An Assume Guarantee Approach for Checking Quantified Array Assertions

Mohamed Nassim Seghir
Freiburg University
Germany

Array verification

- Access is safe (index in range)

  ASTREE, BLAST, SLAM, MAGIC, etc.

- Array content respects the specification (Functional properties)
  - Individual elements (quantifier free assertions)

    BLAST, SLAM, MAGIC, etc.
  - Collection of elements:
    - quantifiers are required

      E.g. “all array elements are initialized”, “Array is sorted”
    - Limitation for existing tools

Problem

- Predicate abstraction: a set of program states is represented by a logical formula φ built up from a finite set of quantifier-free base formulas

  E.g., $x > 0 \wedge y = 0$ represents the set of states where program variable x has strictly positive values and y is null

- Limitation: a quantified formula $\forall x \in [0,n[.\ a[x] = 0$ is represented via

  $a[0] = 0, a[1] = 0, \ldots..$

Motivation

- An automatic method to check quantified array assertions [SAS' 09]
- limitation: can not handle insertion sort
- Observation: can deal with the inner loop of insertion sort
- Task: how to extend the method with modular reasoning

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

backward unrolling

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n-1)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }

     index = a[n-1];
     j = n-1;
     while ((j > 0) && (a[j-1] > index))
     {
        a[j] = a[j-1];
        j = j - 1;
      }
     a[j] = index;
      i++;
```

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

L(1,n)

backward unrolling

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n-1)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;

   }

     index = a[n-1];
     j = n-1;
     while ((j > 0) && (a[j-1] > index))
     {
        a[j] = a[j-1];
         j = j - 1;
      }
     a[j] = index;
      i++;
```

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

L(1,n)

backward unrolling

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n-1)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }

     index = a[n-1];
     j = n-1;
     while ((j > 0) && (a[j-1] > index))
     {
        a[j] = a[j-1];
         j = j - 1;
      }
     a[j] = index;
      i++;
```

L(1,n-1)

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))        L(1,n)
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

L(1,n) = L(1,n-1) ; C

backward unrolling

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n-1)
   {
     index = a[i];                  L(1,n-1)
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }

    index = a[n-1];
    j = n-1;
    while ((j > 0) && (a[j-1] > index))
     {
       a[j] = a[j-1];
       j = j - 1;                   C
     }
    a[j] = index;
    i++;
 }
```

Example

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
 }
```

L(1,n)

L(1,n) = L(1,n-1) ; C

Pre-recursive form

backward unrolling

```
void insertion_sort(int a[], int n)
 {
   int i, j, index;
   i = 1;
   while(i < n-1)
   {
     index = a[i];
     j = i;
     while ((j > 0) && (a[j-1] > index))
       {
         a[j] = a[j-1];
         j = j - 1;
       }
     a[j] = index;
     i++;

   }

    index = a[n-1];
    j = n-1;
    while ((j > 0) && (a[j-1] > index))
    {
       a[j] = a[j-1];
        j = j - 1;
     }
    a[j] = index;
     i++;
 }
```

L(1,n-1)

C

Verification task

$\{\mathrm{true}\}\ L(1,n)\ \{\varphi(n)\}$?

Applying induction

$\{true\}\ L(1,n)\ \{\varphi(n)\}$?

Applying induction

{true} L(1,n) {φ(n)} ?

- Prove {true} L(1,0) {φ(0)}

Applying induction

{true} L(1,n) {φ(n)} ?

- Prove {true} L(1,0) {φ(0)}
- Assume {true} L(1,n-1) {φ(n-1)} and prove {true} L(1,n) {φ(n)}

Assume guarantee reasoning

$$\{true\}\ L(1,n)\ \{\varphi(n)\}$$

Assume guarantee reasoning

{true} L(1,n) ; C {φ(n)}

Assume guarantee reasoning

$\{true\}\ L(1,n)\ ;\ C\ \{\varphi(n)\}$

Using Hoare's rule of composition

$$\frac{\{true\}\ L(1,n-1)\ \{\varphi(n-1)\} \qquad \{\varphi(n-1)\}\ C\ \{\varphi(n)\}}{\{true\}\ L(1,n-1)\ ;\ C\ \{\varphi(n)\}}$$

Assume guarantee reasoning

{true} L(1,n) ; C {φ(n)}

Using Hoare's rule of composition

$$\frac{\{true\}\ L(1,n-1)\ \{\varphi(n-1)\} \qquad \{\varphi(n-1)\}\ C\ \{\varphi(n)\}}{\{true\}\ L(1,n-1)\ ;\ C\ \{\varphi(n)\}}$$

{true} L(1,n-1) {φ(n-1)} is true (induction hypothesis)

Assume guarantee reasoning

{true} L(1,n) ; C {φ(n)}

Using Hoare's rule of composition

$$\frac{\{true\}\ L(1,n-1)\ \{\varphi(n-1)\} \qquad \{\varphi(n-1)\}\ C\ \{\varphi(n)\}}{\{true\}\ L(1,n-1)\ ;\ C\ \{\varphi(n)\}}$$

{true} L(1,n-1) {φ(n-1)} is true (induction hypothesis)

it suffices to prove {φ(n-1)} C {φ(n)}

Simplified task

```
void insertion_sort(int a[], int n)
{
   int i, j, index;
   i = 1;
   while(i < n)
   {
      index = a[i];
      j = i;
      while ((j > 0) && (a[j-1] > index))
        {
           a[j] = a[j-1];
           j = j - 1;
        }
      a[j] = index;
      i++;
   }
   assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
}
```

Simplified task

```
assume(∀x,y. 0≤ x,y < n-1 ∧ x > y ⇒ a[x] ≥ a[y]);
   index = a[n-1];
    j = n-1;
    while ((j > 0) && (a[j-1] > index))
      {
         a[j] = a[j-1];
         j = j - 1;
      }
    a[j] = index;
    i++;
assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
```

Simplified task

```
assume(∀x,y. 0≤ x,y < n-1 ∧ x > y ⇒ a[x] ≥ a[y]);
   index = a[n-1];
    j = n-1;
    while ((j > 0) && (a[j-1] > index))
      {
         a[j] = a[j-1];
         j = j - 1;
      }
    a[j] = index;
    i++;
assert(∀x,y. 0≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
```

Loop nesting level reduced: less loop invariants to compute

Example (2)

```
void selection_sort (int a[], int k, int n)
{
  int i, j, s;
  i = k;
  while(i < n)
    {
      s = i;
      for(j = i+1; j < n; ++j)
       {
         if(a[j] < a[s])
         {
           s = j;
         }
       }
     t = a[i];  a[i] = a[s];  a[s] = t; i++;
   }
   assert(∀x,y.  k≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
}
```

L(k,n)

Example (2)

```
void selection_sort (int a[], int k, int n)
{
  int i, j, s;
  i = k;
  while(i < n)
   {
     s = i;
     for(j = i+1; j < n; ++j)
      {
        if(a[j] < a[s])
        {
          s = j;
        }
      }
     t = a[i];  a[i] = a[s];  a[s] = t; i++;
   }
   assert(∀x,y. k ≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
}
```

L(k,n)

backward unrolling

```
i = k;
while(i < n-1)
 {
   s = i;
   for(j = i+1; j < n; ++j)
    {
      if(a[j] < a[s])
      {
        s = j;
      }
    }
   t = a[i];  a[i] = a[s];  a[s] = t;  i++;
 }
 s =  n-1;  i = n -1;
 for(j = i+1; j < n; ++j)
  {
     if(a[j] < a[s])
     {
       s = j;
     }
  }
 t = a[n-1];  a[n-1] = a[s];  a[s] = t; ++i;
```

L(k,n-1)?

Example (2)

```
i = k;
while(i < n-1)
 {
   s = i;
   for(j = i+1; j < n-1; ++j)
    {
      if(a[j] < a[s])
      {
        s = j;
      }
    }
   t = a[i];  a[i] = a[s];  a[s] = t;
   i++;
 }
```

L(k,n-1)

backward unrolling

```
i = k;
while(i < n-1)
 {
   s = i;
   for(j = i+1; j < n; ++j)
    {
      if(a[j] < a[s])
      {
        s = j;
      }
    }
   t = a[i];  a[i] = a[s];  a[s] = t;  i++;
 }
 s =  n-1; i = n-1;
 for(j = i+1; j < n; ++j)
  {
    if(a[j] < a[s])
     {
       s = j;
     }
  }
 t = a[n-1];  a[n-1] = a[s];  a[s] = t; ++i;
```

L(k,n-1)?

Example (2)

```
void selection_sort (int a[], int k, int n)
{
  int i, j, s;
  i = k;
  while(i < n)
   {
     s = i;
     for(j = i+1; j < n; ++j)
      {
        if(a[j] < a[s])
        {
          s = j;
        }
      }
     t = a[i];  a[i] = a[s];  a[s] = t;
     i++;
   }
  assert(∀x,y. k ≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
}
```

L(k,n)

forward unrolling

```
 s =  k;
 for(j = k-1; j < n; ++j)
   {
     if(a[j] < a[s])
     {
       s = j;
     }
   }
 t = a[k];  a[k] = a[s];  a[s] = t;
 i = k+1;
 while(i < n)
 {
    s = i;
    for(j = i+1; j < n; ++j)
    {
      if(a[j] < a[s])
      {
        s = j;
      }
    }
    t = a[i];  a[i] = a[s];  a[s] = t;  i++;
 }
```

L(k+1,n)

Example (2)

```
void selection_sort (int a[], int k, int n)
{
  int i, j, s;
  i = k;
  while(i < n)
   {
     s = i;
     for(j = i+1; j < n; ++j)
      {
        if(a[j] < a[s])
        {
          s = j;
        }
      }
     t = a[i];  a[i] = a[s];  a[s] = t;
     i++;
   }
   assert(∀x,y.  k≤ x,y < n ∧ x > y ⇒ a[x] ≥ a[y]);
}
```

L(k,n)

L(k,n) = C ; L(k+1,n)

Post-recursive form

forward unrolling

```
s =  k;
for(j = k_1; j < n; ++j)
  {
    if(a[j] < a[s])
    {
      s = j;
    }
  }
t = a[k];  a[k] = a[s];  a[s] = t;
i = k+1;
while(i < n)
{
  s = i;
  for(j = i+1; j < n; ++j)
  {
    if(a[j] < a[s])
    {
      s = j;
    }
  }
  t = a[i];  a[i] = a[s];  a[s] = t;  i++;
}
```

C

L(k+1,n)

Applying induction

$\{true\}\ L(k,n)\ \{\varphi(k)\}$?

- Prove $\{true\}\ L(n,n)\ \{\varphi(n)\}$
- Assume $\{true\}\ L(k+1,n)\ \{\varphi(k+1)\}$ and prove $\{true\}\ L(k,n)\ \{\varphi(k)\}$

Assume guarantee reasoning

$$\{true\}\ C\ ;\ L(k,n)\ \{\varphi(k)\}$$

Assume guarantee reasoning

{true} C ; L(k,n) {φ(k)}

Introducing induction hypothesis

{true} C ; L(k,n) ; assume (φ(k+1)) {φ(k)}

Assume guarantee reasoning

{true} C ; L(k,n) {φ(k)}

Introducing induction hypothesis

{true} C ; L(k,n) ; assume (φ(k+1)) {φ(k)}

moving the assumption into the postcondition

{true} C ; L(k,n) {φ(k+1) ⇒ φ(k)}

Example (postcondition weakening)

$$\varphi(k) \equiv \forall x,y.\ k \leq x,y < n \land x > y \Rightarrow a[x] \geq a[y]$$

$$\varphi(k+1) \equiv \forall x,y.\ k+1 \leq x,y < n \land x > y \Rightarrow a[x] \geq a[y]$$

$$\varphi(k+1) \Rightarrow \varphi(k) \equiv \forall x.\ k+1 \leq x < n \land a[x] \geq a[k]$$

Transformation algorithm

Extended report available upon request

Implementation

- Integration into ACSAR (C++)

  http://swt.informatik.uni-freiburg.de/~seghir/acsar

- Using Yices (API Lite) and Simplify as backend

  http://yices.csl.sri.com

  http://www.hpl.hp.com/downloads/crl/jtk/download-simplify.html

ACSAR

Program
+
spec

Abstract

Add facts to eliminate counterexample

Verify

No Error Trace

The system is safe

Error trace

Refine

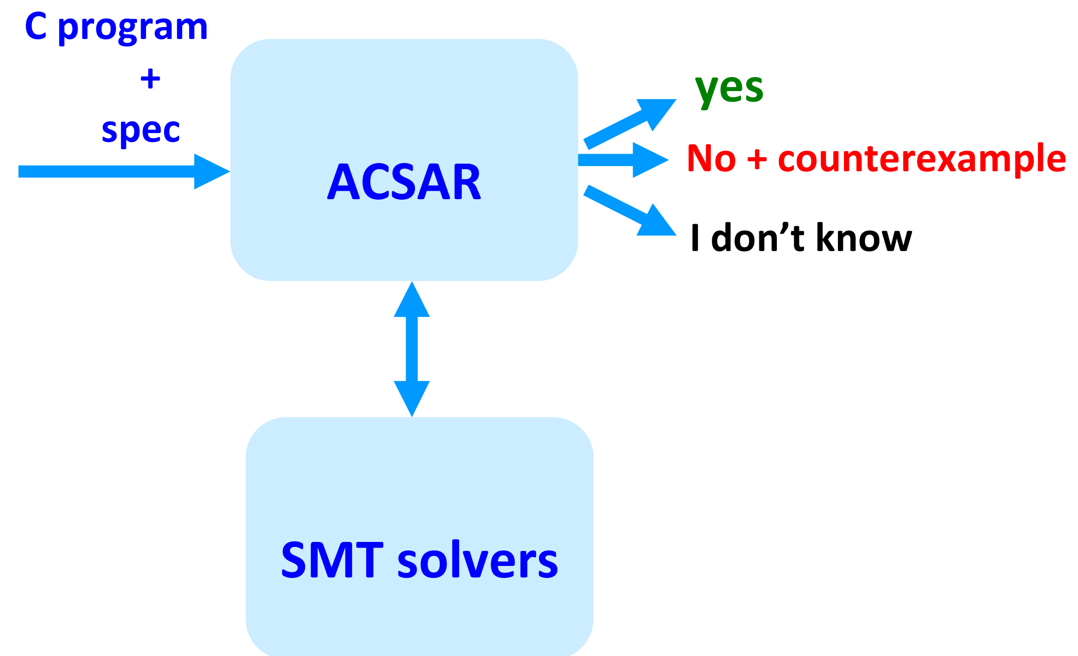
Counterexample analysis

Spurious Counter Example

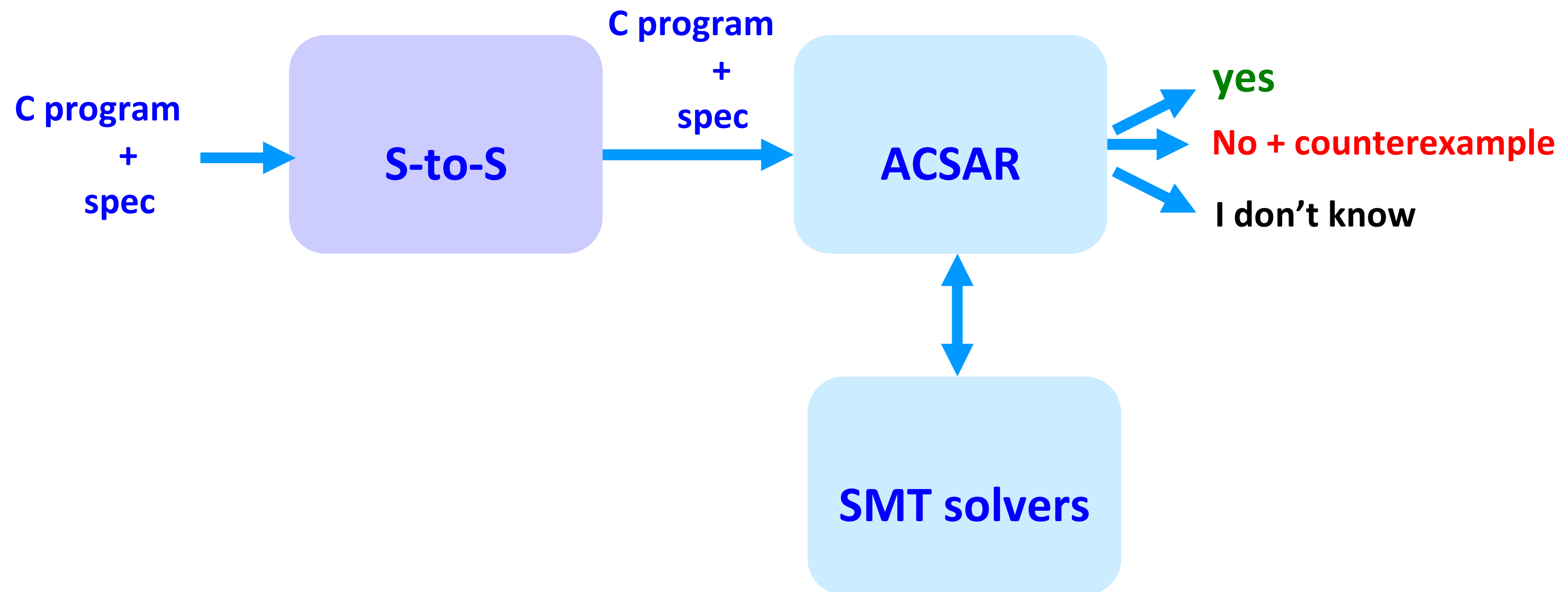
Real Counter Example

ERROR

Integration scheme



Integration scheme



Experimental results (academic)

Program	Property	Tran	Iter.		Pred.		Time(s)	
			S	T	S	T	S	T
string_copy	0 terminal string s1 is copied to s2	PS / PR•	2	2	5	4	0.39	0.41
Scan	array entries before actual entry are not null	PS / PR•	3	2	3	2	0.27	0.14
array_init	array entries are initialized	PS / PR•	2	1	6	2	0.50	0.13
loop1	each array entry is initialized with its index	PS / PR•	3	1	5	2	0.51	0.21
copy1	array *a* is copied to array *b*	PS / PR•	2	1	6	2	0.70	0.23
num_index	for every array entry *i* of array *a* we have a[i] = 2 * i + 3	PS / PR•	2	1	6	2	0.68	0.21

Experimental results (Linux + Xen hypervisor)

Program	Property	Tran	Iter.		Pred.		Time(s)	
			S	T	S	T	S	T
cyber_init	For every entry i of array *a*, if i modulo 4 is equal to 0,1,2 or 3 then a[i] is initialized to the given values v0, v1, v2, v3	PS• / PR	8	8	13	12	9.47	5.60
dvb_net_feed_stop	Entries different from 0 in their pre-state are set to 0	PS / PR•	3	2	8	3	3.41	0.30
perfect_copy_info	For each entry i of array a if a[i] has some property *p* then b[i] and c[i] should be equal	PS / PR•	4	5	16	8	10.57	1.50
do_enoprof_op	If variable op has value v1 and variable s has value v2 then array *a* is copied to array *b*	PS / PR•	3	2	14	6	8.9	0.54

Experimental results (sorting algorithms)

Program	Property	Tran	Iter.		Pred.		Time(s)	
			S	T	S	T	S	T
selection_sort	array is sorted	PS	3	5	37	35	409.87	173.50
insertion_sort	array is sorted	PR	-	5	-	39	-	145.60
bubble_sort	array is sorted	PS	-	8	-	42	-	188.90

Demo

Conclusion

- Assume guarantee approach for the verification of quantified array assertions
- Pro
  - Performance enhancement
  - Automatic verification of challenging programs (sorting algorithms)
  - Flexible and easy to integrate: source-to-source transformation
  - Other interesting properties can be handled by applying similar code manipulation techniques
- Cons
  - Applied before verification
  - Restriction on loop form

Thank you