Supporting Multiple Theories and Provers in ESC and VC

Updates on Verified Java with regards to GC6 and the Verified Software Repository

Joseph Kiniry

Systems Research Group
School of Computer Science and Informatics
University College Dublin

VerifiCard to MOBIUS

- VerifiCard was a multi-year, multi-institutional EU project focused on the verification of JavaCard programs (–2002)
  - two products out of Nijmegen were the Loop compiler and ESC/Java2
- MOBIUS is a multi-year, multi-institutional EU project focused on delivering a full-blown interactive verification environment for multi-threaded Java programs (2005–8)

ESC/Java2

- ESC/Java2 is an *extended static checker*
  - based upon DEC/Compaq SRC ESC/Java
  - operates on JML-annotated Java code
  - behaves like a compiler
    - error messages similar to javac & gcc
    - completely automated
    - hides enormous complexity from user

Soundness and Completeness

- a sound and complete prover is non-automated, very complex, and expensive
  - modular checking
  - properties of arithmetic and floats
  - complex invariants and data structures
- instead, design and build an unsound and incomplete verification tool
  - trade soundness and completeness for automation and usability

The ESC/Java2 Object Logic

- partial semantics for Java and JML
- written in unsorted first-order logic
- highly tuned to current theorem prover's capabilities and quirks
  - Nelson's Simplify prover circa mid-80s
- originally consisted of 81 axioms
- extended by 20 axioms in ESC/Java2

ESC/Java Calculi

- used for verification condition generation in Dijkstra wp/wlp style
- easy for small/research languages
- much harder for “real world” languages
  - typed concurrent object-oriented language
  - dynamic memory allocation and GC
  - exceptions
  - aliasing

Problems with Current Logic

- unsorted
  - no mental model, no type checking
- tightly coupled to Simplify prover
  - unmaintained, two generations old
- very incomplete
  - want to verify new properties and some functional specifications
- never checked for soundness

The New ESC/Java2 Logics

- partial semantics for Java and JML
- written in *sorted* first-order logic
- independent of any particular prover
- written in SMT-LIB
  - supported by many new provers
- ~100 axioms thus far
  - no reduction because no subsorts or overloading

Benefits of New Logic

- ESC/Java2 will support multiple provers
  - use multiple provers concurrently
  - choose prover(s) based upon context
- proof of soundness
  - new logic being encoded in PVS and Coq
- increase ESC/Java2 soundness, completeness, and performance
  - able to verify larger, more complex programs than ever before

Current Work

- initial version of new logic sketched out
  - dramatically more understandable
- beginning to use new provers
  - *Sammy* from Tinelli, *haRVey* from Ranise
  - investigating other new provers from CAV
- incorporate new provers into ESC/Java2
  - increase independence of tool from Simplify and from specific architectures

What's Next

- "factoring out" calculus and logic from implementation of ESC/Java2
- we plan to attempt to prove that the new logic is sound and subsumes the old logic
- soundness and completeness warnings
- integrate with full verification Loop
- proof reuse when moving from first-order to higher-order semantics

Verified Java Software

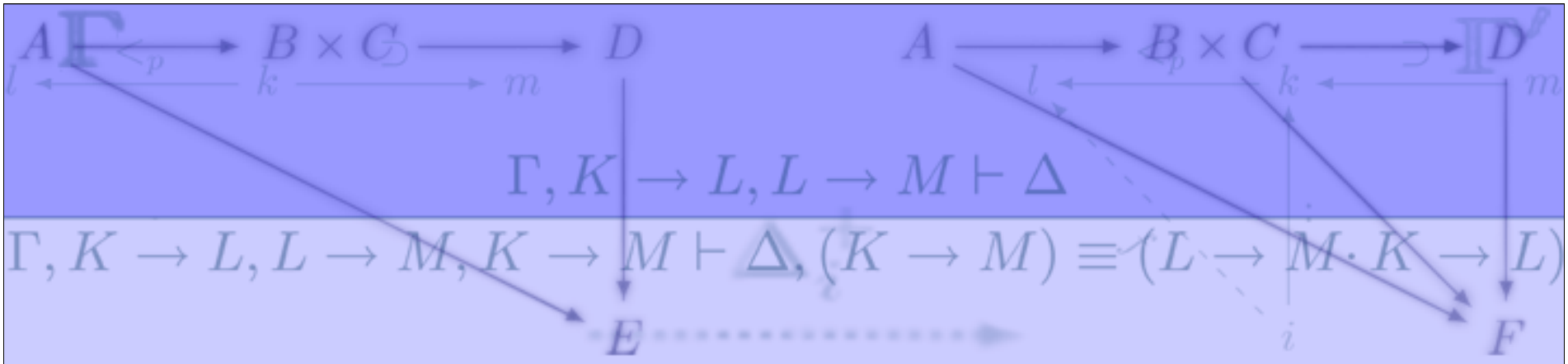
- half a dozen JavaCard programs from industry and academic case studies
- KOA Internet voting tally system
- research tools developed by RUN and UCD
- ESC/Java2 itself
- the new MOBIUS IVE

Overarching Goal: The MOBIUS IVE

- an integrated verification environment built on top of the Eclipse platform
- integrates best-of-breed FM plugins from a variety of teams (e.g., unit testing, model checking, program slicing, etc.)
- full integration of best practices from global, group-centric software engineering in FLOSS community with PCC-based program verification

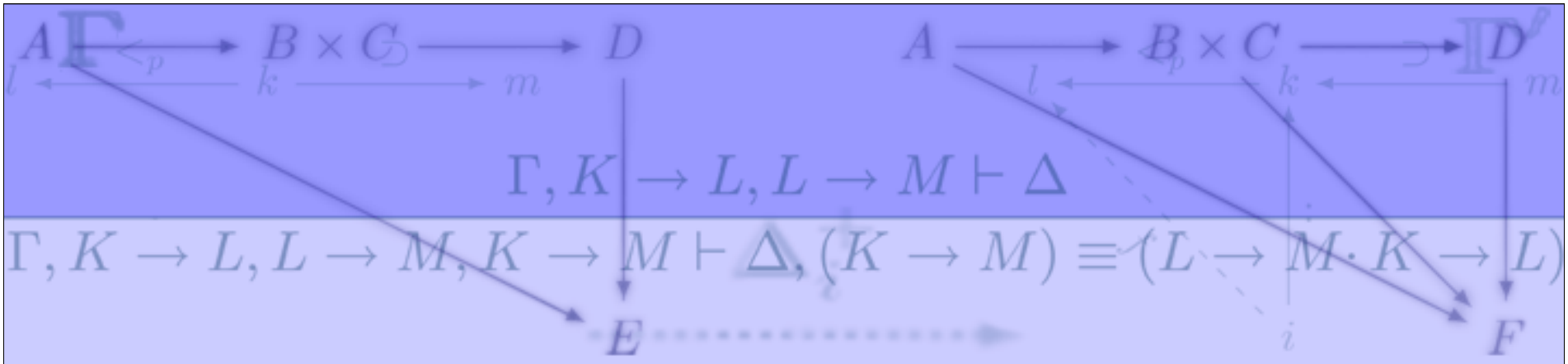
Acknowledgements

- original DEC/Compaq SRC team
  - K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, Jim Saxe, Raymie Stata, Cormac Flanagan, et al
- ESC/Java2 collaborators
  - David Cok (Kodak R&D), Clément Hurlin (Nancy), Cesare Tinelli (Univ. of Iowa), and Aleksey Schubert (Univ. of Warsaw)
- JML community
  - led by Gary Leavens and major participants Yoonsik Cheon, Curtis Clifton, Todd Millstein, and Patrice Chalin
- verification community
  - Peter Müller, Marieke Huisman, Joachim van den Berg
- SoS Group at Radboud University Nijmegen
  - Erik Poll, Bart Jacobs, Cees-Bart Breunesse, Martijn Oostdijk, Martijn Warnier, Wolter Pieters, Ichiro Hasuo



Questions?

Systems Research Group
School of Computer Science and Informatics
University College Dublin



Slides for Questions

Systems Research Group
School of Computer Science and Informatics
University College Dublin

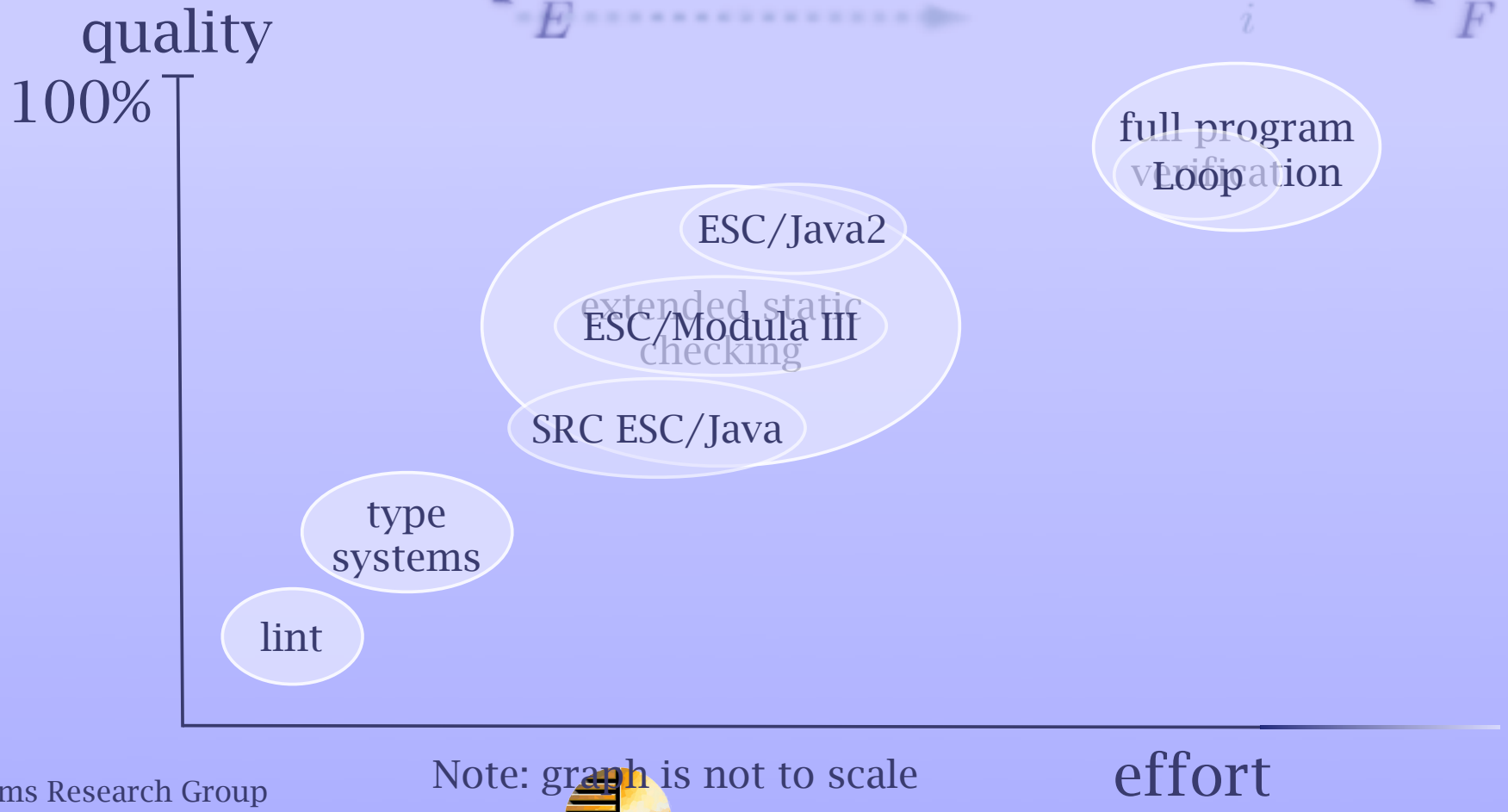
What is Extended Static Checking?

- type systems
  - Error: wrong number of arguments to call
- lint & modern compilers
  - Error: unreachable code
- full program verification
  - Error: qsort does not yield a sorted array

Why Not Just Test?

- testing is essential, but
  - expensive to write and *maintain*
  - finds errors *late*
  - *misses* many errors
- static checking and testing are *complementary techniques*

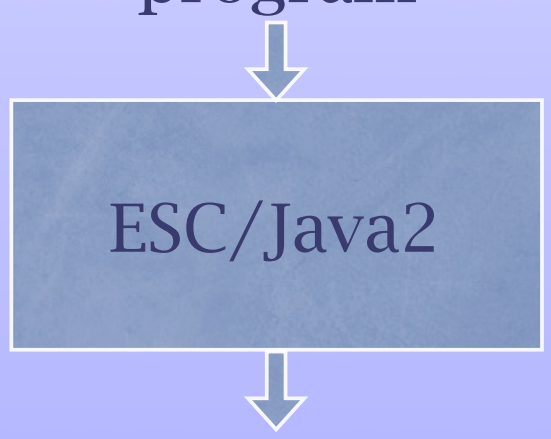
Comparison of Static Checkers



Note: graph is not to scale

Systems Research Group
School of Computer Science and Informatics
University College Dublin

ESC/Java2 Use



- Modularly checks for:
  - null-dereference errors
  - array bounds errors
  - type cast errors
  - specification violations
  - race conditions & deadlocks
  - ... dozens of other errors

JML: The Java Modeling Language

- a behavioral interface specification language
- syntax and semantics are very close to Java
- annotations written as comments in code
- JML is a very rich language
  - standard constructs include preconditions, postconditions, invariants, etc.
- one language used for documentation, runtime checking, and formal verification

A JML Example and ESC/Java2 Demo

```
class Bag {
  int[] a;
  int n;

  Bag(int[] input) {
    n = input.length;
    a = new int[n];
    System.arraycopy(input, 0,
                     a, 0, n);
  }

  boolean isEmpty() {
    return n == 0;
  }
```

```
  int extractMin() {
    int m = Integer.MAX_VALUE;
    int mindex = 0;
    for (int i = 1; i <= n; i++) {
      if (a[i] < m) {
        mindex = i;
        m = a[i];
      }
    }
    n--;
    a[mindex] = a[n];
    return m;
  }
}
```


The Annotated Class

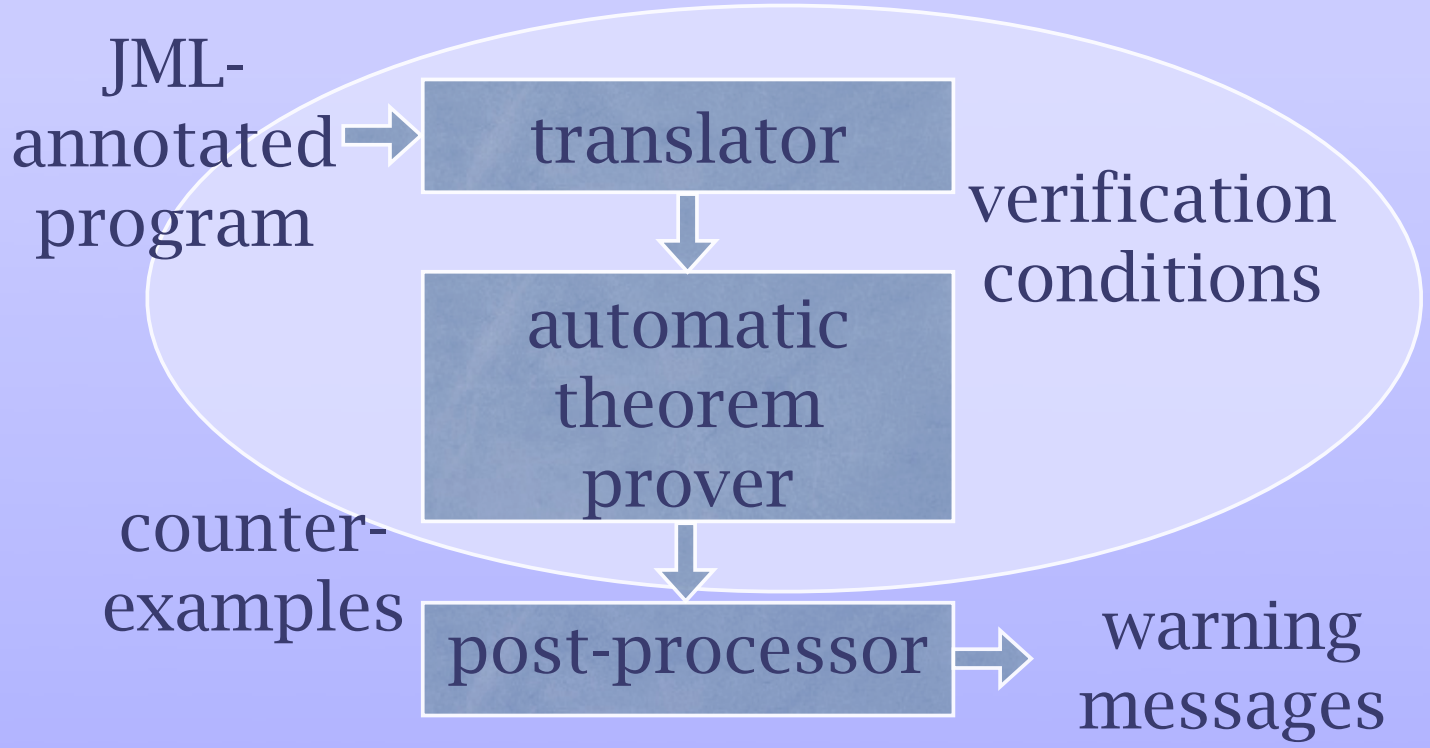
```
class Bag {
  /*@ non_null */ int[] a;
  int n;
  //@ invariant 0 <= n && n <= a.length;
  //@ ghost public boolean empty;
  //@ invariant empty == (n == 0);

  //@ requires input != null;
  //@ ensures this.empty == (input.length == 0);
  public Bag(int[] input) {
    n = input.length;
    a = new int[n];
    System.arraycopy(input, 0, a, 0, n);
    //@ set empty = n == 0;
  }

  //@ ensures \result == empty;
  public boolean isEmpty() {
    return n == 0;
  }

  //@ requires !empty;
  //@ modifies empty;
  //@ modifies n, a[*];
  public int extractMin() {
    int m = Integer.MAX_VALUE;
    int mindex = 0;
    for (int i = 0; i < n; i++) {
      if (a[i] < m) {
        mindex = i;
        m = a[i];
      }
    }
    n--;
    //@ set empty = n == 0;
    //@ assert empty == (n == 0);
    a[mindex] = a[n];
    return m;
  }
}
```

ESC/Java2 Architecture



Example Java Type Axioms

```
(DEFPRED (<: t0 t1))

(BG_PUSH (<: |T_java.lang.Object|
             |T_java.lang.Object|))

; <: reflexive
(BG_PUSH
  (FORALL (t)
    (<: t t)))


; <: transitive
(BG_PUSH
  (FORALL (t0 t1 t2)
    (IMPLIES (AND (<: t0 t1) (<: t1 t2))
      (<: t0 t2))))
```

```
;anti-symmetry
(BG_PUSH
 (FORALL
  (t0 t1)
  (IMPLIES (AND (<: t0 t1) (<: t1 t0))
           (EQ t0 t1))))


; primitive types are final
(BG_PUSH (FORALL (t)
        (IMPLIES (<: t |T_boolean|)
                 (EQ t |
T_boolean|))))
```


Examples Showing Java Incompleteness

```
(BG_PUSH (FORALL (x)
           (IFF (is x |T_char|) (AND (<= 0 x) (<= x 65535)))))
(BG_PUSH (FORALL (x)
           (IFF (is x |T_byte|) (AND (<= -128 x) (<= x 127)))))
(BG_PUSH (FORALL (x)
           (IFF (is x |T_short|) (AND (<= -32768 x) (<= x 32767)))))
(BG_PUSH (FORALL (x)
           (IFF (is x |T_int|) (AND (<= intFirst x) (<= x intLast)))))
(BG_PUSH (FORALL (x)
           (IFF (is x |T_long|) (AND (<= longFirst x) (<= x longLast)))))

(BG_PUSH (< longFirst intFirst))
(BG_PUSH (< intFirst -1000000))
(BG_PUSH (< 1000000 intLast))
(BG_PUSH (< intLast longLast))
```


Examples of Java & JML Semantics

```
(DEFPRED (is x t))

(BG_PUSH (FORALL (x t)
                 (is (cast x t) t)))

(BG_PUSH (FORALL (x t)
                 (IMPLIES (is x t) (EQ (cast x t) x))))

(BG_PUSH
 (FORALL (e a i)
         (is (select (select (asElems e) a) i)
             (elemtype (typeof a)))))

(DEFPRED (nonnullelements x e)
   (AND (NEQ x null)
        (FORALL (i)
                (IMPLIES (AND (<= 0 i) (< i (arrayLength x)))
                         (NEQ (select (select e x) i) null)))))
```

VC Generation for Java

annotated source

guarded commands

verification condition

```
x = a[ i++ ];

assume preconditions
assume invariants
...
i0 = i;
i = i + 1;
assert (LABEL null@218: a != null);
assert (LABEL IndexNeg@218: 0 <= i0);
assert (LABEL IndexTooBig@218: i0 < a.length);
x = elems[a][i0];
...
assert postconditions
assert invariants
```

$$\forall i_0.(i_0 = i \implies \ldots)$$

Verification Condition

- formula in unsorted, first-order predicate calculus
  - equality and function symbols
  - quantifiers
  - arithmetic operators
  - select and store operations
  - e.g., $\forall x.\forall y.\exists z.(x > y \implies z \times 2 == \ldots$

Example Verification Condition

- verification condition large & unstructured

```
(EXPLIES (LBLNEG |vc.Bag.isEmpty.11.2| (IMPLIES (AND (EQ |n@pre:3.6| |n:
3.6|) (EQ |n:3.6| (asField |n:3.6| T_int)) (EQ |a@pre:2.8| |a:2.8|) (EQ |a:
2.8| (asField |a:2.8| (array T_int))) (< (fClosedTime |a:2.8|) alloc) (EQ |
MAX_VALUE@pre:10..| |MAX_VALUE:10..|) (EQ |@true| (is |MAX_VALUE:10..|
T_int)) (EQ |length@pre:unknown| |length:unknown|) (EQ |length:unknown|
(asField |length:unknown| T_int)) (EQ |elems@pre| elems) (EQ elems (asElems
elems)) (< (eClosedTime elems) alloc) (EQ LS (asLockSet LS)) (EQ |
alloc@pre| alloc) (EQ |state@pre| state)) (NOT (AND (EQ |@true| (is this
T_Bag)) (EQ |@true| (isAllocated this alloc)) (NEQ this null) (EQ RES
(integralEQ (select |n:3.6| this) 0)) (LBLPOS |trace.Return^0,12.4| (EQ |
@true| |@true|)) (NOT (LBLNEG |Exception@13.2| (EQ |ecReturn| |
ecReturn|)))))))) (AND (DISTINCT |ecReturn|) (< 1000000 pos2147483647)))
```


Sorts of New Logic

```
:sorts ( # sort that represents *values* of Java's boolean base type
         Boolean
         # sort that represents *values* of all Java's base types
         # but for Boolean
         Number
         # sort that represents all Java non-base types
         ReferenceType
         # ... represents object references
         Reference
         # ... represents object values
         Object
         # Boolean, Number, Object fields
         BooleanField
         NumberField
         ReferenceField
         # ... represents the heap
         Memory )
```


Example Axioms

```
# <: is anti-symmetric
(forall ?x ReferenceType
   (forall ?y ReferenceType
     (implies (and (<: ?x ?y) (<: ?y ?x))
              (= (?x ?y)))))

# java.lang.Object is top of subtype hierarchy
(forall ?x ReferenceType (<: ?x java.lang.Object))

# subtype rules for arrays
(forall ?x ReferenceType
   (forall ?y ReferenceType
     (implies (<: ?x ?y)
              (<: (array ?x) (array ?y)))))
```