ESC /Java 2

Extended Static Checking / Java 2

Melissa J Fernandes

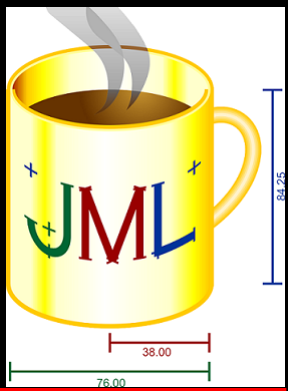
Charan Cherukuri

Srikanth Vadada

CSE 6323 , Spring 2010 , 29th April 2010

Agenda

- ✓ ***Introduction***
- ✓ ***Tool Architecture***
- ✓ ***Discovering Errors with ESC/Java 2***
- ✓ ***Tool Demo – Stack Example***
- ✓ ***ESC/Java 2 Features***
- ✓ ***Conclusion***
- ✓ ***Question & Answers***



History of Extended Static Checking

- 1950 - 1960
  - ✓ Focus on Modern Programming Languages (FORTRAN , LISP, COBOL)
- 1967 - 1978
  - ✓ Focus on Establishing Fundamental Paradigms (System , OO , Logic)
- 1980 - 1984
  - ✓ Focus on Re-Use, Performance (C++..)
- 1990 - 1997 and ….
  - ✓ Internet Age & Rapid Application Development (Java , PHP, Ruby….)
- 1997 - Till Date
  - ✓ Focus on Security and Reliability Verification to the Languages
  - ✓ Birth of Extended Static Checking
  - ✓ Pioneering effort in the use of Static Program Analysis & Verification Methods
  - ✓ ESC for Modula in 1995
  - ✓ ESC / Java in 1997 from DEC
  - ✓ Renaissance of ESC/ Java 2 in 2002 as an Industrial Strength Tool

Classes of Checkers

- Static Checking
  - Ÿ Type Checking
  - Ÿ Extended Static Checking
  - Ÿ Program Verification
- Dynamic Checking
- Coverage vs Effort ?

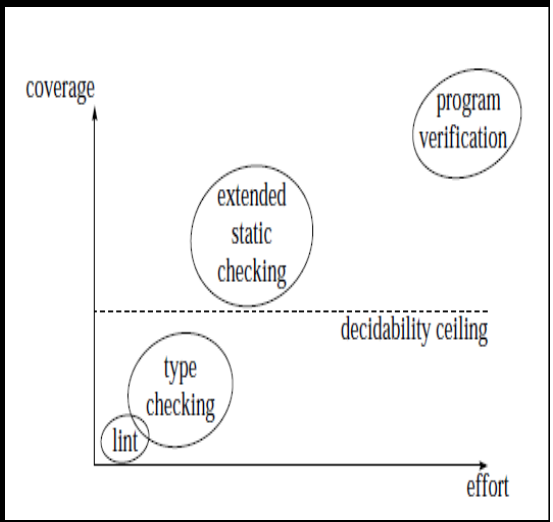


Fig. Source: Extended Static Checking: a Ten-Year Perspective by K. Rustan M. Leino

Theoretical Foundation of Extended Static Checking

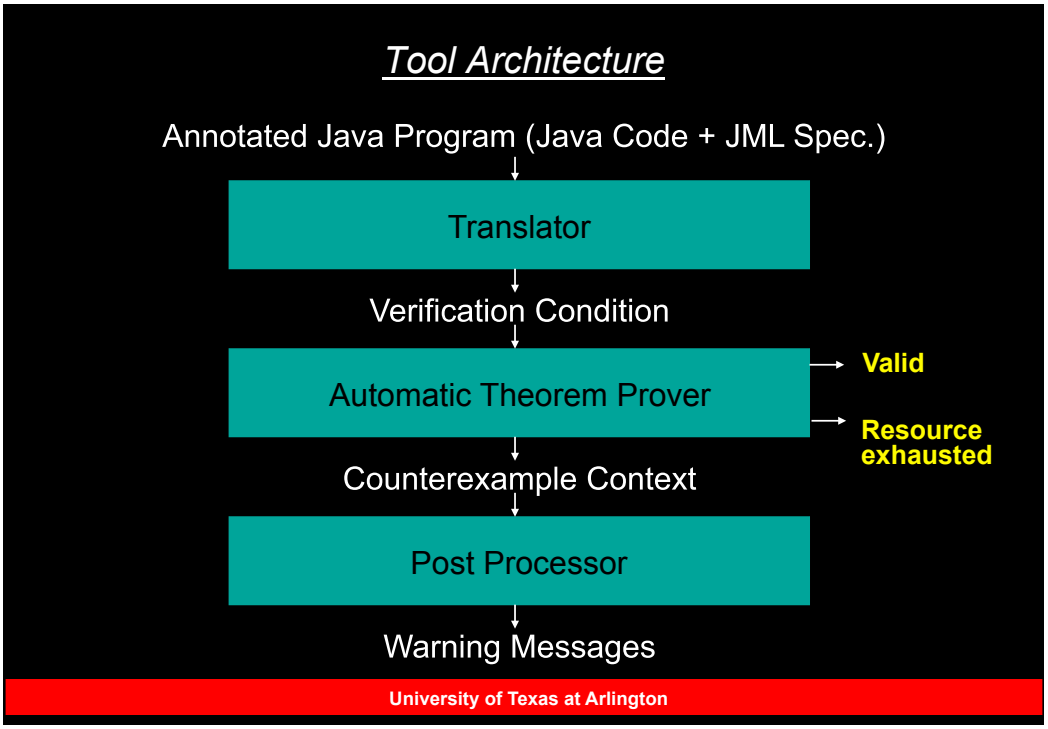
- Deciding which errors to Check
  - *Unsoundness – Missing Errors*
  - *Checks 3 Types of Errors*
    - *Runtime Checks (null dereferences, array index bounds errors…)*
    - *Synchronization Errors (race conditions , deadlocks)*
    - *Violation of Program Annotations (meeting invariants, preconditions…)*
- Defining Formal Semantics for Modern Languages
  - *Guarded Command Languages*
- Using a Theorem Prover
  - *Should be Automated – Else Learning Curve High*
  - *Produce Counter Examples –Reason for Error*
  - *Should be fast – Checker used many times during Development*
- Producing meaningful Warning Messages
- Program Annotations

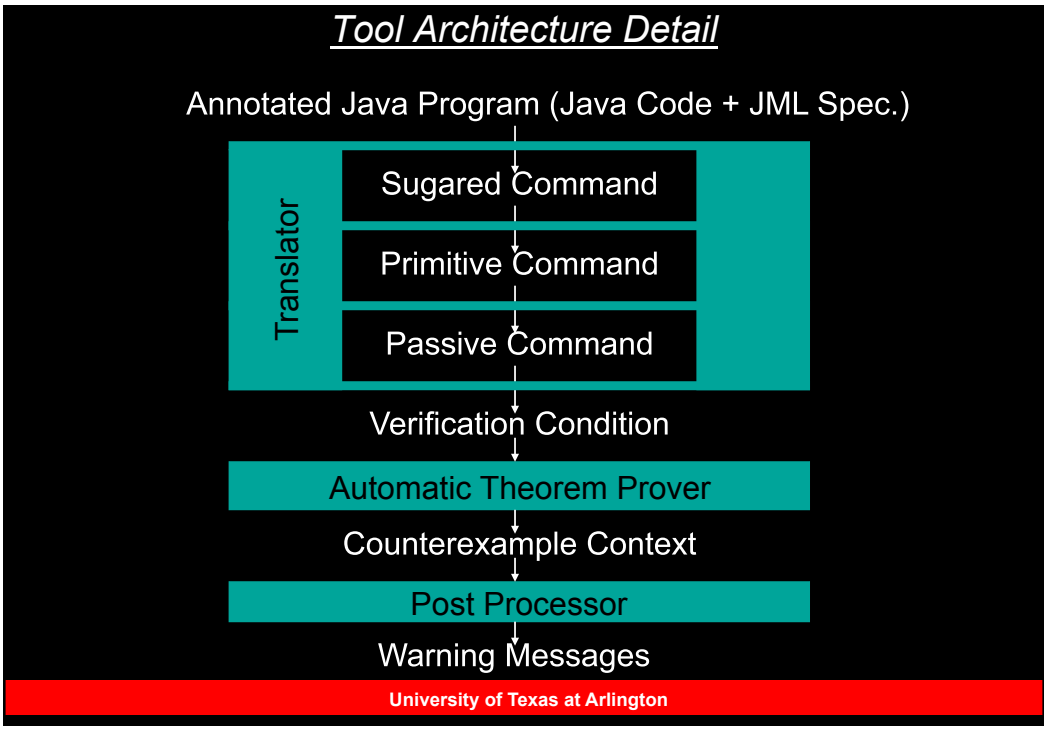
User's View

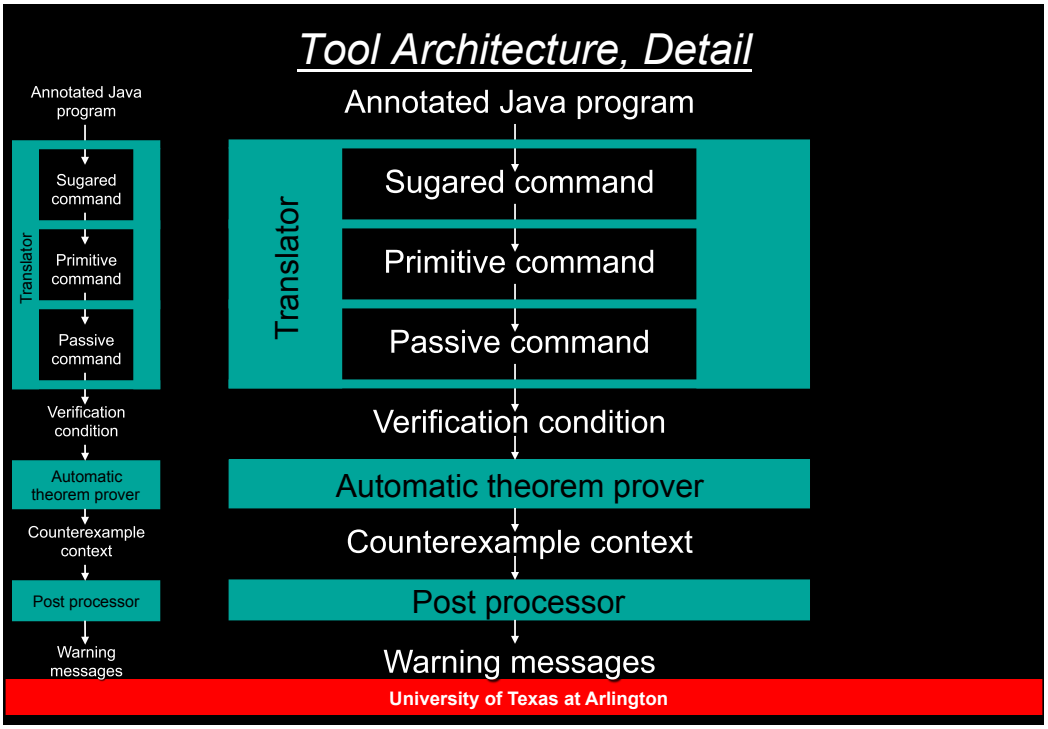

```
public class Bag {
    private /*@non_null*/ int[] a;
    private int n;
    //@ invariant 0 <= n && n <= a.length;
    public Bag(/*@non_null*/ int[] initialElements) {
        n = initialElements.length;
        a = new int[n];
        System.arraycopy(initialElements, 0, a, 0, n);
    }
    ..............
    ..................
```

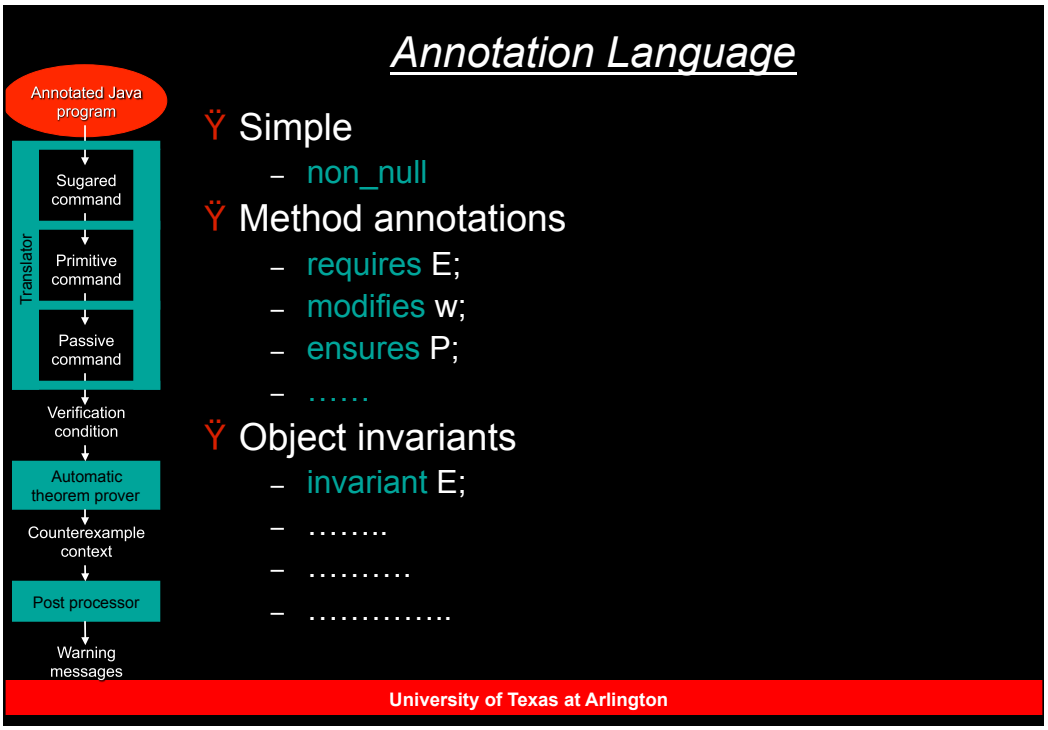
Bag.java:18:

Array index possibly too large

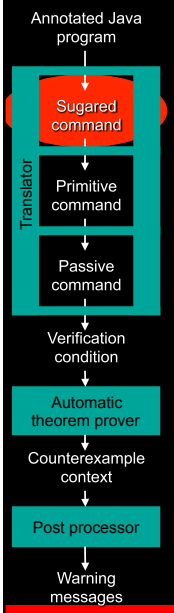

Tool Architecture
Annotated Java Program (Java Code + JML Spec.)
Translator
Verification Condition
Automatic Theorem Prover
Valid
Resource exhausted
Counterexample Context
Post Processor
Warning Messages
University of Texas at Arlington


Tool Architecture Detail
Annotated Java Program (Java Code + JML Spec.)
Translator
Sugared Command
Primitive Command
Passive Command
Verification Condition
Automatic Theorem Prover
Counterexample Context
Post Processor
Warning Messages
University of Texas at Arlington


Tool Architecture, Detail
Annotated Java program
Sugared command
Primitive command
Passive command
Translator
Verification condition
Automatic theorem prover
Counterexample context
Post processor
Warning messages
Annotated Java program
Translator
Sugared command
Primitive command
Passive command
Verification condition
Automatic theorem prover
Counterexample context
Post processor
Warning messages
University of Texas at Arlington

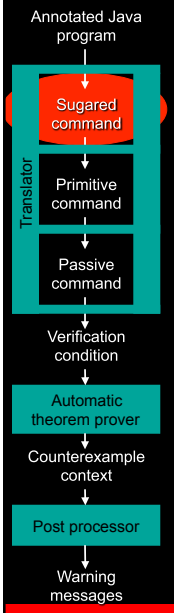

Annotation Language
Annotated Java program
Sugared command
Primitive command
Passive command
Translator
Verification condition
Automatic theorem prover
Counterexample context
Post processor
Warning messages
Simple
non_null
Method annotations
requires E;
modifies w;
ensures P;
……
Object invariants
invariant E;
……..
…….
……..
University of Texas at Arlington

Sugared Commands



Ÿ S,T ::= assert E
| assume E
| x = E
| raise
| S ; T
| S ! T
| S [] T
| loop {inv E} S → T end
| call x = t.m(E)
| …

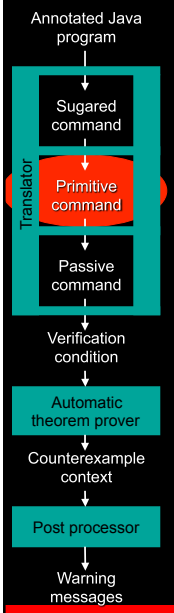
Sugared Commands



- v = o.f; ……..at Line 27
  check Null,27,o!=null;
  v=select(f,o);

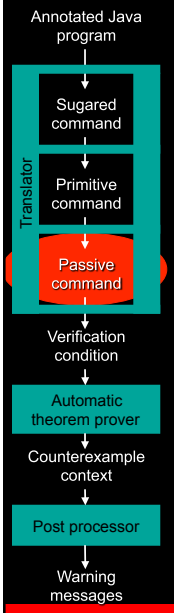
- if (x < 0) { x = -x; }
  /*@ assert x >= 0; */
  (assume x < 0; x = -x
  [] assume !(x < 0)
  );
  assert x >= 0

Primitive Commands



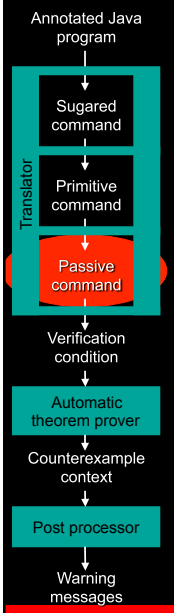
- v = o.f; ……..at Line 27
  check Null,27,o!=null;
  v=select(f,o);
  assert (label Null@27:o!=null);
  or
  assume o!=null;

Passive Commands



- S,T ::= assert E
  | assume E
  | x = E
  | raise
  | S ; T
  | S ! T
  | S [] T

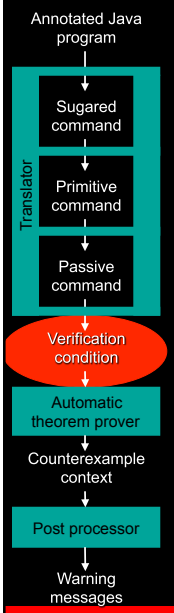
Passive Commands



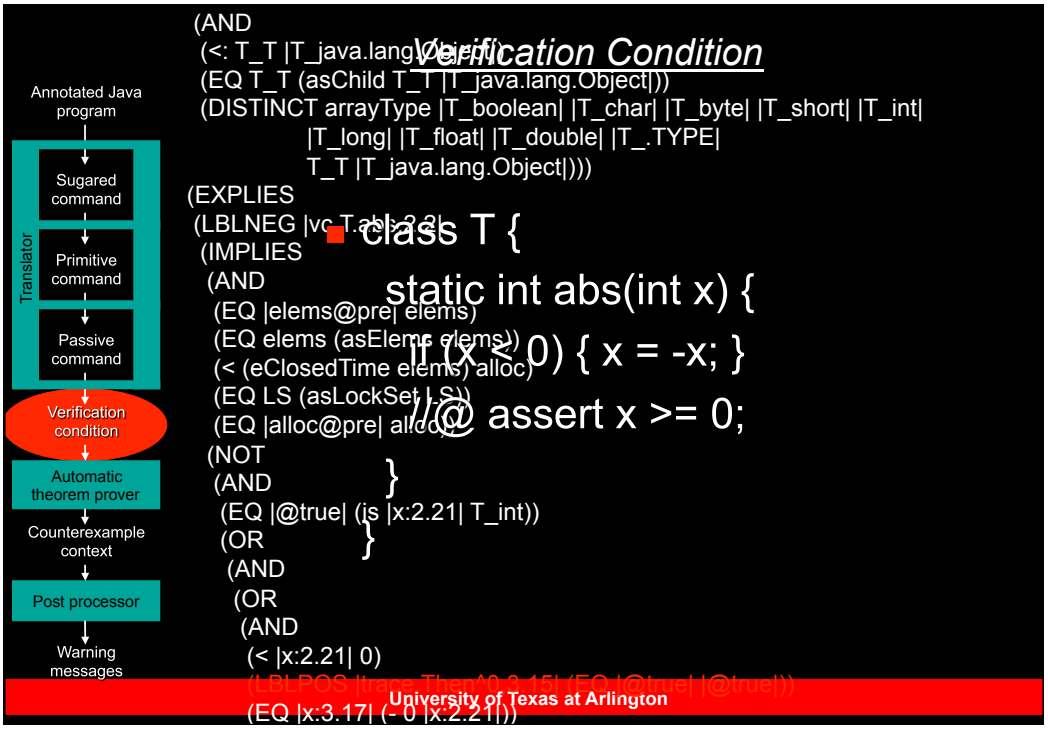
- if (x < 0) { x= -x; }
  /*@ assert x >= 0; */

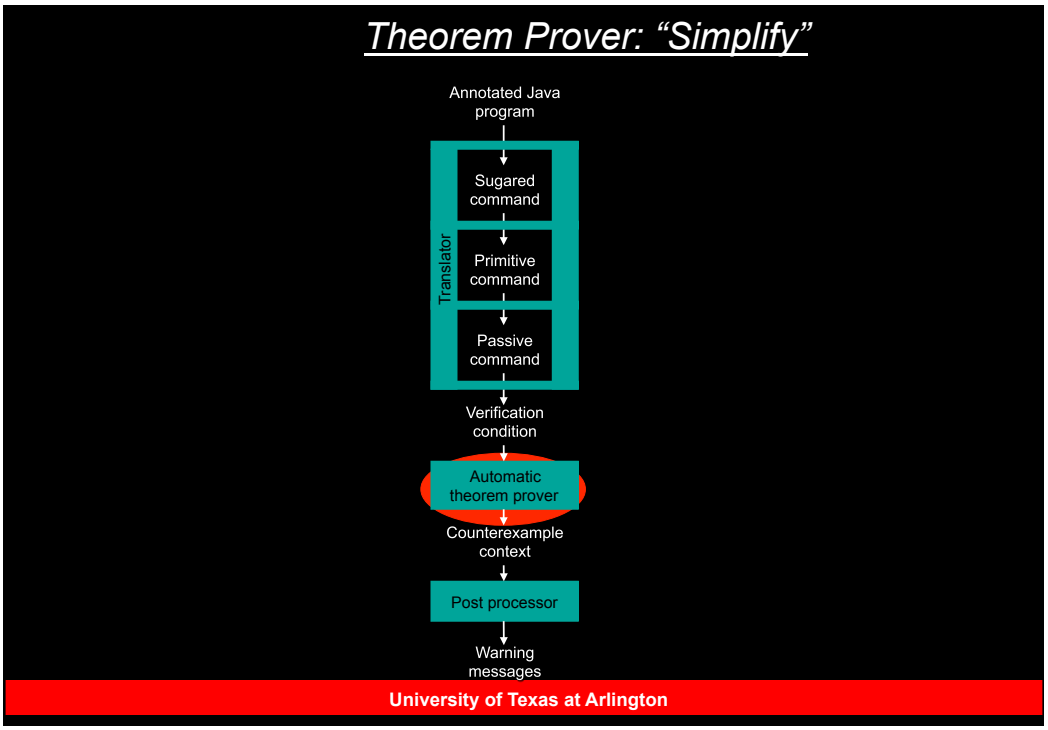
  (assume $x_0 < 0$; assume $x_1 = -x_0$;
  assume $x_2 = x_1$
  [] assume !($x_0 < 0$);
  assume $x_2 = x_0$
  );
  assert $x_2 >= 0$

Verification Condition

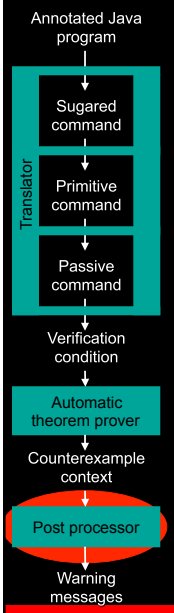


- Universal background predicate (UBP)
- Type-specific background predicate (TSBP)
- Verification Condition Generation
  Uses UBP & TSBP & previous stages


Verification Condition
Annotated Java program
Translator
Sugared command
Primitive command
Passive command
Verification condition
Automatic theorem prover
Counterexample context
Post processor
Warning messages
(AND
(<: T_T |T_java.lang.Object|)
(EQ T_T (asChild T_T |T_java.lang.Object|))
(DISTINCT arrayType |T_boolean| |T_char| |T_byte| |T_short| |T_int|
|T_long| |T_float| |T_double| |T_.TYPE|
T_T |T_java.lang.Object|)))
(EXPLIES
(LBLNEG |vc.T.abs.3.3|
(IMPLIES
(AND
(EQ |elems@pre| elems)
(EQ elems (asElems elems))
(< (eClosedTime elems) alloc)
(EQ LS (asLockSet LS))
(EQ |alloc@pre| alloc)
(NOT
(AND
(EQ |@true| (is |x:2.21| T_int))
(OR
(AND
(OR
(AND
(< |x:2.21| 0)
(EQ |x:3.17| (- 0 |x:2.21|))
class T {
static int abs(int x) {
if (x < 0) { x = -x; }
//@ assert x >= 0;
}
}
University of Texas at Arlington


Theorem Prover: “Simplify”
Annotated Java program
Translator
Sugared command
Primitive command
Passive command
Verification condition
Automatic theorem prover
Counterexample context
Post processor
Warning messages
University of Texas at Arlington

Counter examples and Warnings



Ÿ Counterexample:
labels: (|IndexTooBig@26.5| |vc.Bag.add.20.2| |trace.Then^0,21.23|)
context:
(AND
(NEQ |tmp1!a:23.23| null)
(NEQ this null)
(EQ |alloc@pre| alloc)
(EQ |tmp4!n:26.6| 0)
…
(<= alloc (vAllocTime |tmp3!a:26.4|))
)

Ÿ Bag: add(int) ...

Bag.java:26: Warning: Array index possibly too large (IndexTooBig)
a[n] = x;
^
Execution trace information:
Executed then branch in "Bag.java", line 21, col 23.

Some Errors that ESC / Java 2 discovers

- ✓ Pre Condition
- ✓ Post Condition
- ✓ Invariant
- ✓ Initially
- ✓ Index Negative
- ✓ Index Too Big
- ✓ Null
- ✓ ….
- ✓ …..

Some Runtime Errors Detected by ESC/Java 2

✓ Index Negative

Issued when an array index < 0

✓ Index Too Big

Issued when an array index >= Array Length

✓ Null

Issued when there is a possibility of ***NullPointerException***

Some Annotation Violations Detected by ESC/Java 2

✓Pre and Post

Issued in response to user-written preconditions (requires), post-conditions (ensures….)

✓Invariant

Invariant clause generate additional post-conditions for every method. If they do not hold, appropriate warnings are generated

✓ Initially

Initially clause is a post-condition for every constructor

Modular Reasoning

✓ ESC/Java2 reasons about every method individually

```
public class ModularReasoning {
 int[] b;
 ModularReasoning(){
 b = new int[20]; }
 public void m() {
 b[0] = 2;
 }}
```

Warns that b[0] may be a null dereference here, even though you can see that it won't be.

D E M O

(Stack Example)

Unsound and Incomplete (1 / 3)

- *Unsound*
  - ✓***Misses errors*** *that are actually present in the program*
- *Incomplete*
  - ✓*Warns of Potential Errors when it is impossible for these to occur*

ESC / Java2 not Sound and Complete

- ✓ *Affects Complexity of Annotation Language*
- ✓*Tradeoff to make it Cost effective*

Unsound and Incomplete (2 / 3)

Example 1

```
int[] array = new int[10];

for(int i = 0; i < 20; i++)

array[i] = i;
```

ArrayIndex out of Bound - Error occurs but will not be caught by Tool

Reason : Tool does not consider all Possible Iterations

Example 2

```
int i = 32000;

i= i * i;
```

Arithmetic Overflow - Error occurs but will not be caught by Tool

Reason: Assumes that (i) is of unlimited magnitude

Unsound and Incomplete (3 / 3)

```
12
13    public void n()
14    {
15        char c;
16        c = "ESC/Java2".charAt(2);
17
18       //@ assert c=='C';
19    }
```

Semantics for String Operations are weak.

ESC/Java 2 and Spec# Systems

	ESC/ Java2 Tool	Spec # Tool
Programming Language	**Java**	**C#**
Annotation Language	**JML**	**Spec #**
Automatic Theorem Prover	**Simplify**	**Z3**
Verifier	**ESC/Java2**	**Boogie**

Competing Technologies & Tools (1/2)

- FindBugs
  - ✓ *Finds Bugs in Java*
  - ✓ *Static Checker*
  - ✓ *Detects Synchronization Problems*
  - ✓ *Plug-ins for Eclipse, NetBeans*
- JLint
  - ✓*Static Checker*
  - ✓*C, C++ , Java*

Competing Technologies & Tools (2/2)

Bug Category	Examples	ESC/ Java2	FindBugs	JLint
General	Null dereference	✓*	✓*	✓*
Concurrency	Possible deadlock, race	✓*	✓	✓*
Exceptions	Possible unexpected exception	✓*		
Array	Length may be less than zero	✓		✓*
Mathematics	Division by zero	✓*		✓
Conditional, loop	Unreachable code		✓	
I/O stream	Stream not closed on all paths		✓*	
Unused or duplicate statement	Unused local variable		✓	

✓ Bug Category *** Example only**

Source : A Comparison of Bug Finding Tools for Java by Nick Rutar, Christian B. Almazan, Jeffrey S. Foster

Limitations & Future Challenges

- Limitations
  - *Iterates through Loops only once*
  - *Limitations on checking Arithmetic Overflow*
  - *Does not check for Non Functional Properties*
  - *Does not check Functional Properties not specified by User*
  - *Feasible only on Small Programs*
  - *Writing Annotations is labor Intensive*
- Future Challenges
  - *Reduce Annotation Burden*
    - ✓ *Perform Non-Modular Checking*
    - ✓ *Develop Annotation Assistants (Houdini is for ESC/Java2)*
  - *Teaching JML & ESC/Java2 with Programming Languages*

How ESC/Java2 is Useful

- ✓ *Possible run-time errors can be identified at compile time.*
- ✓ *Assumptions made by the programmer are made explicit.*
- ✓ *JML annotations provide documentation.*

Our Opinion on the Tool

- Likes
  - ✓ *Uses JML which is easy to understand*
  - ✓ *Integrated into Eclipse*
- Dislikes
  - ✓ *Counter example difficult to decode*
  - ✓ *Manuals for Installing & Configuring Tool is not comprehensive*

Things learnt from the Tool

- *Thinking in terms of Specifications while programming*
- *Improving Quality of Code*
- *Thinking from both perspectives*
  - ✓ *Client*
  - ✓ *Supplier*

Summary

- ✓ *Purpose of Extended Static Checking*
- ✓ *ESC/Java2 Tool Architecture*
- ✓ *Errors detected by ESC/Java2*
- ✓ *Features of ESC/Java2*

References (1 / 2)

1. http://secure.ucd.ie/products/opensource/ESCJava2/
2. http://en.wikipedia.org/wiki/History_of_programming_languages
3. Extended Static Checking for Java by Cormac Flanagan,K.Rustan M.Leino,Mark Lillibridge,Greg Nelson,James B.Saxe,Raymie Stata
4. ESC/Java2:Uniting ESC/Java and JML by David R.Cok,Joseph R.Kiniry
5. An overview of JML tools and applications by Lilian Burdy, Yoonsik Cheon, David R. Cok, Michael D. Ernst, Joseph R. Kiniry, Gary T. Leavens, K. Rustan M. Leino, Erik Poll
6. Extended Static Checking: a Ten-Year Perspective by K.Rustan M Leino
7. http://www.cs.ru.nl/~erikpoll/talks/jml_tutorial/3_warnings4up.pdf
8. Soundness and completeness warnings in ESC/Java2 by Joseph R.Kiniry, Alan E Morkan,Barry Denby
9. Improving the Quality of Web-based Enterprise Applications with Extended Static Checking:A Case Study by Fredric Rioux, Patrice Chalin
10. ESC/Java2 as a Tool to Ensure Security in the Source Code of Java Applications by Aleksy Schubert and Jacek Chrzaszcz

References (2 / 2)

11. A Comparison of Bug Finding Tools for Java by Nick Rutar, Christian B. Almazan, Jeffrey S. Foster

12. ESC/Java User's Manual by K. Rustan M. Leino, Greg Nelson, and James B. Saxe

13. ESC/Java2 Implementation Notes by David R. Cok,Joseph R. Kiniry,Dermot Cochran

14. http://santos.cis.ksu.edu/771/node/8