Software Model Checking Using Bogor

- a Modular and Extensible Model Checking Framework

3rd Estonian Summer School in Computer and System Science (ESSCaSS'04)

Slide Set 07: Checking JML Specifications

http://bogor.projects.cis.ksu.edu

http://www.cis.ksu.edu/~hatcliff/ESSCaSS04

John Hatcliff Matthew B. Dwyer Robby

SAnToS Laboratory, Kansas State University, USA

Support

US Army Research Office (ARO)
US National Science Foundation (NSF)
US Department of Defense
Advanced Research Projects Agency (DARPA)

Boeing
Honeywell Technology Center
IBM
Intel

Lockheed Martin
NASA Langley
Rockwell-Collins ATC
Sun Microsystems

Motivation and Acknowledgements

- All other model-checkers that we know of support only simple predicates on system states (e.g., the primitive propositions occurring in temporal logic formulas).
- Especially when modeling OO languages, states themselves can be quite complicated (they include the heap).
- Therefore we are interested in supporting specification predicates over Bogor states that are significantly stronger than those supported in other model-checking frameworks.
- Moreover, we are interested in supporting, as much as possible, rich specification languages that other verification tools using different technologies (e.g., theorem proving) also support.
- These slides are taken from our talk given at TACAS 2004 on "Checking Strong Specifications Using an Extensible Model-Checking Framework"
- A significant portion of this work was carried out by Edwin Rodriguez

Assertions for Software Verification

- Assertions have become a common practice among developers
  - 10 years ago assertions were not considered useful by developers
  - recent evidence of the effectiveness of assertions
    - David Rosenblum (1995)
  - now some programming languages have included assertions in their standard specifications
    - c.f. Java 1.4 assertions

Concurrent Queue based on Linked List (Doug Lea's util.concurrent package)

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  public LinkedNode(Object x) {
    v
  }
}

publi
  pro
  pro
  pro
  pro

  pub
    p
    h
  }

  pub
    s

    }
  }

  pub
    if (x == null)
      throw new IllegalArgumentException();
    insert(x);
  }
```

```
protected synchronized Object extract() {
  synchronized (head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {

otify();

    if (x != null) return x;
    else …
  }
```

```
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) {
      last.next = p;
      last = p;
    }
    if (waitingForTake > 0) putLock.notify();
      return;
  }
}
```

… allows concurrent access to put() and take()

An example

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  public LinkedNode(Object x) {
    v
  }
}

publi
  pro
  pro
  pro
  pro

  pub
    pu
    he
  }

  public boolean isEmpty() {
    synchronized (head) {
      return head.next == null;
    }
  }

  public void put(Object x) {
    if (x == null)
      throw new IllegalArgumentException();
    insert(x);
  }
```

```
protected synchronized Object extract() {
  synchronized (head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {

    }
    if (waitingForTake > 0) putLock.notify();
      return;
  }
}

public Object take() {
  Object x = extract();
  if (x != null) return x;
  else …
}
```

```
public class LinkedQueue {
  protected final Object putLock;
  p
  p
  protected int waitingForTake = 0;
  .
  .
  .
```

Specify that putLock is never null

An example

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  public LinkedNode(Object x) {
    value = x;
  }
}

public class LinkedQueue {
  protected final Object putLock;
  protected LinkedNode head;
  protected Li
  protected in

  public Linke
    assert(putLock != null);
    putLock = new Object();
    head = new LinkedNode(null);
    assert(putLock != null);
  }

  public boolean isEmpty() {
    assert(putLock != null);
    synchronized (head) {
      return head.next == null;
    }
    assert(putLock != null);
  }

  public void put(Object x) {
    assert(putLock != null);
    if (x == null)
      throw new IllegalArgumentException();
    insert(x);
    assert(putLock != null);
  }
```

```
protected synchronized Object extract() {
  assert(putLock != null);
  synchronized (head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {
      x = first.value;
      first.value = null;
      head = first;
    }
    return x;
  }
  assert(putLock != null);

  assert(putLock != null);
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) {
      last.next = p;
      last = p;
    }
    if (waitingForTake > 0) putLock.notify();
      return;
  }
  assert(putLock != null);
}

public Object take() {
  assert(putLock != null);
  Object x = extract();
  if (x != null) return x;
  else …
  assert(putLock != null);
}
```

Need more declarative formalisms

Specify that putLock is never null

Specification Languages

- We want specification languages that
  - have a rich set of primitives for observing program state
    - heap-allocated objects, concurrency, *etc.*
  - make it easy to write useful specifications
    - support lightweight and deep-semantic specifications
  - be checkable using a variety of analysis techniques
    - static analysis, theorem proving, *etc.*

Java Modeling Language (JML)

- Developed by G. Leavens and other colleagues at Iowa State University
  - very rich set of operators, especially for describing complex heap properties
    - **`\reach(r), \forall(), \old(), etc.`**
  - support for specifications with varying degrees of complexity
    - lightweight vs. heavyweight specifications
  - has been checked with a variety of different techniques
    - so far, static analysis, theorem proving and runtime checking
- Emerging as a standard specification language for Java within the research community

Java Modeling Language (JML)

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  public LinkedNode(Object x) {
    v
  }
}

publi
  pro
  pro
  pro
  pro

  pub
    p
    h
  }

  pub
    s

    }
  }

  pub
    if (x == n
      throw ne
    insert(x);
  }
```

```
protected synchronized Object extract() {
  synchronized (head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {
```

```
protected void insert(Object x) {
  assert(x != null);
  synchronized (putLock) {

  }
}
```

```
//@ requires x != null;
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) {
      last.next = p;
      last = p;
    }
    if (waitingForTake > 0) putLock.notify();
      return;
  }
}
```

An example

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  public LinkedNode(Object x) {
    value = x;
  }
}
```

```
    head = new LinkedNode(null);
    assert(putLock != null);
  }

  public boolean isEmpty() {
    assert(putLock != null);
    synchronized (head) {
      return head.next == null;
    }
    assert(putLock != null);
  }

  public void put(Object x) {
    assert(putLock != null);
    if (x == null)
      throw new IllegalArgumentException();
    insert(x);
    assert(putLock != null);
  }
```

```
protected synchronized Object extract() {
  assert(putLock != null);
  synchronized (head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {
      x = first.value;
```

```
    LinkedNode p = new LinkedNode(x);
    synchronized (last) {
      last.next = p;
      last = p;
    }
    if (waitingForTake > 0) putLock.notify();
      return;
  }
  assert(putLock != null);
}

public Object take() {
  assert(putLock != null);
  Object x = extract();
  if (x != null) return x;
  else …
  assert(putLock != null);
}
```

```
public class LinkedQueue {
  protected final /*@ non_null @*/ Object putLock;
    •
    •
    •
```

Java Modeling Language (JML)

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  /*@ behavior
```

```
/*@ behavior
  @   requires x != null;
  @   ensures true;
  @ also behavior
  @   requires x == null;
  @   signals (Exception e) e instanceof IllegalArgumentException;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                                \old(\reach(head)).has(n);
  @                                 n.value == \result
  @                                 && !(\reach(head).has(n)));
  @*/
protected synchronized Object extract() {
  synchronized(head) {
    Object x = null;
    LinkedNode first = head.next;
    if (first != null) {
      x = first.value;
      first.value = null;
      head = first;
    }
    return x;
  }
}
```

```
protected void refactoredInsert(LinkedNode n) {
  last.next = n;
  last = n;
}
```

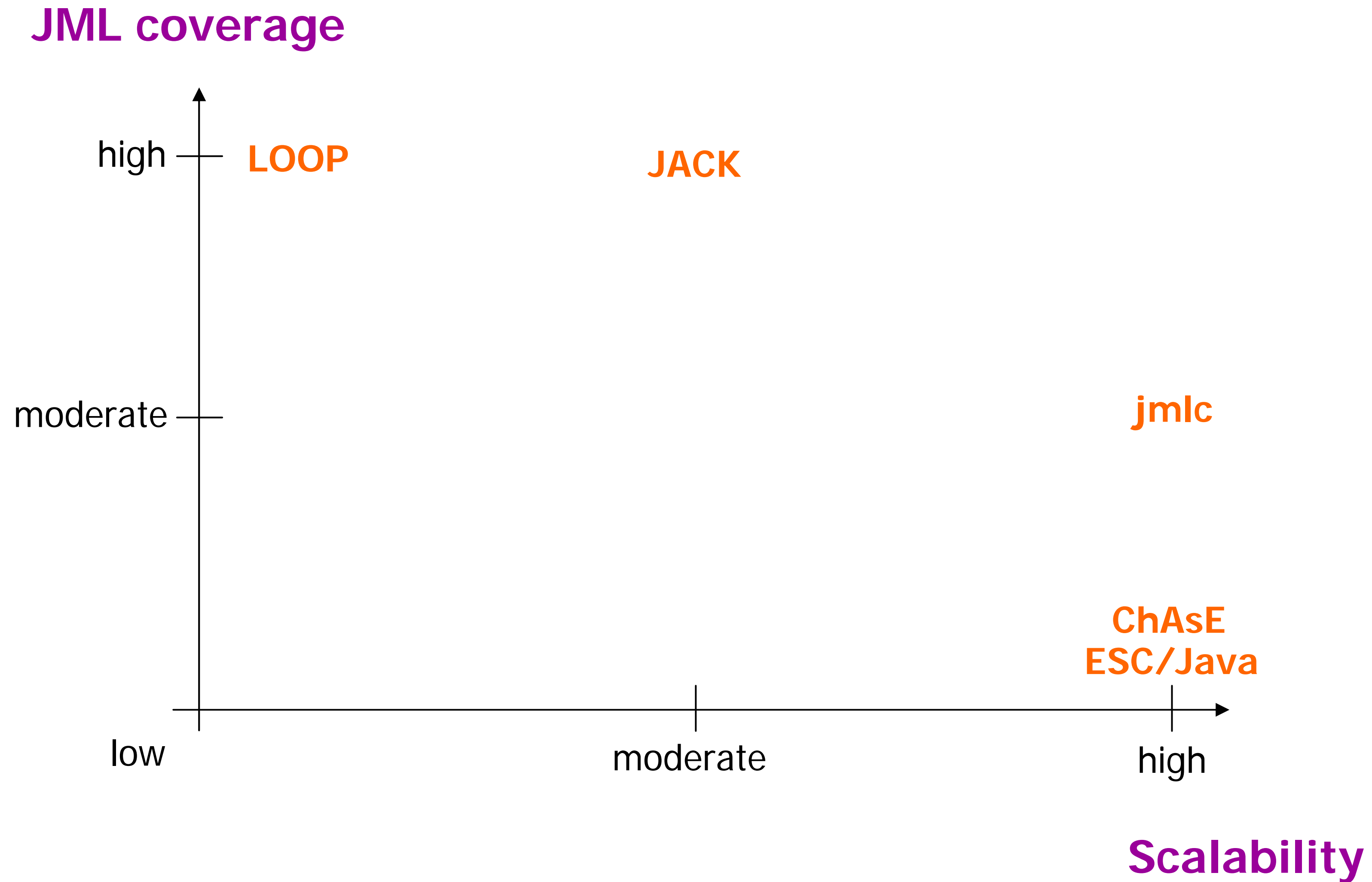
```
return;
```

... ability to make deep-semantic specifications

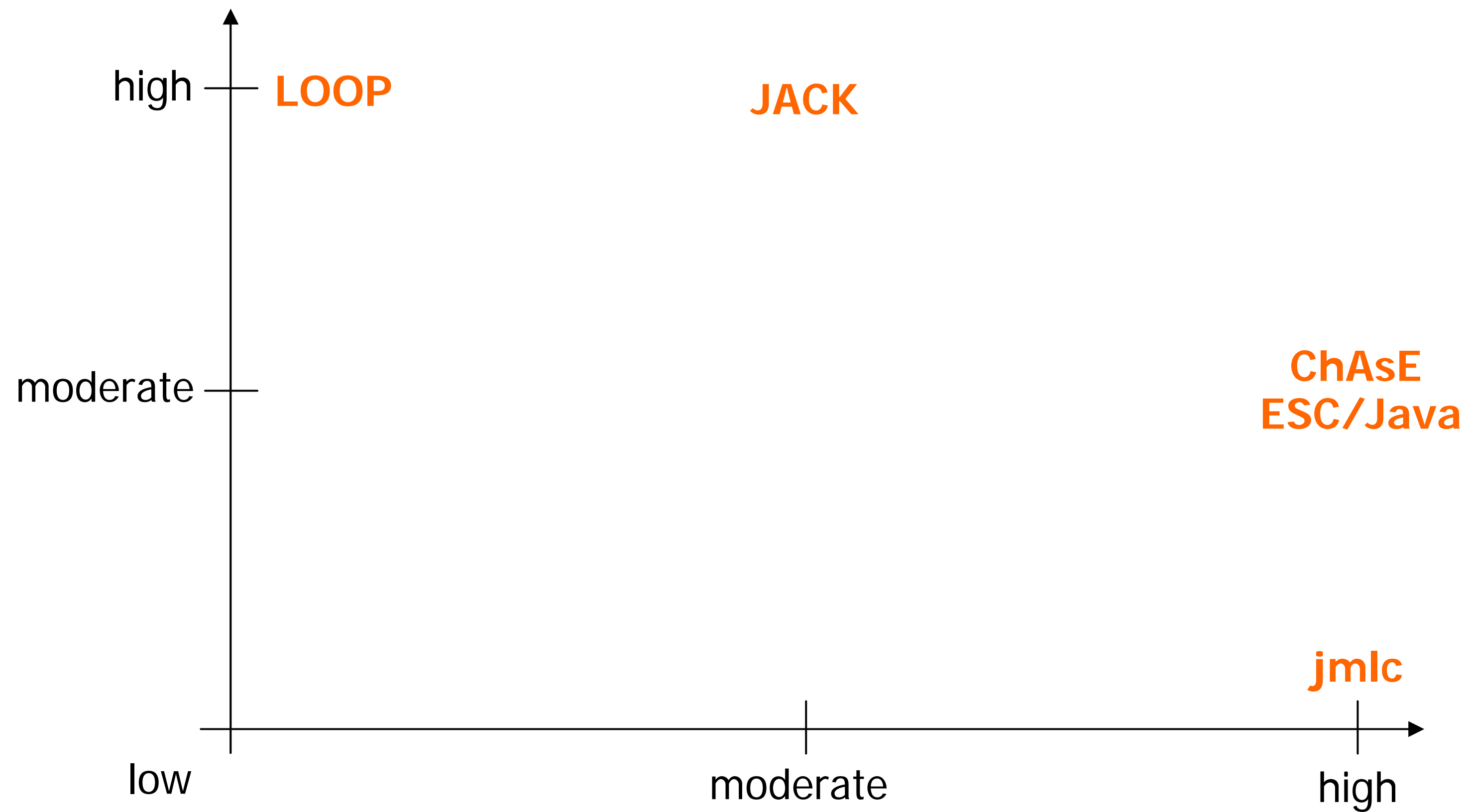
Tool support for JML

- Many tools have been developed to support verification of JML
  - jmlc (Leavens et al)
  - LOOP (Jacobs et al)
  - ESC/Java (Compaq SRC)
  - KeY (Ahrendt et al)
  - Calvin (Flanagan et al)
  - JACK (Burdy et al)
  - ChAsE (N. Cataño)
  - Krakatoa (Marché et al)
  - Jive (Poetzsch-Heffter et al)
- Every tool provide different trade offs in terms of several factors
  - coverage of the JML language
  - coverage of Java
  - degree of automation
  - scalability

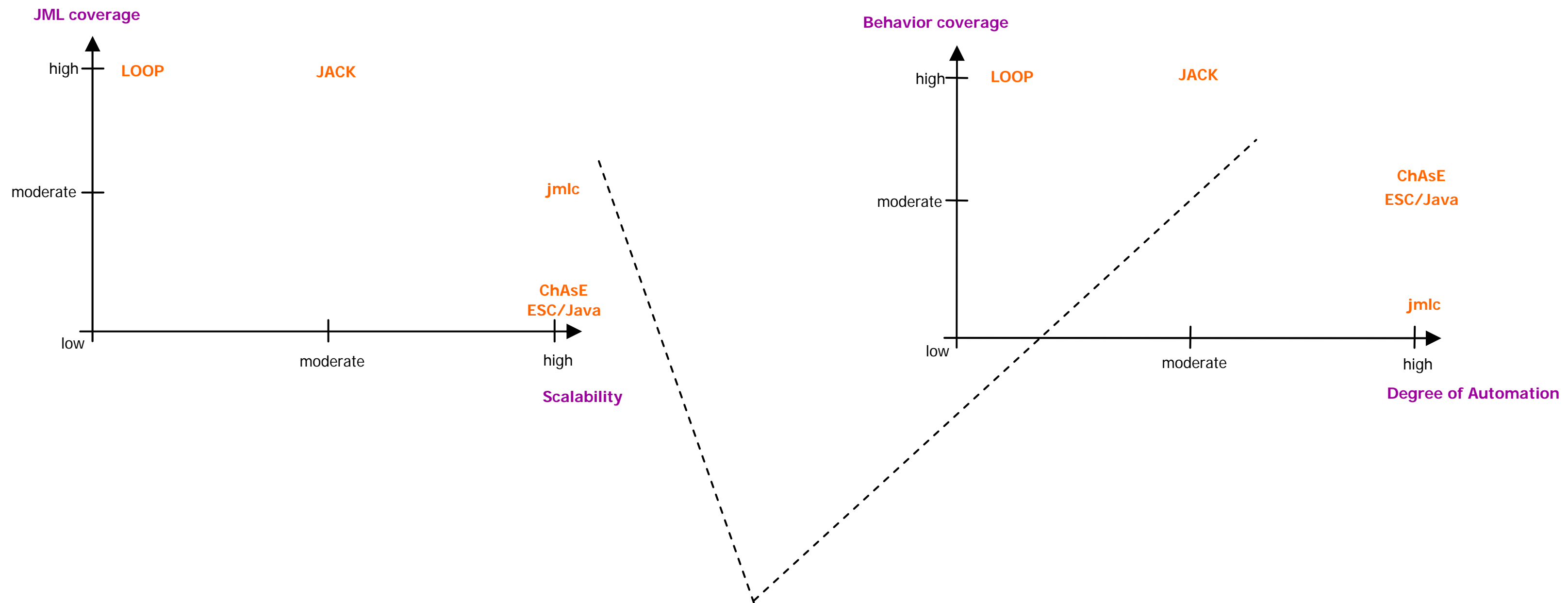
Assessment of JML Tools



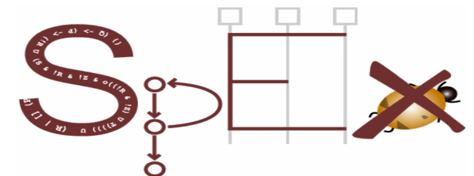
Assessment of JML Tools



Assessment of JML Tools



Motivation: explore model checking as a technique to fill these gaps



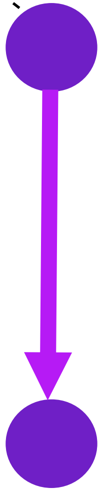
Bogor

Questions…

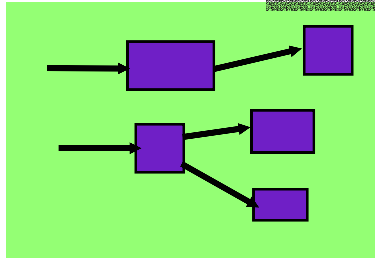
- What is it?
- Why is it useful?
- What makes it particularly good for checking JML?

Bogor's Heap Representation

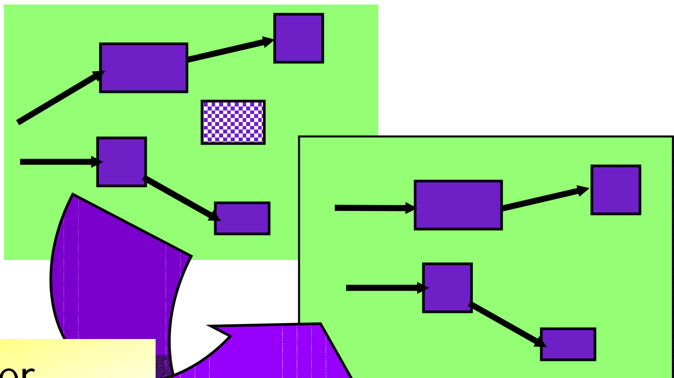
State



Heap



...transition may create new objects, garbage, etc.



...sort walks over heap, canonicalizes, and collects info

Canonical heap

Key Points...

...explicit heap representation

...after each transition, a topological sort gives heap objects a canonical order

...garbage is eliminated

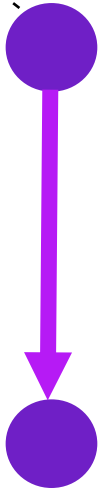
...precise heap model

...precise alias information

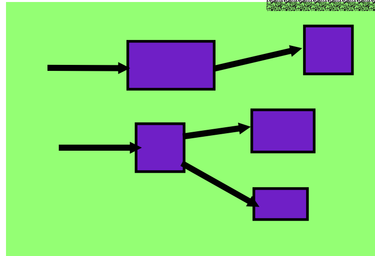
...have access to all visited states (but, efficiently stored using collapse compression)

Bogor’s Heap Representation – Enables JML Specs Check

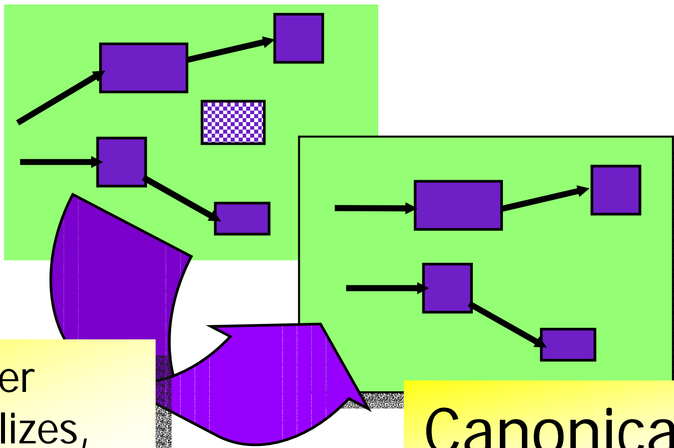
State



Heap



...transition may create new objects, garbage, etc.



...sort walks over heap, canonicalizes, and collects info

Canonical heap

Key Points...

... many JML features are easy to support in Bogor

...precise heap model (c.f., \reach)

...precise alias information (c.f., assignable)

...can easily compare objects in methods pre/post-states (c.f., \old)

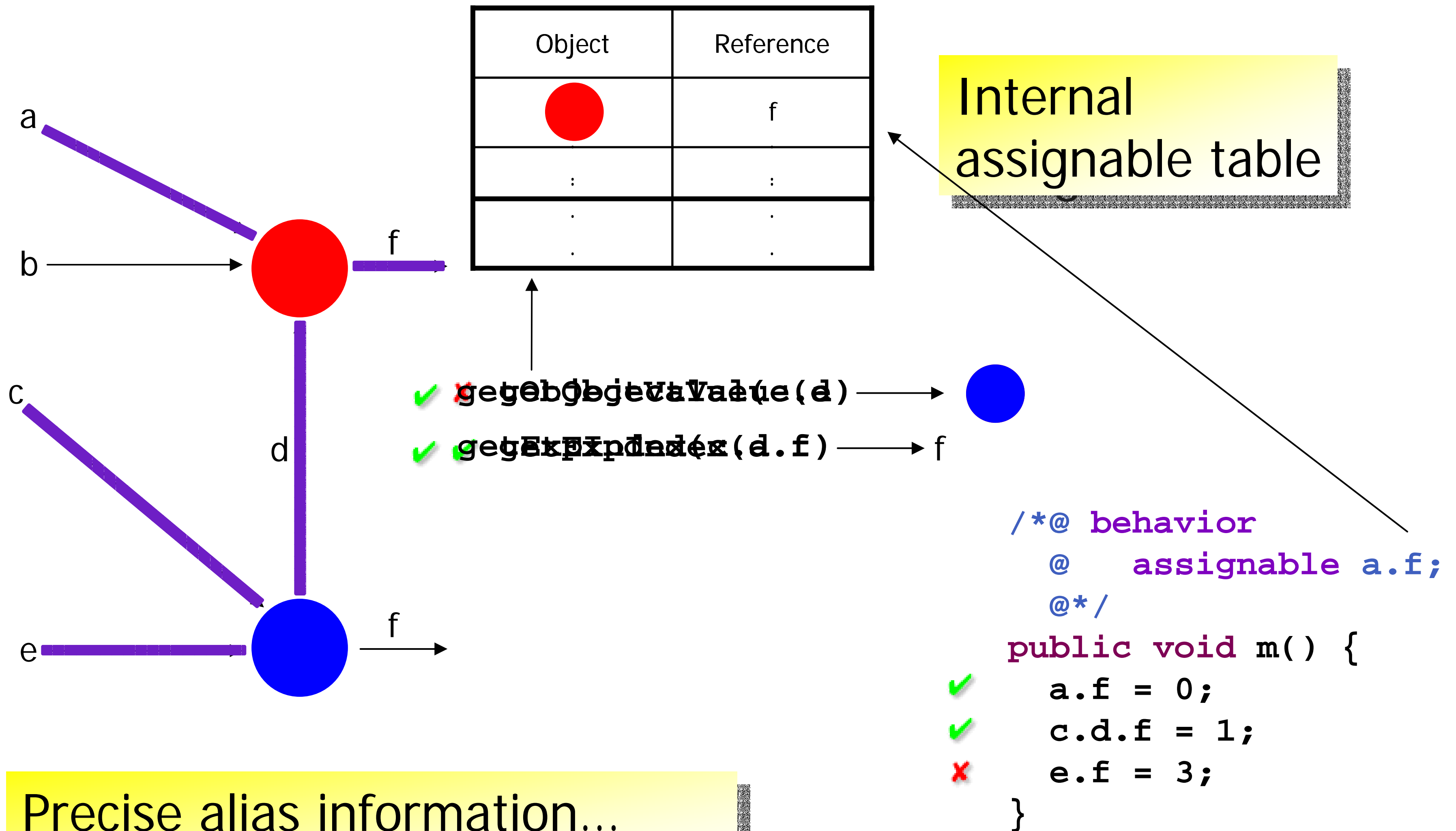
Checking JML Specs with Bogor

- Object operations
  - **`assignable`,`\reach(x)`,`\lockset`, `\fresh(`x_1`,…,`x_n`)`**
  - Quantification over objects of a specified type
    - **`\forall(), \exists()`**
- Pre-/post-conditions, invariants
- Referencing Pre-states
- Methods in JML expressions (the purity issue)

Object operations - assignable

- **`assignable`** allows to specify frame conditions for a method
  - **`assignable`** $v_1, v_2, \ldots, v_n$;
    - $v_1, v_2, \ldots, v_n$ can be modified by the method
    - modification to any other memory location is forbidden
- Traditionally hard to check
  - aliasing makes it hard to determine unambiguously which memory locations can actually be assigned to
  - verified conservatively in the best cases

Object operations - assignable



Object operations

- Maintaining a precise, dynamic heap model allows performing accurate object operations
  - Eliminated aliasing issues when checking **`assignable`**
  - Other object operations easily performed too
    - **`\reach(x)`** – simple DFS traversal from **`x`**
    - **`\forall`** – compute quantification set by DFS from root objects, then post-filtering by type

Checking JML Specs with Bogor

- Object operations
  - **`assignable`,`\reach(x)`,`\lockset`, `\fresh(`$x_1, \ldots, x_n$`)`**
  - Quantification over objects of a specified type
- Pre-/post-conditions, invariants
- Referencing Pre-states
- Methods in JML expressions (the purity issue)

LinkedQueue Example (JML)

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  /*@ behavior
    @   ensures value == x;
    @*/
  public LinkedNode(Object x) {
```

```
/*@ behavior
  @   requires x != null;
  @   ensures true;
  @ also behavior
  @   requires x == null;
  @   signals (Exception e) e instanceof IllegalArgumentException;
  @*/
 public void put(Object x) {
   if (x == null)
```

```
public class LinkedQueue {
  protected final /*@ non_null @*/ Object putLock;
  protected /*@ non_null @*/ LinkedNode head;
  protected /*@ non_null @*/ LinkedNode last = head;
  protected int waitingForTake = 0;

  //@ instance invariant waitingForTake >= 0;
  //@ instance invariant \reach(head).has(last);

 ...
```

```
    return head.next == null;
  }
}

/*@ behavior
  @ requires n != null;
  @ assignable last, last.next;
  @*/
protected void refactoredInsert(LinkedNode n) {
  last.next = n;
  last = n;
}
```

```
  @   requires x != null;
  @   ensures last.value == x && \fresh(last);
  @*/
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) refactoredInsert(p);
    if (waitingForTake > 0) putLock.notify();
      return;
    }
  }
}
```

LinkedQueue Example (JML)

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  /*@ behavior
    @   ensures value == x;
```

```
/*@ behavior
  @   requires x != null;
  @   ensures true;
  @ also behavior
  @   requires x == null;
  @   signals (Exception e) e instanceof IllegalArgumentException;
  @*/
```

```
/*@ behavior
    @   requires x != null;
    @   ensures last.value == x && \fresh(last);
    @*/
  protected void insert(Object x) {
    synchronized (putLock) {
      LinkedNode p = new LinkedNode(x);
      synchronized (last) refactoredInsert(p);
      if (waitingForTake > 0) putLock.notify();
        return;
      }
    }
  }
```

```
  /*@ behavior
    @  requires n != null;
    @  assignable last, last.next;
    @*/
  protected void refactoredInsert(LinkedNode n) {
    last.next = n;
    last = n;
  }
```



```
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) refactoredInsert(p);
    if (waitingForTake > 0) putLock.notify();
      return;
    }
  }
}
```

Pre/Post-Conditions

```
T m(T1 x1, ..., Tn xn) {
  checkInv$S();
  checkPre$m$S(x1, ..., xn);
  T rac$result;
  try {
    rac$result = orig$m(x1, ..., xn);
    checkPost$m$S(x1, ..., xn, rac$result);
    return rac$result;
  }
  catch (JMLEntryPreconditionError rac$e) {
    throw new JMLInternalPreconditionError(rac$e);
  }
  catch (JMLAssertionError rac$e) {
    throw rac$e;
  }
  catch (Throwable rac$e) {
    checkXPost$m$S(x1, ..., xn, rac$e);
  }
  finally {
    if (/* no postcondition violation? */) {
       checkInv$S();
       checkHC$S();
    }
  }
}
```

`jmlc` generates a wrapper method for each method in the class

Figure 4.3, “A Runtime Assertion Checker for the Java Modeling Language”, Y. Cheon

Pre/Post-Conditions

```
T m(T1 x1, ..., Tn xn) {
  checkInv$S();
  checkPre$m$S(x1, ..., xn);
  T rac$result;
  try {
    rac$result = orig$m(x1, ..., xn);
    checkPost$m$S(x1, ..., xn, rac$result);
    return rac$result;
  }
  catch (JMLEntryPreconditionError rac$e) {
    throw new JMLInternalPreconditionError(rac$e);
  }
  catch (JMLAssertionError rac$e) {
    throw rac$e;
  }
  catch (Throwable rac$e) {
    checkXPost$m$S(x1, ..., xn, rac$e);
  }
  finally {
    if (/* no postcondition violation? */) {
       checkInv$S();
       checkHC$S();
    }
  }
}
```

check invariants and method preconditions

Figure 4.3, "A Runtime Assertion Checker for the Java Modeling Language", Y. Cheon

Pre/Post-Conditions

```
T m(T1 x1, ..., Tn xn) {
  checkInv$S();
  checkPre$m$S(x1, ..., xn);
  T rac$result;
  try {
    rac$result = orig$m(x1, ..., xn);
    checkPost$m$S(x1, ..., xn, rac$result);
    return rac$result;
  }
  catch (JMLEntryPreconditionError rac$e) {
    throw new JMLInternalPreconditionError(rac$e);
  }
  catch (JMLAssertionError rac$e) {
    throw rac$e;
  }
  catch (Throwable rac$e) {
    checkXPost$m$S(x1, ..., xn, rac$e);
  }
  finally {
    if (/* no postcondition violation? */) {
       checkInv$S();
       checkHC$S();
    }
  }
}
```

call original method

Figure 4.3, “A Runtime Assertion Checker for the Java Modeling Language”, Y. Cheon

Pre/Post-Conditions

```
T m(T1 x1, ..., Tn xn) {
  checkInv$S();
  checkPre$m$S(x1, ..., xn);
  T rac$result;
  try {
    rac$result = orig$m(x1, ..., xn);
    checkPost$m$S(x1, ..., xn, rac$result);
    return rac$result;
  }
  catch (JMLEntryPreconditionError rac$e) {
    throw new JMLInternalPreconditionError(rac$e);
  }
  catch (JMLAssertionError rac$e) {
    throw rac$e;
  }
  catch (Throwable rac$e) {
    checkXPost$m$S(x1, ..., xn, rac$e);
  }
  finally {
    if (/* no postcondition violation? */) {
       checkInv$S();
       checkHC$S();
    }
  }
}
```

check post-conditions

Figure 4.3, "A Runtime Assertion Checker for the Java Modeling Language", Y. Cheon

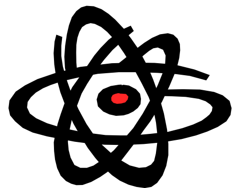
Pre/Post-Conditions

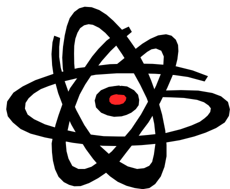
```
/*@ behavior
  @   ensures \result <==> head.next == null;
  @*/
public boolean isEmpty() {
  synchronized (head) {
    return head.next == null;
  }
}
```

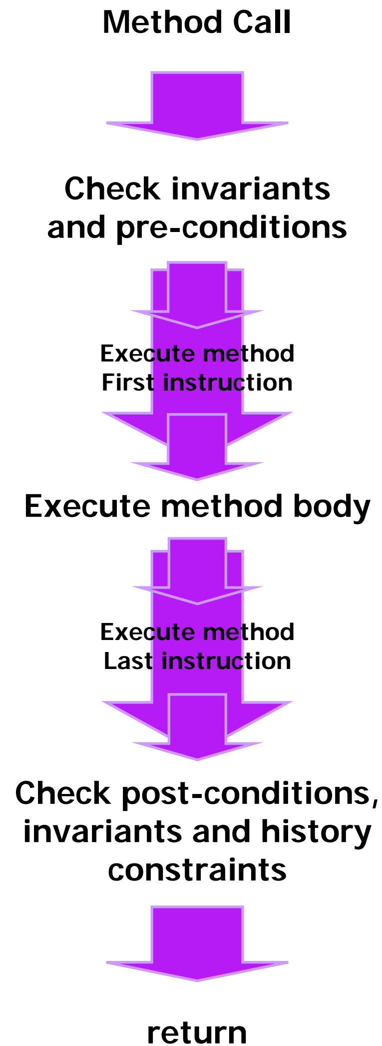
```
public boolean isEmpty() {
  …
  boolean rac$result;
  …
    rac$result = orig$isEmpty();
    checkPost$isEmpty$LinkedQueue(rac$result);
    return rac$result;
  …
}
```

At this point a thread can interleave and insert an object in the LinkedQueue; so there actually exists an execution race where the post-condition is violated.

Pre/Post-Conditions







Execute atomically to avoid interference from other threads.

Easy to do with a model checker

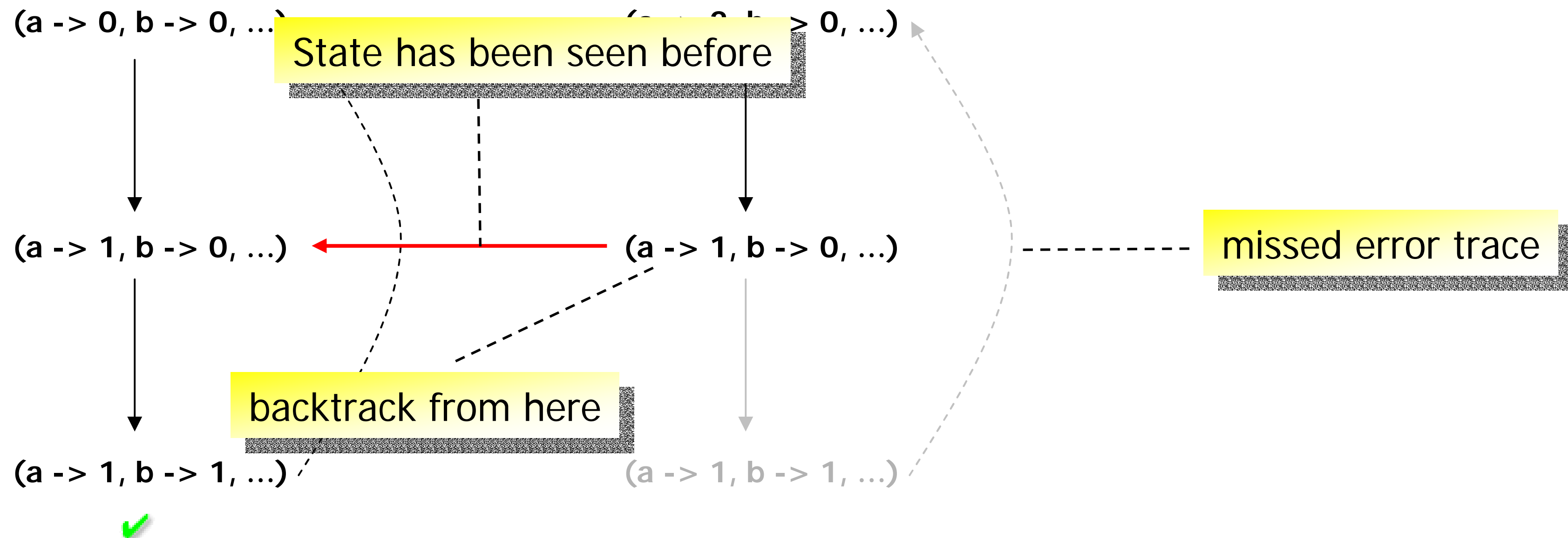
Checking JML Specs with Bogor

- Object operations
  - **`assignable`,`\reach(x)`,`\lockset`, `\fresh(`x_1`,…,`x_n`)`**
  - Quantification over objects of a specified type
- Pre-/post-conditions, invariants
- Referencing Pre-states
- Methods in JML expressions (the purity issue)

JML's \old() clause

- The **`\old()`** clause provides a way to access pre-state values in post-state conditions
  - e.g. ensures **`\old(a) + 1 == a;`**
  - asserts that the current value of **`a`** has increased by one w.r.t. the value that it had at the beginning of this method
  - very useful for constraining the behavior of a method

Issues with \old() and model checking

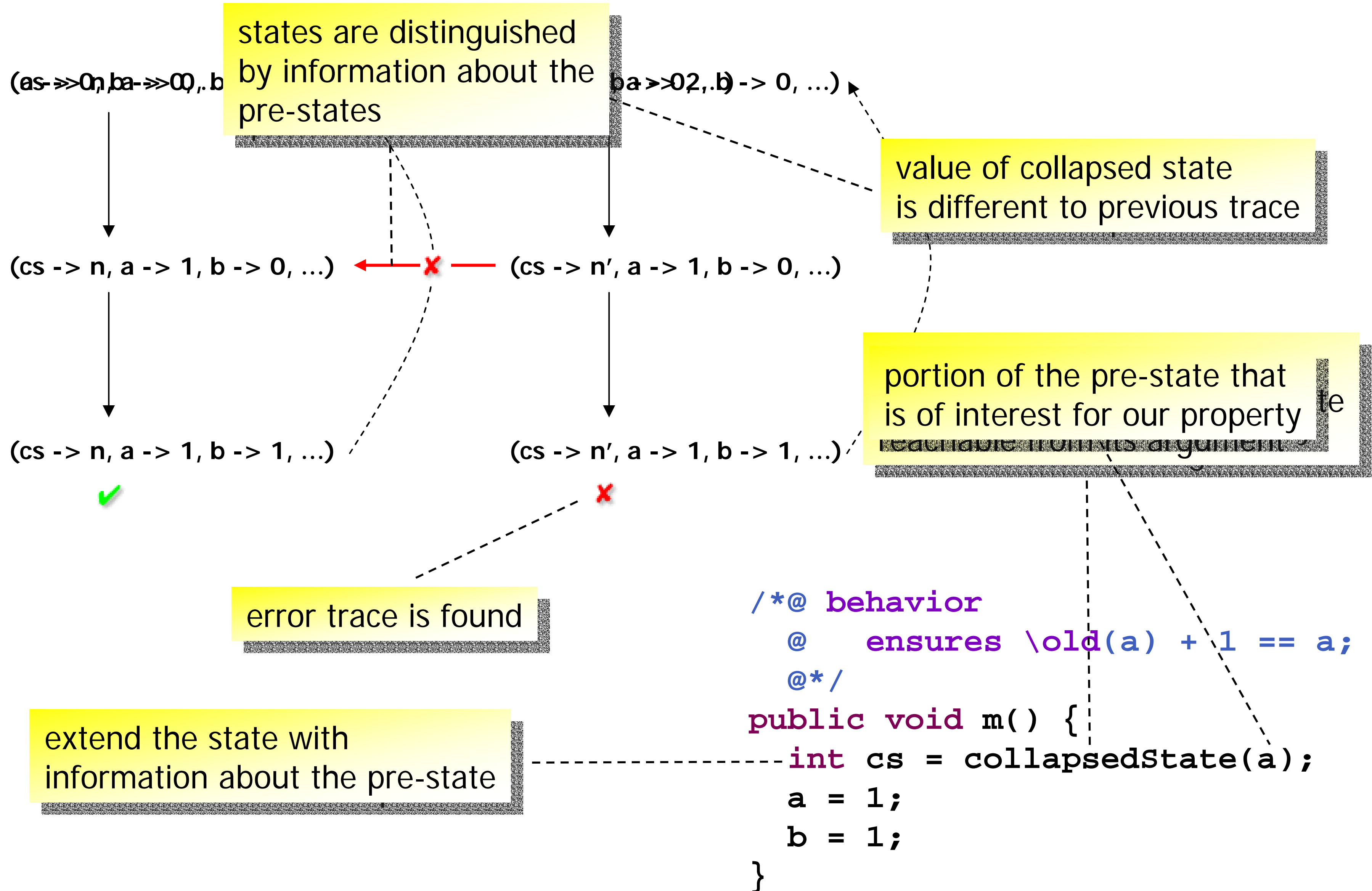



```
/*@ behavior
  @   ensures \old(a) + 1 == a;
  @*/
public void m() {
  a = 1;
  b = 1;
}
```

Issues with \old() and model checking

```
/*@ behavior
  @   ensures \old(a) + 1 == a;
  @*/
public void m() {
  a = 1;
  b = 1;
}
```

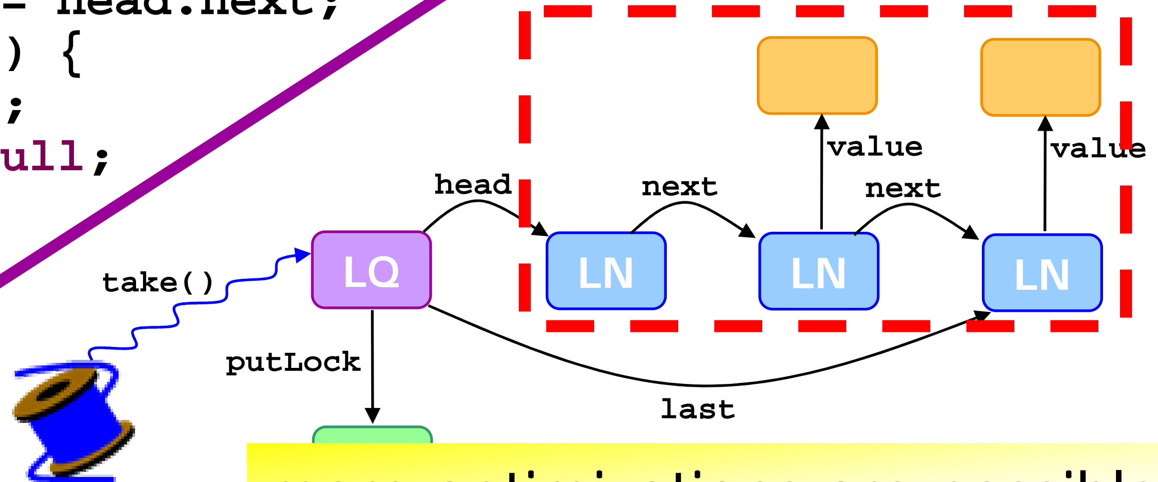
Issues with \old() and model checking



Issues with \old() and model checking

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null
  @           || (\exists LinkedNode n;
  @                \old(\reach(head)).has(n);
  @                 n.value == \re
  @                   && !(\reach(
  @*/
protected Object extract() {
  Object x = null;
  LinkedNode first = head.next;
  if (first != null) {
    x = first.value;
    first.value = null;
    head = first;
  }
  return x;
}
```

use collapse compression for efficiency



more optimizations are possible

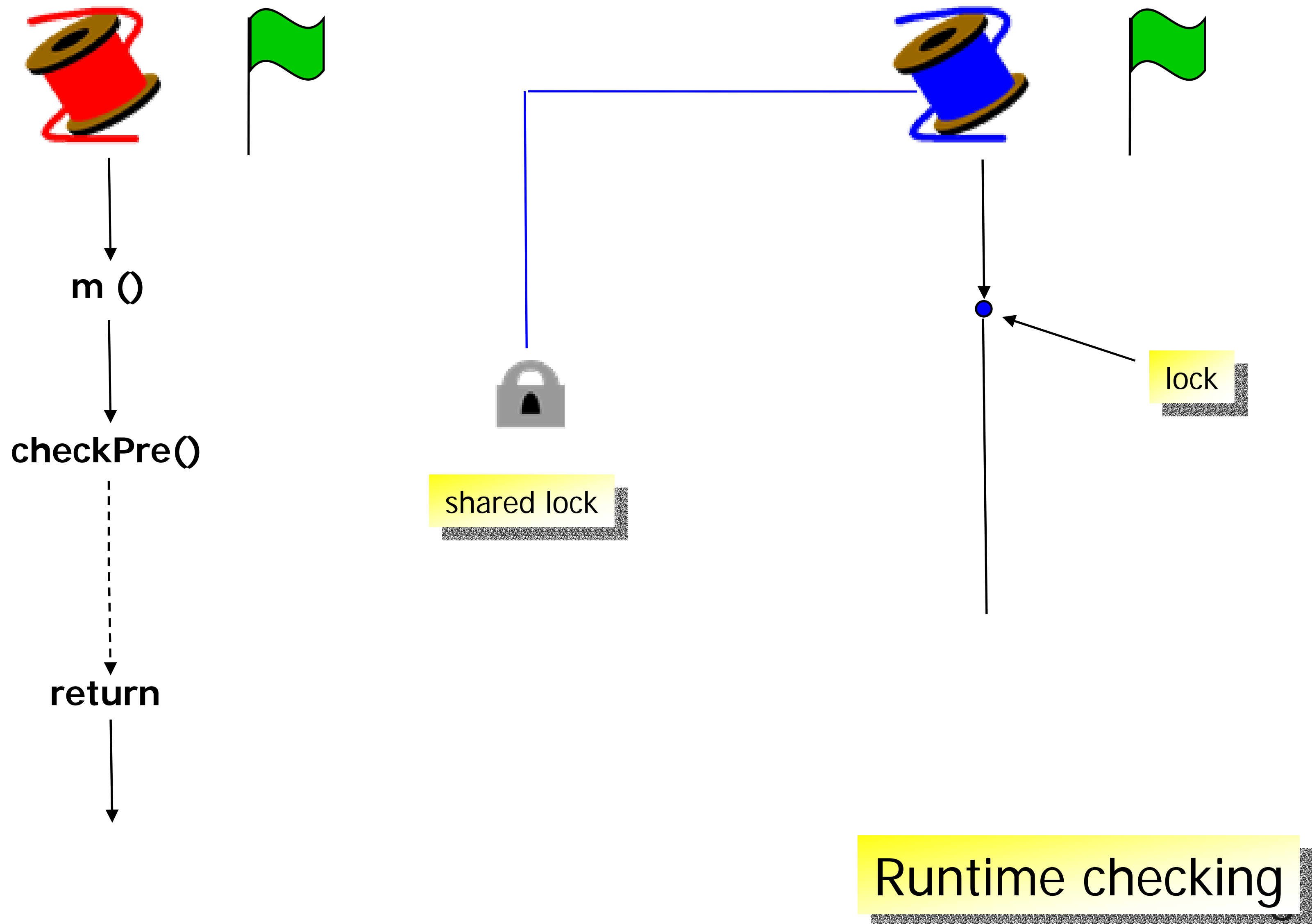
Checking JML Specs with Bogor

- Object operations
  - **assignable,\reach(x),\lockset, \fresh(x_1,…,x_n)**
  - Quantification over objects of a specified type
    Pre-/post-conditions, invariants
- Referencing Pre-states
- Methods in JML expressions (the purity issue)

Methods in JML expressions

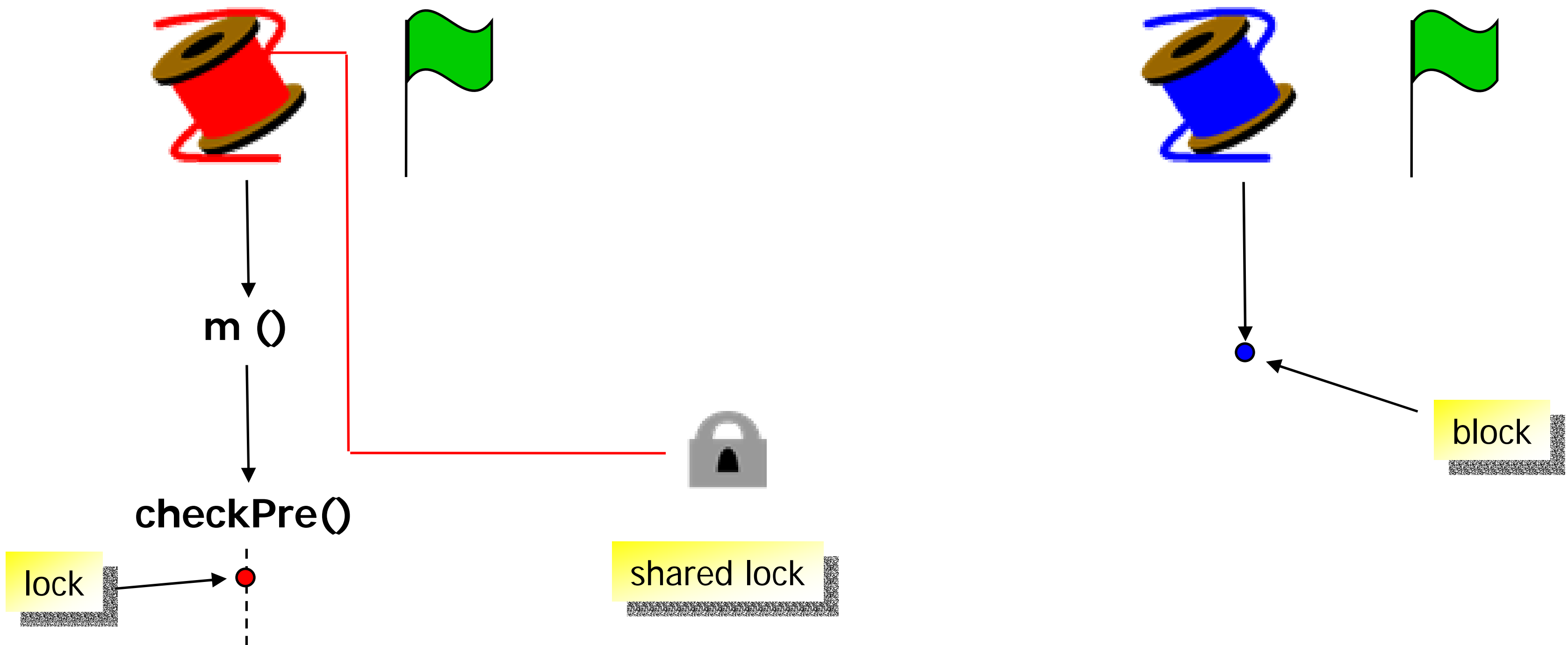
- All methods called from JML expressions must be **`pure`** :
  - A pure method must be guaranteed to have no side effects
    - … must refine **`assignable \nothing;`**
  - JML considers the locking used in synchronization as a kind of side effect
    - **`synchronized`** methods cannot be declared **`pure`**

The purity issue



Runtime checking

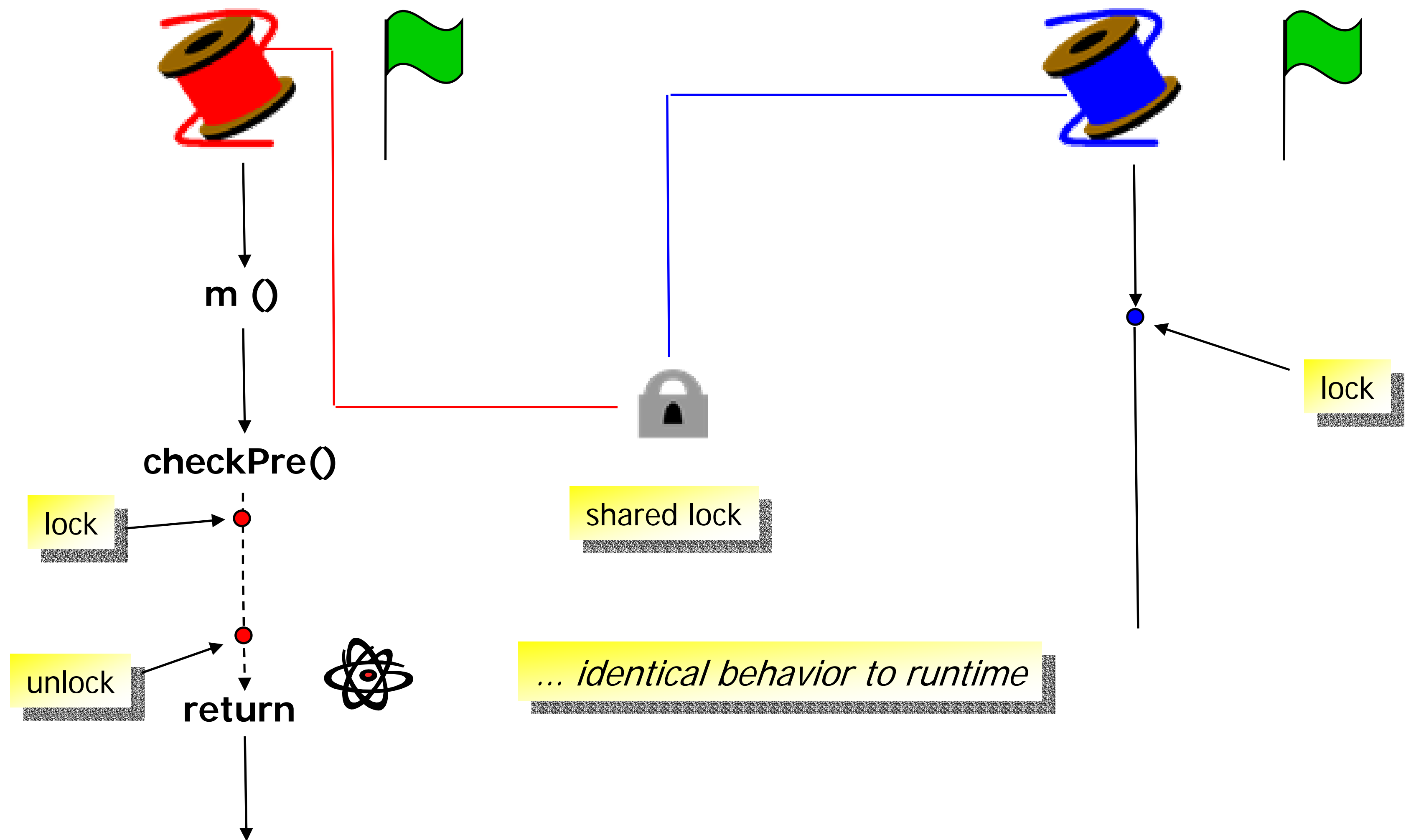
The purity issue



... different behavior than without runtime checking

Runtime checking

The purity issue



... allows to define a kind of weak purity

Model checking

LinkedQueue Example (JML)

```
/*@ behavior
  @   ensures \result <==> head.next == null;
  @*/
public boolean isEmpty() {
  synchronized (head) {
    return head.next == null;
  }
}
```

```
public class LinkedNode {
  public Object value;
  public LinkedNode next;

  /*@ behavior
    @   ensures value == x;
    @*/
  public LinkedNode(Object x) {
    value = x;
  }
}

public c
  protec
  protec
  protec
  protec

  //@ in
  //@ in

  /*@ be
    @
    @
    @*/
  public
    putL
    head
  }

  /*@ be
    @
    @*/
  public boolean isEmpty() {
    synchronized (head) {
      return head.next == null;
    }
  }

  /*@ behavior
    @   requires n != null;
    @   assignable last, last.next;
    @*/
  protected void refactoredInsert(LinkedNode n) {
    last.next = n;
    last = n;
  }
```

```
/*@ behavior
  @   requires x != null;
  @   ensures true;
  @ also behavior
  @   requires x == null;
  @   signals (Exception e) e instanceof IllegalArgumentException;
  @*/
public void put(Object x) {
  if (x == null)
    throw new IllegalArgumentException();
  insert(x);
}

n;
has(n);

has(n)));

}

/*@ behavior
  @   requires x != null;
  @   ensures last.value == x && \fresh(last);
  @*/
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) refactoredInsert(p);
    if (waitingForTake > 0) putLock.notify();
    return;
  }
}
}
```

LinkedQueue Example

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                               \old(\reach(head)).has(n);
  @                             n.value == \result && !(\reach(head).has(n)));
  @*/
protected Object refactoredExtract() {
  Object x = null;
  LinkedNode first = head.next;
  if (first != null) {
    x = first.value;
    first.value = null;
    head = first;
  }
  return x;
}
```

```
class LinkedNode {
  public Object value;
```

```
    last = head = new LinkedNode(null);
  }

  /*@ behavior
    @   ensures \result <==> head.next == null;
    @*/
  public boolean isEmpty() {
    synchronized (head) {
      return head.next == null;
    }
  }

  /*@ behavior
    @   requires n != null;
    @   assignable last, last.next;
    @*/
  protected void refactoredInsert(LinkedNode n) {
    last.next = n;
    last = n;
  }
```

```
/*@ behavior
  @   requires x != null;
```

```
      first.value = null;
      head = first;
    }
    return x;
}

/*@ behavior
  @   requires_redundantly x != null;
  @   ensures last.value == x && \fresh(last);
  @*/
protected void insert(Object x) {
  synchronized (putLock) {
    LinkedNode p = new LinkedNode(x);
    synchronized (last) {
      refactoredInsert(p);
    }
    if (waitingForTake > 0)
      putLock.notify();
    return;
  }
}
```

LinkedQueue Example

```
class LinkedNode {
  public Object value;
```

```
/*@ behavior
  @   requires x != null;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                               \old(\reach(head)).has(n);
  @                             n.value == \result && !(\reach(head).has(n)));
  @*/
protected 
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
        /|linkedqueue.LinkedNode.value|\ == result
      && !(Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

```
  last = head = new
}

/*@ behavior
  @   ensures \resu
  @*/
public boolean isEm
  synchronized (hea
    return head.nex
  }
}

/*@ behavior
  @  requires n !=
  @  assignable las
  @*/
protected void refa
  last.next = n;
  last = n;
}
```

LinkedQueue Example

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                                \old(\reach(head)).has(n);
  @                              n.value == \result && !(\reach(head).has(n)));
  @*/
protected
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
        /|linkedqueue.LinkedNode.value|\ == result
      && !Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
        Set.exists2Context<(|java.lang.Object|),
                           (|java.lang.Object|),
                           Set.type<(|java.lang.Object|)>>(
          specFun1,
          State.preVal<Set.type<(|java.lang.Object|)>>(
            State.reachSet<(|java.lang.Object|)>(
              [|r0|]./|linkedqueue.LinkedQueue.head|\),
            State.getCurrentThreadId()),
          [|r1|],
          State.reachSet<(|java.lang.Object|)>(
            [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

LinkedQueue Example

```
class LinkedNode {
  public Object value;
```

```
/*@ behavior
  @   requires x != null;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                                \old(\reach(head)).has(n);
  @                               n.value == \result && !(\reach(head).has(n)));
  @*/
protected 
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
        /|linkedqueue.LinkedNode.value|\ == result
      && !Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

```
  putLock = new Obj
  last = head = new
}

/*@ behavior
  @   ensures \resu
  @*/
public boolean isEm
  synchronized (hea
    return head.nex
  }
}

/*@ behavior
  @  requires n !=
  @  assignable las
  @*/
protected void refa
  last.next = n;
  last = n;
}
```

```
}
```

LinkedQueue Example

```
class LinkedNode {
  public Object value;
```

```
/*@ behavior
  @   requires x != null;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                                \old(\reach(head)).has(n);
  @                               n.value == \result && !(\reach(head).has(n)));
  @*/
protected 
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
     (((|linkedqueue.LinkedNode|)) n)
      /|linkedqueue.LinkedNode.value|\ == result
     && !Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

```
  last = head = new
}

/*@ behavior
  @   ensures \resu
  @*/
public boolean isEm
  synchronized (hea
    return head.nex
  }
}

/*@ behavior
  @  requires n !=
  @  assignable las
  @*/
protected void refa
  last.next = n;
  last = n;
}
```

LinkedQueue Example

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                  \old(\reach(head)).has(n);
  @                  n.value == \result && !(\reach(head).has(n)));
  @*/
protected
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
    (((|linkedqueue.LinkedNode|)) n)
     /|linkedqueue.LinkedNode.value|\ == result
    && !Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

LinkedQueue Example

```
class LinkedNode {
  public Object value;
```

```
/*@ behavior
  @   requires x != null;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                              \old(\reach(head)).has(n);
  @                            n.value == \result && !(\reach(head).has(n)));
  @*/
protected 
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
       /|linkedqueue.LinkedNode.value|\ == result
      && !(Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

```
    last = head = new
  }

  /*@ behavior
    @   ensures \resu
    @*/
  public boolean isEm
    synchronized (hea
      return head.nex
    }
  }

  /*@ behavior
    @   requires n !=
    @   assignable las
    @*/
  protected void refa
    last.next = n;
    last = n;
  }
```

LinkedQueue Example

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                               \old(\reach(head)).has(n);
  @                            n.value == \result && !(\reach(head).has(n)));
  @*/
protected (
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
        /|linkedqueue.LinkedNode.value|\ == result
      && !(Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

LinkedQueue Example

```
class LinkedNode {
  public Object value;
```

```
/*@ behavior
  @   requires x != null;
```

```
/*@ behavior
  @   assignable head, head.next.value;
  @   ensures \result == null || (\exists LinkedNode n;
  @                                \old(\reach(head)).has(n);
  @                             n.value == \result && !(\reach(head).has(n)));
  @*/
protected 
  Object x
  LinkedNo
  if (firs
    x = fi
    first.
    head =
  }
  return x
}
```

```
fun specFun1((|java.lang.Object|) n,
             (|java.lang.Object|) result,
             Set.type<(|java.lang.Object|)> s) returns boolean =
  n instanceof (|linkedqueue.LinkedNode|) ?
      (((|linkedqueue.LinkedNode|)) n).
       /|linkedqueue.LinkedNode.value|\ == result
      && !(Set.contains<(|java.lang.Object|)>(s, n))
  : false;

...

assert([|r1|] == null ||
       Set.exists2Context<(|java.lang.Object|),
                          (|java.lang.Object|),
                          Set.type<(|java.lang.Object|)>>(
         specFun1,
         State.preVal<Set.type<(|java.lang.Object|)>>(
           State.reachSet<(|java.lang.Object|)>(
             [|r0|]./|linkedqueue.LinkedQueue.head|\),
           State.getCurrentThreadId()),
         [|r1|],
         State.reachSet<(|java.lang.Object|)>(
           [|r0|]./|linkedqueue.LinkedQueue.head|\)));
```

```
    last = head = new
  }

  /*@ behavior
    @   ensures \resu
    @*/
  public boolean isEm
    synchronized (hea
      return head.nex
    }
  }

  /*@ behavior
    @   requires n !=
    @   assignable las
    @*/
  protected void refa
    last.next = n;
    last = n;
  }
```

JML Language Coverage

- assignable ✔
- invariant ✔
- \old() ✔
- constraint ✔
- \forall and \exists ✔
- \max, \min, \product and \sum ✔
- \num_of ✔
- Set Comprehension ✔
- \fresh() ✔
- \lockset ✔
- \nonnullelements() ✔
- \not_modified() ✔
- \reach() ✔
- maintaining ✘
- decreasing ✘
- diverges ✘
- \signals ✔
- \when ✘

large language coverage…

Preliminary Results

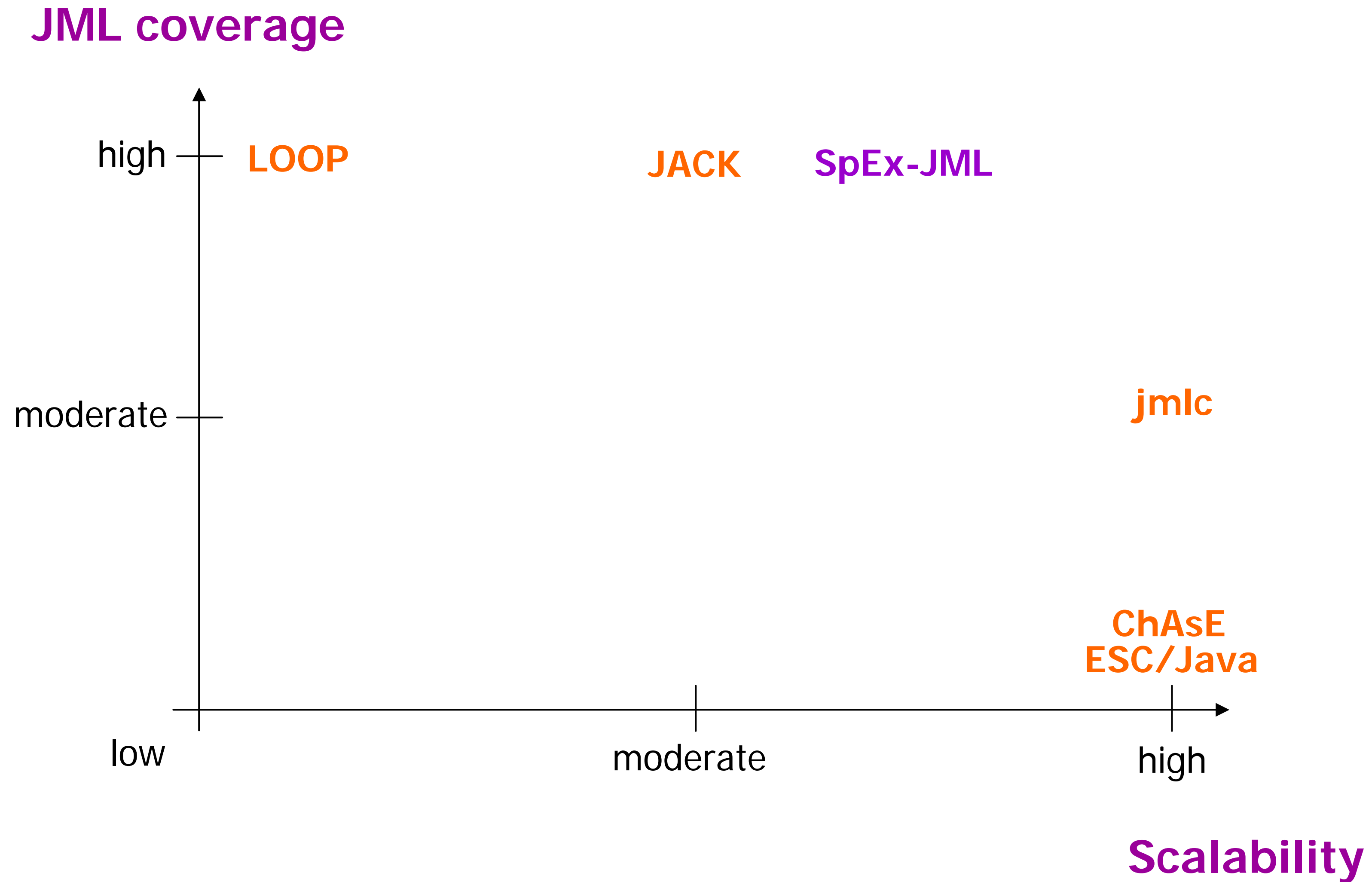
		w/ JML	w/o JML
LinkedQueue[12]	228 loc	\fresh, \reach, \old, signals, \exists	
3 threads 22 objects		17064 states 38 sec/3.7 MB	11594 states 21 sec/2.3 MB
5 threads 32 objects		1364007 states 14557 sec/140.5 MB	423538 states 2415 sec/46.4 MB
RWVSN[12]	227 loc	\old	
4 threads 5 objects		2621 states 2 sec/1.5 MB	2255 states 2 sec/1.0 MB
7 threads 9 objects		4995560 states 34804 sec/463.7 MB	4204332 states 26153 sec/366.3 MB
ReplicatedWorkers[4]	543 loc	\fresh, \old, \reach	
4 threads 19 objects		322016 states 897 sec/29.8 MB	269593 states 716 sec/26.6 MB
6 threads 21 objects		12347415 states 30191 sec/391.8 MB	10016554 states 21734 sec/282.5 MB

Bogor's Reduction Algorithms – Enables Checking JML Specs

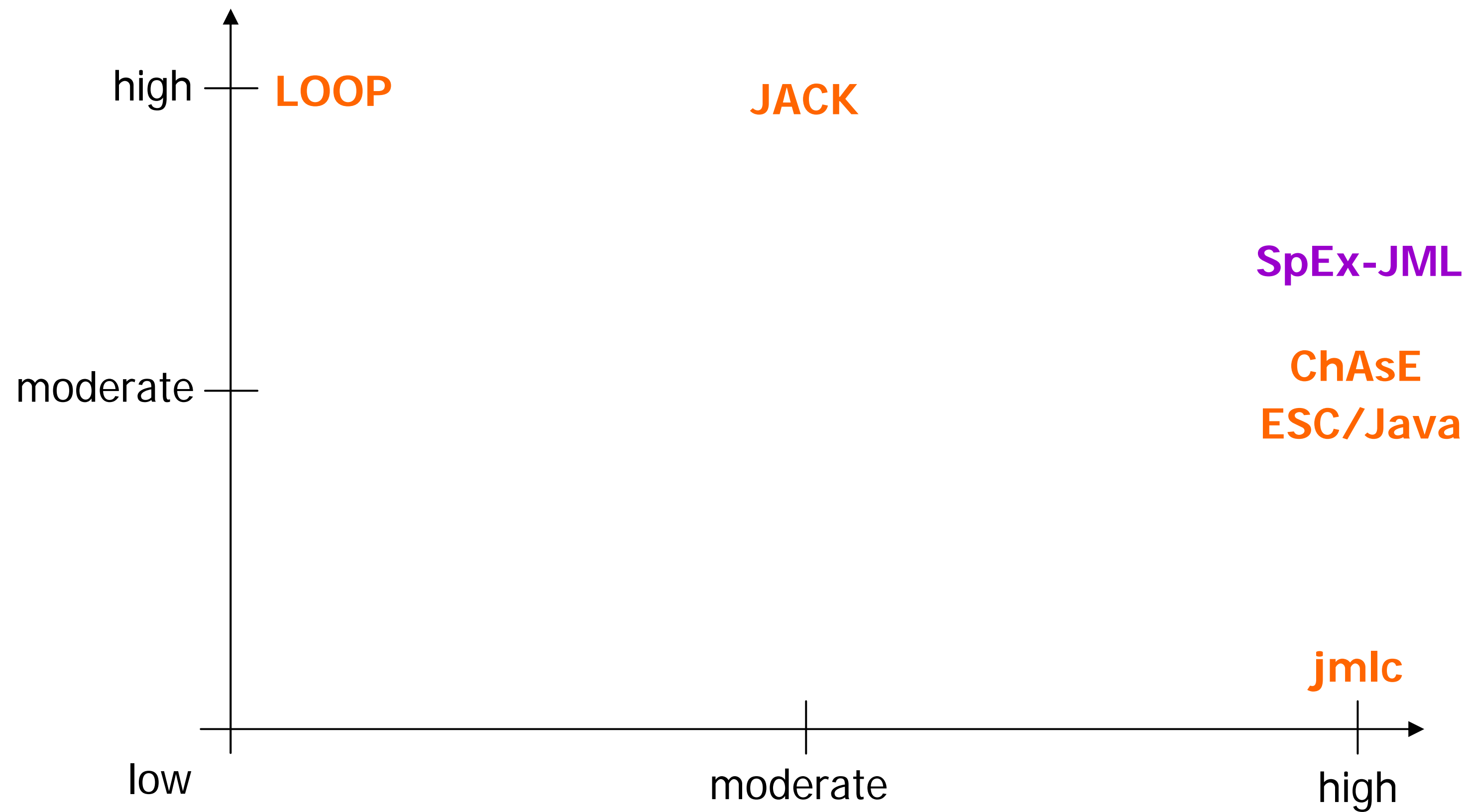
	w/ POR: w/ JML	w/ POR: w/o JML	w/o POR: w/ JML	w/o POR: w/o JML
LinkedQueue[12]	228 loc	\fresh, \reach, \old, signals, \exists		
3 threads 22 objects	2833 states 10 sec/1.6 MB	1533 states 5 sec/1.0 MB	17064 states 38 sec/3.7 MB	11594 states 21 sec/2.3 MB
5 threads 32 objects	39050 states 144 sec/5.9 MB	12807 states 72 sec/2.5 MB	1364007 states 14557 sec/140.5 MB	423538 states 2415 sec/46.4 MB
RWVSN[12]	227 loc	\old		
4 threads 5 objects	183 states 1 sec/1.0 MB	183 states 1 sec/0.8 MB	2621 states 2 sec/1.5 MB	2255 states 2 sec/1.0 MB
7 threads 9 objects	18398 states 185 sec/6.8 MB	18398 states 144 sec/3.0 MB	4995560 states 34804 sec/463.7 MB	4204332 states 26153 sec/366.3 MB
ReplicatedWorkers[4]	543 loc	\fresh, \old, \reach		
4 threads 19 objects	1751 states 14 sec/2.1 MB	1751 states 13 sec/1.9 MB	322016 states 897 sec/29.8 MB	269593 states 716 sec/26.6 MB
6 threads 21 objects	10154 states 99 sec/3.3 MB	10154 states 92 sec/2.8 MB	12347415 states 30191 sec/391.8 MB	10016554 states 21734 sec/282.5 MB

Indicates little overhead compared with simply exploring the state-space

Assessment of JML Tools



Assessment of JML Tools



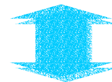
Assessment of SpEx-JML

- Model checking based JML verification technique
  - Bogor
- Excellent automation usability
  - similar to that of jmlc
- Very high JML coverage
- Moderate behavior coverage
  - determined by test harness
    - but sound with respect to the test harness
- Good scalability
  - unit level reasoning

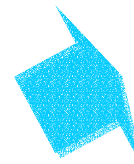
JMLEclipse

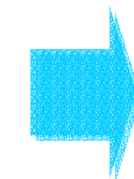
```
/*@ behavior
    @    requires x != null;
    @    ensures true;
    @also
    @ behavior
    @    requires x == null;
    @    signals (Exception e) e instanceof
    @                IllegalArgumentException;
    @*/
  public void put(Object x)
  {
    if (x == null)
      throw new IllegalArgumentException();
    insert(x);
  }
```

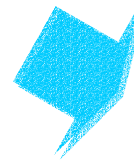
JML annotated Java source



JML well-formedness checker

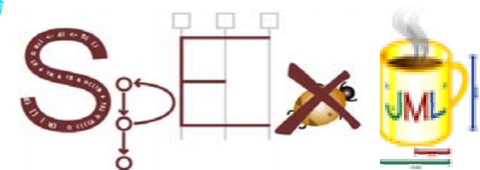


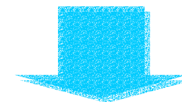




jmlc

other tool





JMLEclipse

Java - LinkedQueue.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- bogor-site
- bogor-temp
- bogor-tests
- bogor-thesis
- bogor-tool
- bogor-ui
- bogor-web
- book-source
- book-web
- cef-core
- cef-xml
- cis842
- examples
- hdccsr04
- jaxb
- job-talk
- mc-comp
- test
  - src
    - (default package)
      - LinkedNode.java
      - LinkedQueue.java
      - LinkedQueueDriver.java

Package Explorer | Hierarchy

Outline

- LinkedQueue
  - putLock : Object
  - head : LinkedNode
  - extract()
  - atomicExtract()
  - put(Object)
  - take()
  - peek()
  - isEmpty()

LinkedQueue.java

```
public class LinkedQueue {
    protected final /*@ non_null @*/ Object putLock;
    protected /*@ non_null @*/ LinkedNode head;
    protected /*@ non_null @*/ LinkedNode last = head;
    protected int waitingForTake = 0;

    //@ instance invariant waitingForTake && 0;
    //@ instance invariant \reach(head).has(last);

    /*@ behavior
      @   assignable head,
      @   ensures \fresh(h
      @*/
    public LinkedQueue() {
      putLock = new Object();
      head = new LinkedNode(null);
    }

    /*@ behavior
      @    requires_redundantly x != null;
      @    ensures last.value == x && \fresh(last);
      @*/
    protected void insert(Object x)
      ynchronized (putLock)
        LinkedNode p = new LinkedNode(x);
```

JML well-formedness checking

JML syntax highlighting

Tasks (Filter matched 8 of 122 items)

Description	Resource	In Folder
error: Expected an expression of type boolean and not int	LinkedQueue.java	test/src
notice: There is no package declaration for this compilation unit [KOPI]	LinkedQueue.java	test/src
warning: Else part of if-statement should be enclosed in block statement [JCC 7.2, 7.4]	LinkedQueue.java	test/src
warning: Runtime exception thrown while typechecking compilation unit	LinkedQueue.java	test/src
warning: Then part of if-statement should be enclosed in block statement [JCC 7.2, 7.4]	LinkedQueue.java	test/src
warning: Then part of if-statement should be enclosed in block statement [JCC 7.2, 7.4]	LinkedQueue.java	test/src
warning: Then part of if-statement should be enclosed in block statement [JCC 7.2, 7.4]	LinkedQueue.java	test/src
warning: Then part of if-statement should be enclosed in block statement [JCC 7.2, 7.4]	LinkedQueue.java	test/src

Tasks | Console | Bogor Counter Examples

Writable | Insert | 7 : 48

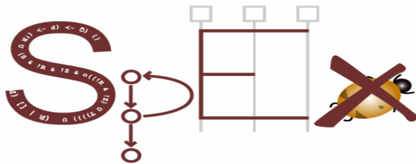
Future Work

- Proposal for concurrency specifications in JML
  - thread-locality
  - regionized pre-/post-conditions
  - atomicity, etc.
- JMLEclipse as an open ended JML plugin for Eclipse
- Other specification formalisms

For More Information…

SAnToS Laboratory,
Kansas State University
http://www.cis.ksu.edu/santos

Bogor Project
http://bogor.projects.cis.ksu.edu



SpEx Project
http://spex.projects.cis.ksu.edu

JMLEclipse Project
http://jmleclipse.projects.cis.ksu.edu

Bandera Project
http://bandera.projects.cis.ksu.edu