> [-oo,+oo],e -> [-oo,+oo]) {n 10}> ]
     [ i: (e, [0,0]) n: (e, [10,10]) ]
p2 = [ A: <{0} (o -> [-16,-16],e -> [0,0]) {i}?
           (o -> [-oo,+oo],e -> [-oo,+oo]) {n 10}?> ]
     [ i: (e, [0,10]) n: (e, [10,10]) ]
p8 = [ A: <{0} (o -> [-16,-16],e -> [0,0]) {n 10 i}> ]
     [ i: (e, [10,10]) n: (e, [10,10]) ]
```

- Must be relational for arrays!

Performance

Lib	# func.	time	arr.	Δ	# inv.
`mscorlib.dll`	23 475	4:06	4:15	0:09	2 430
`System.dll`	15 489	3:40	3:46	0:06	1 385
`System.data.dll`	12 408	4:49	4:55	0:06	1 325
`System.Drawings.dll`	3 123	0:28	0:29	0:01	289
`System.Web.dll`	23 647	4:56	5:02	0:06	840
`System.Xml.dll`	10 510	3:59	4:16	0:17	807

Conclusion

- It has proved to be simple enough to scale up in production-quality static analysis tools.
- They used it to validate its own implementation, effectively reducing false alarms to zero.
- `FunArray` is a part of `Clousot` and available in Visual Studio "as a single checkbox".