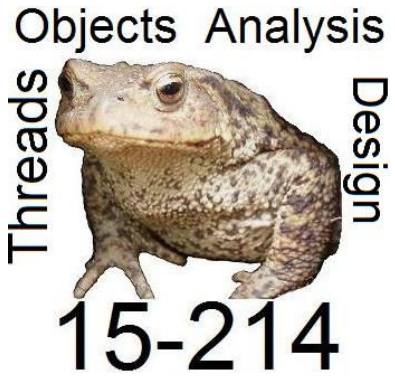
Analysis of Software Artifacts



Checking Program Properties with ESC/Java

Jonathan Aldrich

ESC/Java

- A checker for Java programs
  - Finds null pointers, array dereferences…
  - Checks Hoare logic specifications
    - Expressed in Java Modeling Language (JML)
- Goal:
  - Find errors
  - Increase confidence in correctness
    - Unlike a Hoare Logic proof, not a guarantee of correctness
- Developed at DEC/Compaq SRC in the 90s
  - Now open sourced as ESC/Java 2

ESC/Java Uses JML Specifications

```
/*@ requires len >= 0 && array != null && array.length == len;
  @
  @ ensures \result == (\sum int j;  0 <= j && j < len;  array[j]);
  @*/
int total(int array[], int len) {
    int answer = 0;
    int i = 0;
    /*@ loop_invariant answer == (\sum int j;  0 <= j && j < i;  array[j]); */
    while (i < len) {
        answer = answer + array[i];
        i = i + 1;
    }
    return answer;
}
```

Modular Checking with Hoare Logic

procedure multiply

```
    product := 0;
    i := 0;


    while i < n do
       product := add(product, m);
       i := i + 1;


// modifies result
int add(int n, int m)

    return n + m
```

Modular checking: verifying one procedure using *only the specification* of other procedures

Modular Checking with Hoare Logic

procedure multiply

```
    product := 0;
    i := 0;


    while i < n do
       product := add(product, m);
       i := i + 1;
    { product = m*n }

// modifies result
int add(int n, int m)

    return n + m
```

Modular Checking with Hoare Logic

```
procedure multiply
    { n > 0 }
    product := 0;
    i := 0;


    while i < n do
       product := add(product, m);
       i := i + 1;
    { product = m*n }

// modifies result
int add(int n, int m)

    return n + m
```

Modular Checking with Hoare Logic

```
procedure multiply
    { n > 0 }
    product := 0;
    i := 0;


    while i < n do
       product := add(product, m);
       i := i + 1;
    { product = m*n }

// modifies result
int add(int n, int m)

    return n + m
    { \result = n+m }
```

Modular Checking with Hoare Logic

```
procedure multiply
    { n > 0 }
    product := 0;
    i := 0;


    while i < n do
       product := add(product, m);
       i := i + 1;
    { product = m*n }

// modifies result
int add(int n, int m)
    { true }
    return n + m
    { \result = n+m }
```

Modular Checking with Hoare Logic

```
procedure multiply
    { n > 0 }
    product := 0;
    i := 0;
    { invariant: product = m*i
    && 0 ≤ i ≤ n && n > 0 }
    while i < n do
       product := add(product, m);
       i := i + 1;
    { product = m*n }

// modifies result
int add(int n, int m)
    { true }
    return n + m
    { \result = n+m }
```

Modular Checking with Hoare Logic

procedure multiply
{ n > 0 }
product := 0;
i := 0;
*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
product := add(product, m);
i := i + 1;
*{ product = m*n }*

// modifies result
int add(int n, int m)
{ true }
return n + m
{ \result = n+m }

- Verification condition for loop:

*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
*{product = m*i && 0 ≤ i ≤ n*
&& n > 0 && i < n}

product := add(product, m);

i := i + 1;
*{product = m*i && 0 ≤ i ≤ n && n>0 }*

Modular Checking with Hoare Logic

procedure multiply
{ n > 0 }
product := 0;
i := 0;
*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
product := add(product, m);
i := i + 1;
*{ product = m*n }*

// modifies result
int add(int n, int m)
{ true }
return n + m
{ \result = n+m }

- Verification condition for loop:

*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
*{product = m*i && 0 ≤ i ≤ n*
&& n > 0 && i < n}

product := add(product, m);
{product = m(**i+1**) && 0 ≤ **i+1** ≤ n && n>0}*
i := i + 1;
*{product = m*i && 0 ≤ i ≤ n && n>0 }*

Modular Checking with Hoare Logic

procedure multiply
{ n > 0 }
product := 0;
i := 0;
*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
product := add(product, m);
i := i + 1;
*{ product = m*n }*

// modifies result
int add(int n, int m)
{ true }
return n + m
{ \result = n+m }

- Verification condition for loop:

*{ invariant: product = m*i*
&& 0 ≤ i ≤ n && n > 0 }
while i < n do
*{product = m*i && 0 ≤ i ≤ n*
&& n > 0 && i < n}

*{**add(product,m)**=m*(i+1)&&0≤i+1≤n&&n>0}*
product := add(product, m);
{product = m(**i+1**) && 0 ≤ **i+1** ≤ n && n>0}*
i := i + 1;
*{product = m*i && 0 ≤ i ≤ n && n>0 }*

Modular Checking with Hoare Logic

procedure multiply

{ n > 0 }

product := 0;

i := 0;

*{ invariant: product = m*i*

&& 0 ≤ i ≤ n && n > 0 }

while i < n do

product := add(product, m);

i := i + 1;

*{ product = m*n }*

// modifies result

int add(int n, int m)

{ true }

return n + m

{ \result = n+m }

- Verification condition for loop:

*{ invariant: product = m*i*

&& 0 ≤ i ≤ n && n > 0 }

while i < n do

*{product = m*i && 0 ≤ i ≤ n*

&& n > 0 && i < n}

*{**product+m** = m*(i+1) && 0≤i+1≤n && n>0}*

*{**add(product,m)**=m*(i+1)&&0≤i+1≤n&&n>0}*

product := add(product, m);

{product = m(**i+1**) && 0 ≤ **i+1** ≤ n && n>0}*

i := i + 1;

*{product = m*i && 0 ≤ i ≤ n && n>0 }*

Multiply in ESC/Java (1)

```
class Multiply {


    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Multiply in ESC/Java (1)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Start with postcondition

Multiply in ESC/Java (2)

```
$ ../escj.bat Multiply.java

c:\develop\escjava\LG>"C:\develop\Java\jdk1.4.2\bin\java.exe"  -Dsimplify=C:\dev
elop\escjava\Simplify-1.5.4.exe -classpath "C:\develop\escjava\esctools2.jar" es
cjava.Main -classpath   C:\develop\escjava\jmlspecs.jar -classpath . -nowarn Dea
dlock Multiply.java
ESC/Java version ESCJava-2.0a9
    [0.06 s 4528552 bytes]

Multiply ...
  Prover started:0.03 s 4597272 bytes
    [0.21 s 4842000 bytes]

Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:9: Warning: Postcondition possibly not established (Post)
    }
    ^
Associated declaration is "Multiply.java", line 2, col 8:
    //@ ensures \result == m*n;
        ^
Execution trace information:
    Reached top of loop after 0 iterations in "Multiply.java", line 6, col 1.

------------------------------------------------------------------------
    [0.081 s 4743064 bytes]  failed

Multiply: add(int, int) ...
    [0.02 s 4810816 bytes]  passed

Multiply: Multiply() ...
    [0.01 s 4442112 bytes]  passed
  [0.321 s 4442840 bytes total]
1 warning
```

Multiply in ESC/Java (2)

```
$ ../escj.bat Multiply.java

c:\develop\escjava\LG>"C:\develop\Java\jdk1.4.2\bin\java.exe"  -Dsimplify=C:\dev
elop\escjava\Simplify-1.5.4.exe -classpath "C:\develop\escjava\esctools2.jar" es
cjava.Main -classpath   C:\develop\escjava\jmlspecs.jar -classpath . -nowarn Dea
dlock Multiply.java
ESC/Java version ESCJava-2.0a9
    [0.06 s 4528552 bytes]

Multiply ...
  Prover started:0.03 s 4597272 bytes
    [0.21 s 4842000 bytes]

Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:9: Warning: Postcondition possibly not established (Post)
    }
    ^
Associated declaration is "Multiply.java", line 2, col 8:
    //@ ensures \result == m*n;
        ^
Execution trace information:
    Reached top of loop after 0 iterations in "Multiply.java", line 6, col 1.

------------------------------------------------------------------------
    [0.081 s 4743064 bytes]  failed

Multiply: add(int, int) ...
    [0.02 s 4810816 bytes]  passed

Multiply: Multiply() ...
    [0.01 s 4442112 bytes]  passed
  [0.321 s 4442840 bytes total]
1 warning
```

Multiply in ESC/Java (2)

```
$ ../escj.bat Multiply.java

c:\develop\escjava\LG>"C:\develop\Java\jdk1.4.2\bin\java.exe"  -Dsimplify=C:\dev
elop\escjava\Simplify-1.5.4.exe -classpath "C:\develop\escjava\esctools2.jar" es
cjava.Main -classpath   C:\develop\escjava\jmlspecs.jar -classpath . -nowarn Dea
dlock Multiply.java
ESC/Java version ESCJava-2.0a9
    [0.06 s 4528552 bytes]

Multiply ...
  Prover started:0.03 s 4597272 bytes
    [0.21 s 4842000 bytes]

Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:9: Warning: Postcondition possibly not established (Post)
    }
    ^
Associated declaration is "Multiply.java", line 2, col 8:
    //@ ensures \result == m*n;
        ^
Execution trace information:
    Reached top of loop after 0 iterations in "Multiply.java", line 6, col 1.

------------------------------------------------------------------------
    [0.081 s 4743064 bytes]  failed

Multiply: add(int, int) ...
    [0.02 s 4810816 bytes]  passed

Multiply: Multiply() ...
    [0.01 s 4442112 bytes]  passed
  [0.321 s 4442840 bytes total]
1 warning
```

Multiply in ESC/Java (2)

```
$ ../escj.bat Multiply.java

c:\develop\escjava\LG>"C:\develop\Java\jdk1.4.2\bin\java.exe"  -Dsimplify=C:\dev
elop\escjava\Simplify-1.5.4.exe -classpath "C:\develop\escjava\esctools2.jar" es
cjava.Main -classpath   C:\develop\escjava\jmlspecs.jar -classpath . -nowarn Dea
dlock Multiply.java
ESC/Java version ESCJava-2.0a9
    [0.06 s 4528552 bytes]

Multiply ...
  Prover started:0.03 s 4597272 bytes
    [0.21 s 4842000 bytes]

Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:9: Warning: Postcondition possibly not established (Post)
    }
    ^
Associated declaration is "Multiply.java", line 2, col 8:
    //@ ensures \result == m*n;
        ^
Execution trace information:
    Reached top of loop after 0 iterations in "Multiply.java", line 6, col 1.

------------------------------------------------------------------------
    [0.081 s 4743064 bytes]  failed

Multiply: add(int, int) ...
    [0.02 s 4810816 bytes]  passed

Multiply: Multiply() ...
    [0.01 s 4442112 bytes]  passed
  [0.321 s 4442840 bytes total]
1 warning
```

Multiply in ESC/Java (3)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Multiply in ESC/Java (3)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Error occurs when we skip the loop, so n <= i

Multiply in ESC/Java (3)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Error occurs when we skip the loop, so n <= i

But code is correct if n = i

Multiply in ESC/Java (3)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Error occurs when we skip the loop, so n <= i

But code is correct if n = i

Error comes if n < i

Multiply in ESC/Java (3)

```
class Multiply {

    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Error occurs when we skip the loop, so n <= i

But code is correct if n = i

Error comes if n < i

But this should be illegal!

Multiply in ESC/Java (3)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Add precondition

Error occurs when we skip the loop, so n <= i

But code is correct if n = i

Error comes if n < i

But this should be illegal!

Multiply in ESC/Java (4)

```
Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:12: Warning: Postcondition possibly not established

Associated declaration is "Multiply.java", line 2, col 8:
   //@ ensures \result == m*n;

Execution trace information:
   Reached top of loop after 0 iterations in "Multiply.java", line 7, col 1.
   Reached top of loop after 1 iteration in "Multiply.java", line 7, col 1.
   Executed return in "Multiply.java", line 11, col 1.
```

Multiply in ESC/Java (4)

```
Multiply: multiply(int, int) ...
------------------------------------------------------------------------
Multiply.java:12: Warning: Postcondition possibly not established

Associated declaration is "Multiply.java", line 2, col 8:
   //@ ensures \result == m*n;

Execution trace information:
   Reached top of loop after 0 iterations in "Multiply.java", line 7, col 1.
   Reached top of loop after 1 iteration in "Multiply.java", line 7, col 1.
   Executed return in "Multiply.java", line 11, col 1.
```

Unclear what causes this problem.

However, it's in a loop, so adding a loop invariant may help

Multiply in ESC/Java (5)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;

        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Multiply in ESC/Java (5)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;
        //@ loop_invariant product==m*i && i>=0 && i<=n && n>0;
        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Multiply in ESC/Java (6)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;
        //@ loop_invariant product==m*i && i>=0 && i<=n && n>0;
        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Now ESC/Java complains that the loop invariant may not hold after the first loop iteration.

Multiply in ESC/Java (6)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;
        //@ loop_invariant product==m*i && i>=0 && i<=n && n>0;
        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }

    int add(int n, int m) {
        return n+m;
    }
}
```

Now ESC/Java complains that the loop invariant may not hold after the first loop iteration.

We notice that add was unspecified and we add a postcondition

Multiply in ESC/Java (6)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;
        //@ loop_invariant product==m*i && i>=0 && i<=n && n>0;
        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }
    //@ ensures \result == n + m;
    int add(int n, int m) {
        return n+m;
    }
}
```

Now ESC/Java complains that the loop invariant may not hold after the first loop iteration.

We notice that add was unspecified and we add a postcondition

Multiply in ESC/Java (6)

```
class Multiply {
    //@ requires n > 0
    //@ ensures \result == m*n;
    int multiply(int m, int n) {
        int product = 0;
        int i = 0;
        //@ loop_invariant product==m*i && i>=0 && i<=n && n>0;
        while (i < n) {
            product = add(product, m);
            i = i + 1;
        }
        return product;
    }
    //@ ensures \result == n + m;
    int add(int n, int m) {
        return n+m;
    }
}
```

Now ESC/Java complains that the loop invariant may not hold after the first loop iteration.

We notice that add was unspecified and we add a postcondition

ESC/Java report success!

Check Your Understanding

Consider the following function:

```
int subtract(int x, int y) {
    return x + y;
}
```

This code clearly does not correctly implement subtraction. However, if you run ESC/Java on it, you will not get any errors. Why not?

Specifying Classes

```
public class SimpleSet {

   int contents[];
   int size;


   SimpleSet(int capacity) { … }



   boolean add(int i) { … }



   boolean contains(int i) { … }
}
```

- Define a SimpleSet class
  - constructor
  - add, contains methods, etc.
- Representation: sorted array
  - add inserts the element in place
  - contains uses binary search
- How can we ensure the array is sorted when contains is called?

Specifying Classes

```
public class SimpleSet {

   int contents[];
   int size;

   //@ ensures sorted(contents);
   SimpleSet(int capacity) { … }

   //@ requires sorted(contents);
   //@ ensures sorted(contents);
   boolean add(int i) { … }

   //@ requires sorted(contents);
   //@ ensures sorted(contents);
   boolean contains(int i) { … }
}
```

- Define a SimpleSet class
  - constructor
  - add, contains methods, etc.
- Representation: sorted array
  - add inserts the element in place
  - contains uses binary search
- How can we ensure the array is sorted when contains is called?

- One solution: preconditions and postconditions everywhere
  - Is this a good solution?

Data Invariants

- Data Invariant
  - A predicate that is true on every entry and exit of ADT functions
    - Not usually part of public specification
    - On function entry, you can count on the invariant because the last function left the data in a good state
    - Must verify that all ADT functions preserve invariant
  - Does not have to be true in middle of function
    - May violate invariant temporarily while updating state
- Motivation
  - Reduces duplication in pre-/post-conditions
    - e.g. sorted(s)
  - Hides representation decision from clients
    - Why should public spec say the array is sorted? That's an internal decision that shouldn't affect clients if it were to change.

SimpleSet in ESC/Java (1)

```
public class SimpleSet {

   int contents[];
   int size;



   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet in ESC/Java (1)

```
public class SimpleSet {

   int contents[];
   int size;



   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet.java:6: Warning: Possible attempt to allocate array of negative length
contents = new int[capacity];

SimpleSet in ESC/Java (1)

```
public class SimpleSet {

   int contents[];
   int size;



   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet.java:6: Warning: Possible attempt to allocate array of negative length
contents = new int[capacity];

- Need to add precondition

SimpleSet in ESC/Java (1)

```
public class SimpleSet {

   int contents[];
   int size;



   //@ requires capacity >= 0;
   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet.java:6: Warning: Possible attempt to allocate array of negative length
contents = new int[capacity];

- Need to add precondition

SimpleSet in ESC/Java (2)

```
public class SimpleSet {

   int contents[];
   int size;



   //@ requires capacity >= 0;
   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet.java:14: Warning: Possible null dereference (Null)
contents[size++]=i;

SimpleSet in ESC/Java (2)

```
public class SimpleSet {

   int contents[];
   int size;



   //@ requires capacity >= 0;
   SimpleSet(int capacity) {
      contents = new int[capacity];
      size = 0;
   }


   boolean add(int i) {
      if (contains(i))
         return false;
      contents[size++]=i;
      return true;
   }

   boolean contains(int i) {
      for (int j = 0; j < size; ++j)
         if (contents[j] == i) {
               return true;
         }
      return false;
   }
}
```

SimpleSet.java:14: Warning: Possible null dereference (Null)
contents[size++]=i;

- Need to add data invariant

SimpleSet in ESC/Java (2)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;



  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }
```

```
  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
            return true;
      }
    return false;
  }
}
```

SimpleSet.java:14: Warning: Possible null dereference (Null)
contents[size++]=i;

- Need to add data invariant

SimpleSet in ESC/Java (3)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;



  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }
```

```
  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
          return true;
      }
    return false;
  }
}
```

SimpleSet.java:15: Warning: Possible negative array index
contents[size++]=i;

SimpleSet in ESC/Java (3)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;



  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }
```

```
  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
            return true;
      }
    return false;
  }
}
```

SimpleSet.java:15: Warning: Possible negative array index
contents[size++]=i;

- Need to add data invariant

SimpleSet in ESC/Java (3)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;
  //@ invariant size >= 0;


  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }

  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
           return true;
      }
    return false;
  }
}
```

SimpleSet.java:15: Warning: Possible negative array index
contents[size++]=i;

- Need to add data invariant

SimpleSet in ESC/Java (4)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;
  //@ invariant size >= 0;


  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }

  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
            return true;
      }
    return false;
  }
}
```

```
SimpleSet.java:17: Warning: Array index
      possibly too large
contents[size++]=i;
```

SimpleSet in ESC/Java (4)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;
  //@ invariant size >= 0;


  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }


  boolean add(int i) {
    if (contains(i))
       return false;
    contents[size++]=i;
    return true;
  }

  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
            return true;
      }
    return false;
  }
}
```

SimpleSet.java:17: Warning: Array index possibly too large
contents[size++]=i;

- Need to add data invariant *and* precondition

SimpleSet in ESC/Java (4)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;
  //@ invariant size >= 0;
  //@ invariant size <= contents.length;

  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }

  //@ requires size < contents.length;
  boolean add(int i) {
    if (contains(i))
      return false;
    contents[size++]=i;
    return true;
  }

  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
        return true;
      }
    return false;
  }
}
```

SimpleSet.java:17: Warning: Array index possibly too large
contents[size++]=i;

- Need to add data invariant *and* precondition

SimpleSet in ESC/Java (4)

```
public class SimpleSet {
  //@ non_null
  int contents[];
  int size;
  //@ invariant size >= 0;
  //@ invariant size <= contents.length;

  //@ requires capacity >= 0;
  SimpleSet(int capacity) {
    contents = new int[capacity];
    size = 0;
  }

  //@ requires size < contents.length;
  boolean add(int i) {
    if (contains(i))
      return false;
    contents[size++]=i;
    return true;
  }

  boolean contains(int i) {
    for (int j = 0; j < size; ++j)
      if (contents[j] == i) {
        return true;
      }
    return false;
  }
}
```

SimpleSet.java:17: Warning: Array index possibly too large
contents[size++]=i;

- Need to add data invariant *and* precondition
- No more warnings for SimpleSet implementation

Adding Test Cases for SimpleSet

```
class SimpleSetDriver {
   void testConstructor() {
     SimpleSet s = new SimpleSet(2);
     //@ assert !s.contains(5);
   }

   void testAdd() {
     SimpleSet s = new SimpleSet(2);
     s.add(5);
     //@ assert !s.contains(4);
     //@ assert s.contains(5);
     s.add(4);
     //@ assert s.contains(4);
     s.add(4);
   }
}
```

- assert encodes tests
  - ESC/Java tries to prove assert will not fail

Adding Test Cases for SimpleSet

```
class SimpleSetDriver {
   void testConstructor() {
     SimpleSet s = new SimpleSet(2);
     //@ assert !s.contains(5);
   }

   void testAdd() {
     SimpleSet s = new SimpleSet(2);
     s.add(5);
     //@ assert !s.contains(4);
     //@ assert s.contains(5);
     s.add(4);
     //@ assert s.contains(4);
     s.add(4);
   }
}
```

- assert encodes tests
  - ESC/Java tries to prove assert will not fail
- Constructor
  - New object doesn't contain a sample element

Adding Test Cases for SimpleSet

```
class SimpleSetDriver {
   void testConstructor() {
     SimpleSet s = new SimpleSet(2);
     //@ assert !s.contains(5);
   }

   void testAdd() {
     SimpleSet s = new SimpleSet(2);
     s.add(5);
     //@ assert !s.contains(4);
     //@ assert s.contains(5);
     s.add(4);
     //@ assert s.contains(4);
     s.add(4);
   }
}
```

- assert encodes tests
  - ESC/Java tries to prove assert will not fail
- Constructor
  - New object doesn't contain a sample element
- Add
  - Adds the relevant element, not others
  - Can call add multiple times on an element, and it's only added once

SimpleSet in ESC/Java (5)

```
public class SimpleSet {
   //@ non_null
   int contents[];
   int size;

   //@ invariant size >= 0;
   //@ invariant size <= contents.length;

   //@ requires capacity >= 0;
   //@ ensures contents.length == capacity;
   //@ ensures size == 0;
   SimpleSet(int capacity) {
     contents = new int[capacity];
     size = 0;
   }
```

SimpleSet in ESC/Java (6)

```
//@ requires size < contents.length || contains(i);
//@ ensures \result == (size == \old(size+1));
//@ ensures !\result == (size == \old(size));
//@ ensures (\forall int j; (0 <= j && j < \old(size))
      ==> contents[j] == \old(contents[j]));
//@ ensures contains(i);

boolean add(int i) {
  if (contains(i))
     return false;
  contents[size++]=i;
  return true;
}
```

SimpleSet in ESC/Java (7)

```
//@ ensures \result == (\exists int j; 0 <= j && j < size && contents[j]==i);

boolean contains(int i) {
  for (int j = 0; j < size; ++j)
     if (contents[j] == i) {
          return true;
     }
  return false;
}
```

Check Your Understanding

Consider the following function:

```
void getF(FHolder fHolder) {
    return fHolder.f;
}
```

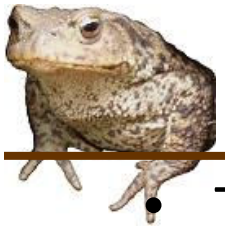
ESC/Java will give an error of the form "Warning: Possible null dereference (Null)." What is the best way to eliminate this warning?

ESC/Java's Limitations

- Does not check for some errors
  - Infinite loops, arithmetic overflow
  - Functional properties not stated by user
  - Non-functional properties

ESC/Java's Limitations

- Does not check for some errors
  - Infinite loops, arithmetic overflow
  - Functional properties not stated by user
  - Non-functional properties
- Unsound: may miss some errors
  - Only checks one iteration of loops
  - @modifies is unchecked
  - Assumptions about invariants in referred-to objects
  - Several others as well!



Loops in ESC/Java

- The loop:

```
//@ loop_invariant E;
while (B) {
  S
}
```

- Is treated as:

```
//@ assert E;
if (B) {
  S
  //@ assert E;
  //@ assume !B;
}
```

- Can optionally increase # iterations with *-loop n*

ESC/Java's Limitations (con't)

- May report false positives
  - Often can be solved with an extra precondition or invariant
  - Spurious warnings can also be disabled

ESC/Java Tradeoffs

- Attempts to automate Hoare-logic style checking
- Benefits

- Drawbacks

- Applicability

ESC/Java Tradeoffs

- Attempts to automate Hoare-logic style checking
- Benefits
  - Easier than manual proof
- Drawbacks
  - Unsound
  - Still quite labor-intensive
- Applicability
  - Checking of critical code
    - When it's worth the extra effort to get it right
    - When you can't do a complete Hoare-logic proof
  - Still must use other techniques
    - ESC/Java is unsound
    - The spec must also be validated!

Session Summary

- Tools such as ESC/Java can make Hoare Logic-style checking much more practical
  - Reduces effort relative to proof by hand
    - Still considerable work in writing specifications and invariants
  - Can be useful in documenting code and finding errors
  - The tool may miss some defects

Further Reading

- Cormac Flanagan, K. Runstan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. **Extended Static Checking for Java.** Proc. *Programming Language Design and Implementation*, June 2002.