Automatic Extraction of Object-Oriented Observer Abstractions from Unit-Test Executions

Tao Xie David Notkin

Dept. of Computer Science & Engineering
University of Washington, Seattle

Nov. 2004

ICFEM 04, Seattle

Motivation

- Manually created unit tests
  - Valuable but often insufficient
- Automatically generated unit-test inputs
  - A large number
  - Without specifications, test-result inspection is impractical

Motivation

- Manually created unit tests
  - Valuable but often insufficient
- Automatically generated unit-test inputs
  - A large number
  - Without specifications, test-result inspection is impractical

Test selection for inspection
[Xie&Notkin ASE 03]

Motivation

- Manually created unit tests
  - Valuable but often insufficient
- Automatically generated unit-test inputs
  - A large number
  - Without specifications, test-result inspection is impractical

Test selection for inspection
[Xie&Notkin ASE 03]

Test abstraction for inspection

Synopsis

- Dynamically extract observer abstractions from test executions
  - A set of object state machines
  - States represented by observer returns
  - Focus on both method returns and object-state transitions

- Succinct and useful for inspection
  - bug finding, bug isolation, component understanding, etc.

Outline

- Motivation
- Observer Abstractions
- Experience
- Related Work
- Conclusion

Object State Machine (OSM)

M = $(I, O, S, \delta, \lambda, INIT)$ of a class c

- I: method calls in c's interface
- O: returns of method calls
- S: states of c's objects
- δ: $S \times I \rightarrow P(S)$ state transition function
- λ: $S \times I \rightarrow P(O)$ output function
- $INIT$: initial state

Object State Machine (OSM)

M = $(I, O, S, \delta, \lambda, INIT)$ of a class c

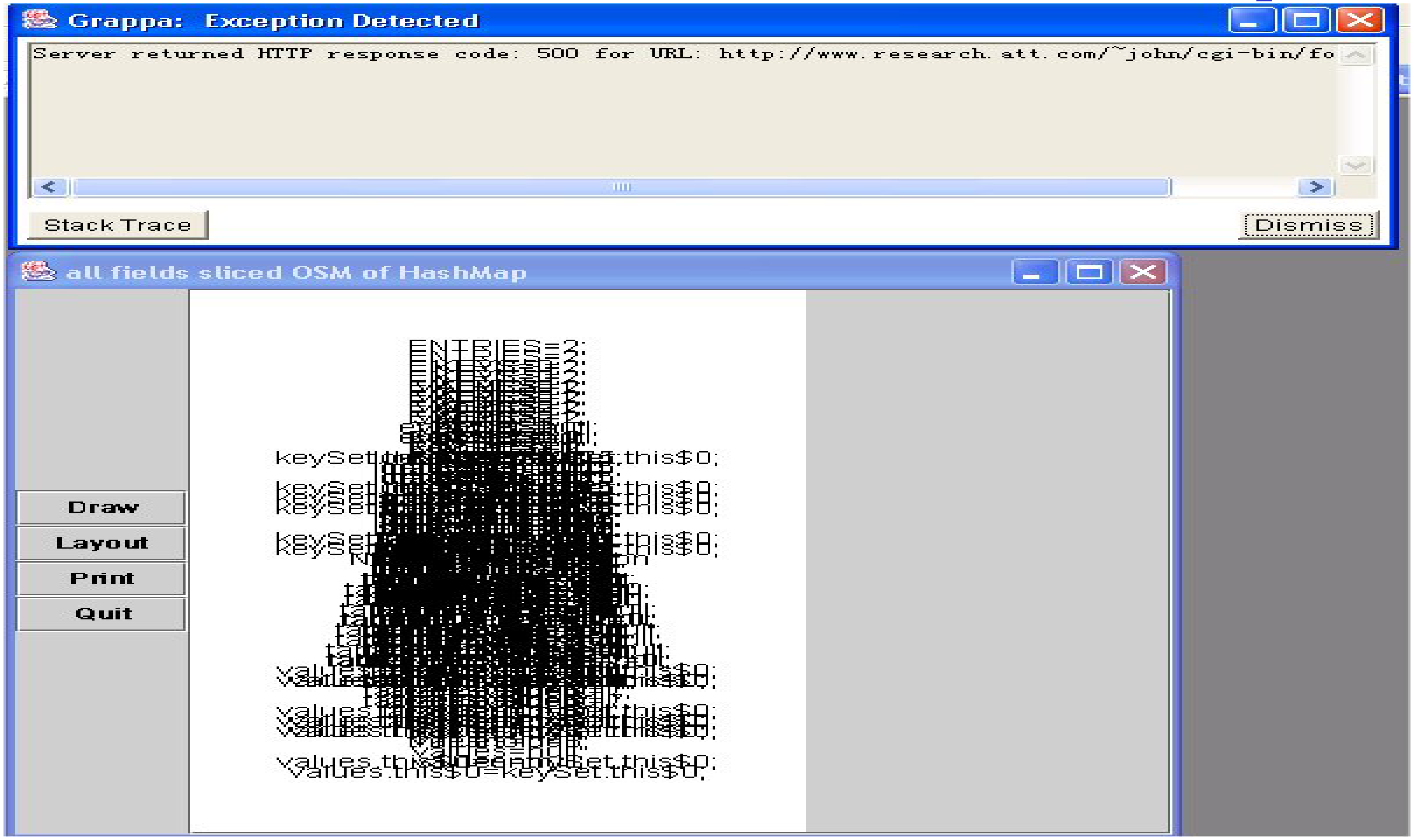
- I: method calls in c's interface
- O: returns of method calls
- S: states of c's objects
- δ: $S \text{ X } I \rightarrow P(S)$ state transition function
- λ: $S \text{ X } I \rightarrow P(O)$ output function
- $INIT$: initial state

States can be concrete or abstract

Concrete State Representation

- Rostra includes five techniques for state representation [Xie, Marinov, and Notkin ASE 04]
- WholeState technique
  - Traversal: collect the values of all the fields transitively reachable from the object
  - Linearization: remove reference addresses but keep reference relationship
- State comparison is reduced to sequence comparison

Concrete OSM of HashMap



Grappa: Exception Detected

```
Server returned HTTP response code: 500 for URL: http://www.research.att.com/~john/cgi-bin/fo
```

- **58 concrete states, 5186 tests generated by Parasoft Jtest 4.5,**
- **Too complex to be useful** (even too complex for graphviz [AT&T])

Abstract State Representation

- Abstraction function: observer
  - A public method whose return type is not void.
- Abstract state representation:
  - Return values of observers invoked on the concrete state
- State comparison is reduced to sequence comparison
- Observer abstraction
  - An OSM with observer-abstracted states

Construction of Observer Abstractions

- Run the existing tests (generated by Parasoft Jtest)
  - Collect concrete state representation
- Augment the existing tests
  - Invoke all method arguments on each concrete state
    - Collect abstract state representations (observer returns)
    - Facilitate inspections (e.g. missing transitions)
- Generate one OSM for each observer method by default
  - Group transitions (of the same method) with the same starting and ending states

Outline

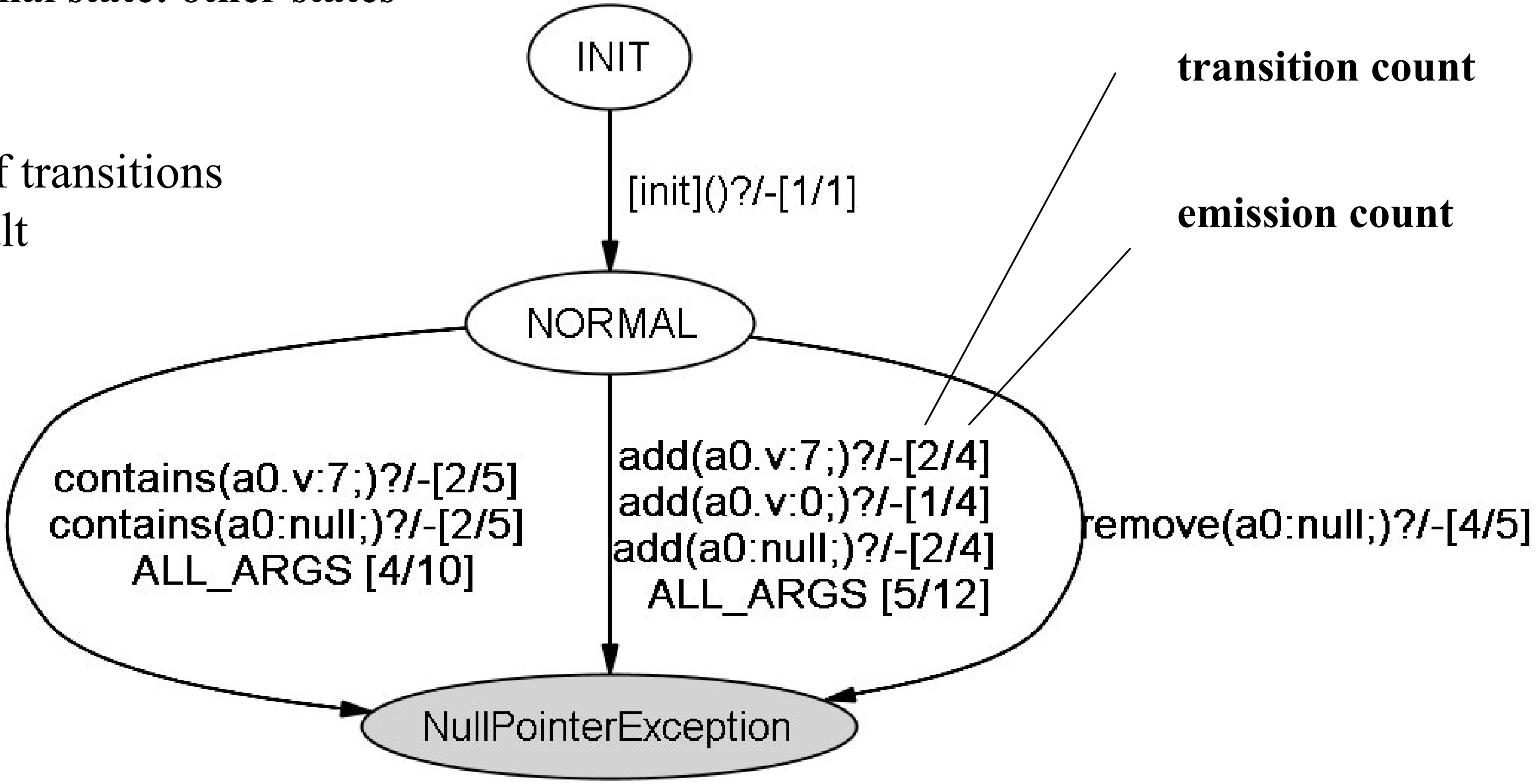
- Motivation
- Observer Abstractions
- Experience
- Related Work
- Conclusion

exception OSM of BinSearchTree

Exception observer

- **Exception state: a state reached after invoking an exception-throwing method**
- **Normal state: other states**

Hide self transitions by default

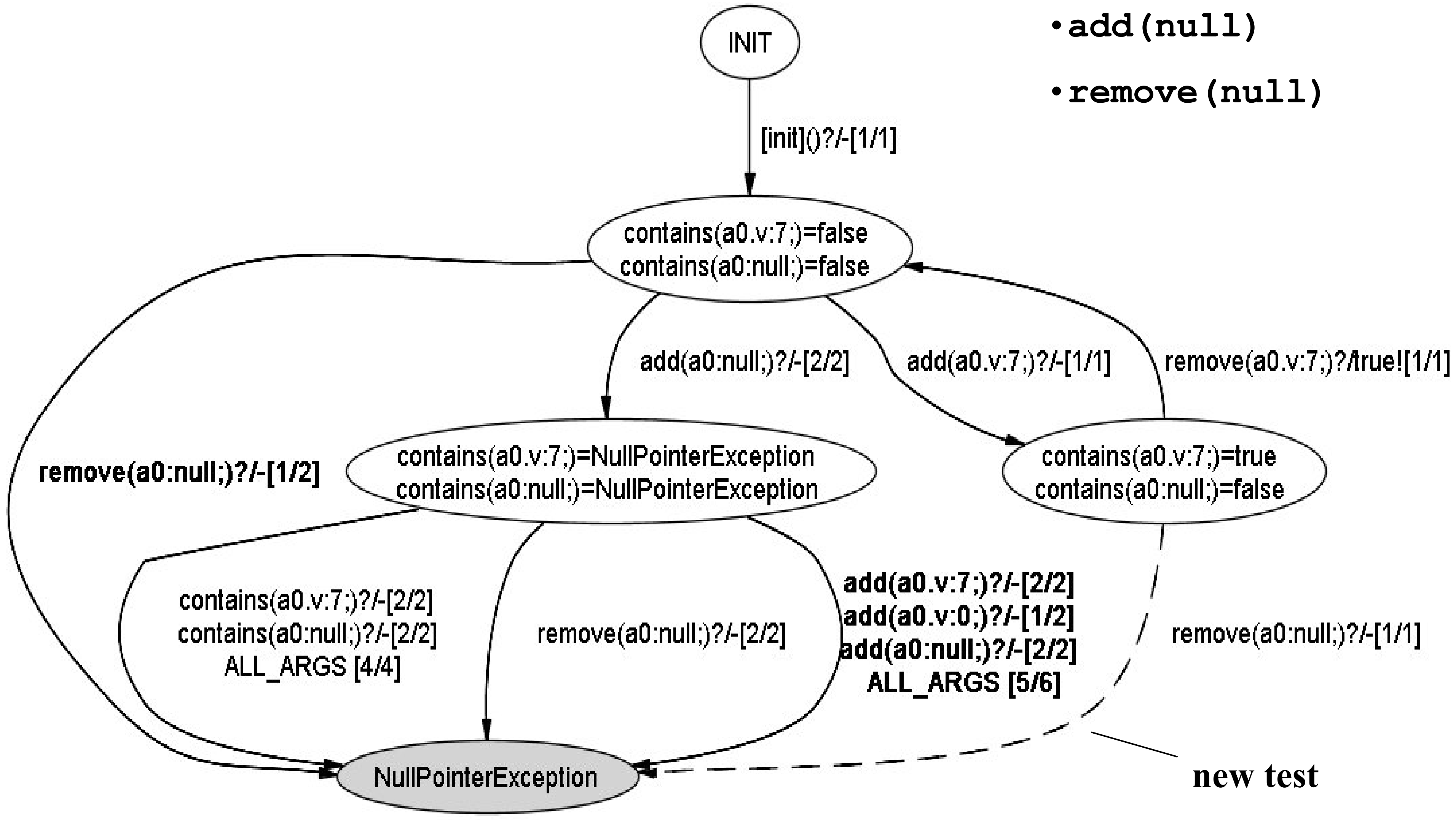


- **Bug/illegal-input isolation**

  where to put preconditions/guard-condition checking?

contains OSM of BinSearchTree

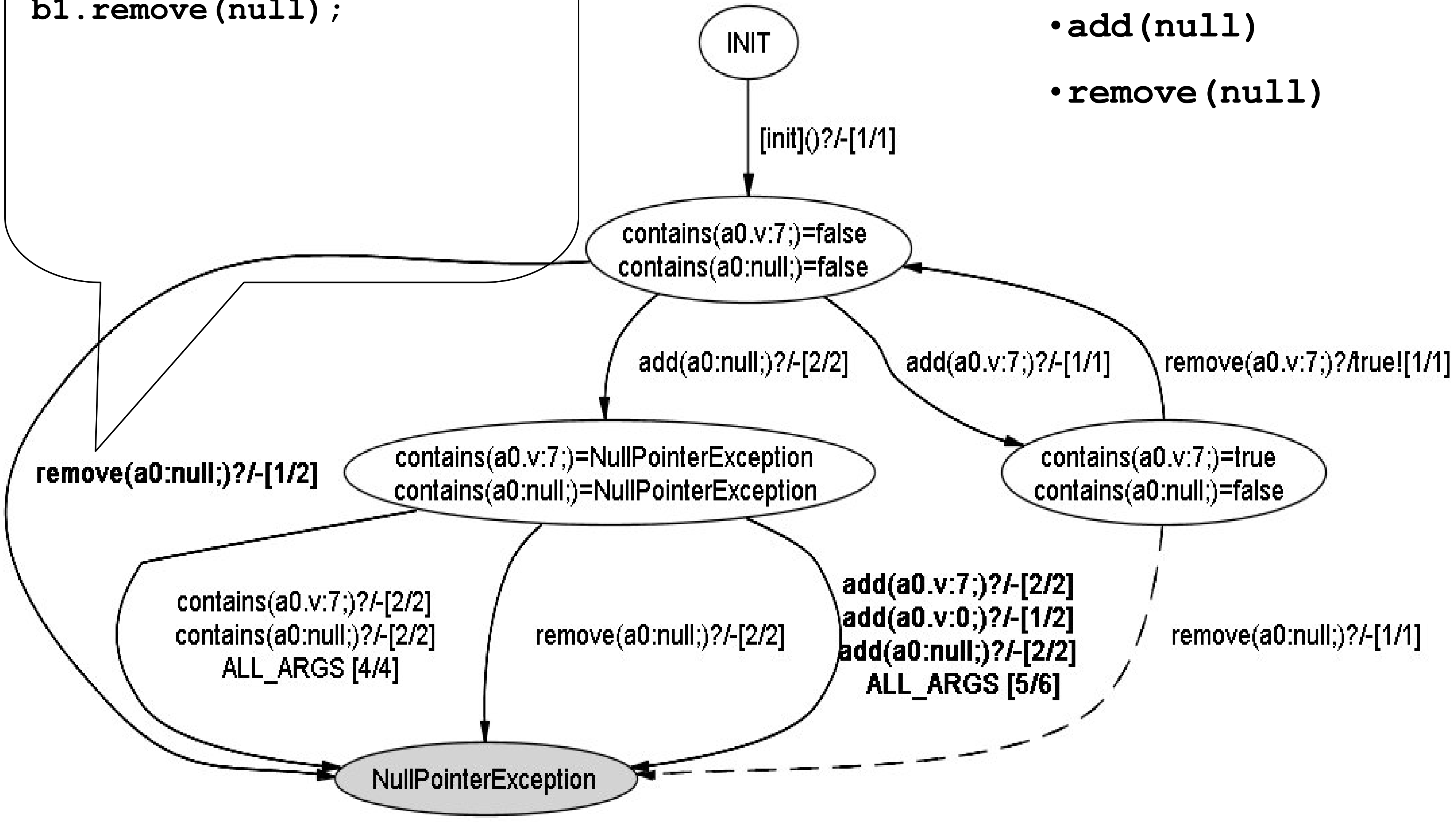
- **Bug/illegal-input isolation**
  - **`add(null)`**
  - **`remove(null)`**



contains OSM of BinSearchTree

```
Test 1 (T1):
BSTree b1 = new BSTree();
b1.remove(null);
```

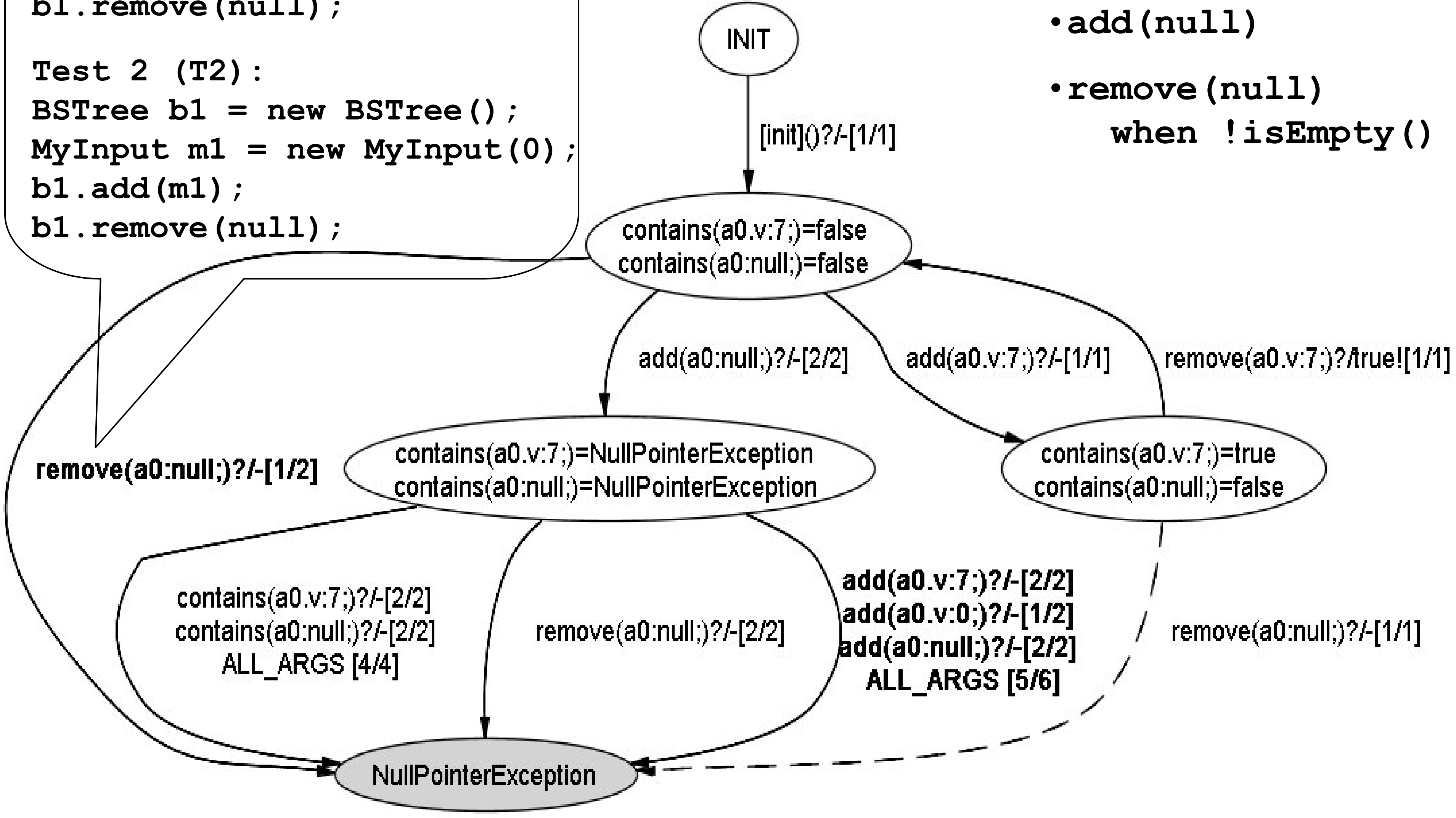
- Bug/illegal-input isolation
  - `add(null)`
  - `remove(null)`



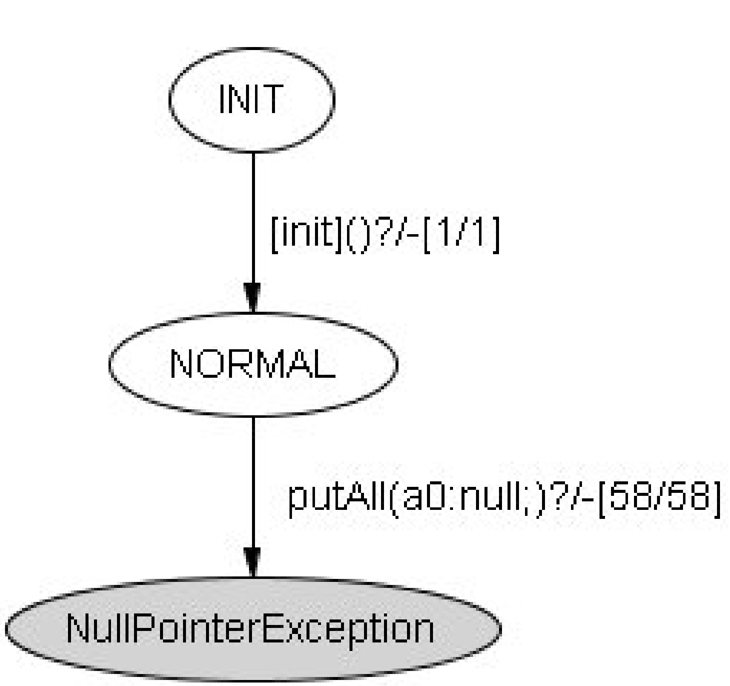
contains OSM of BinSearchTree

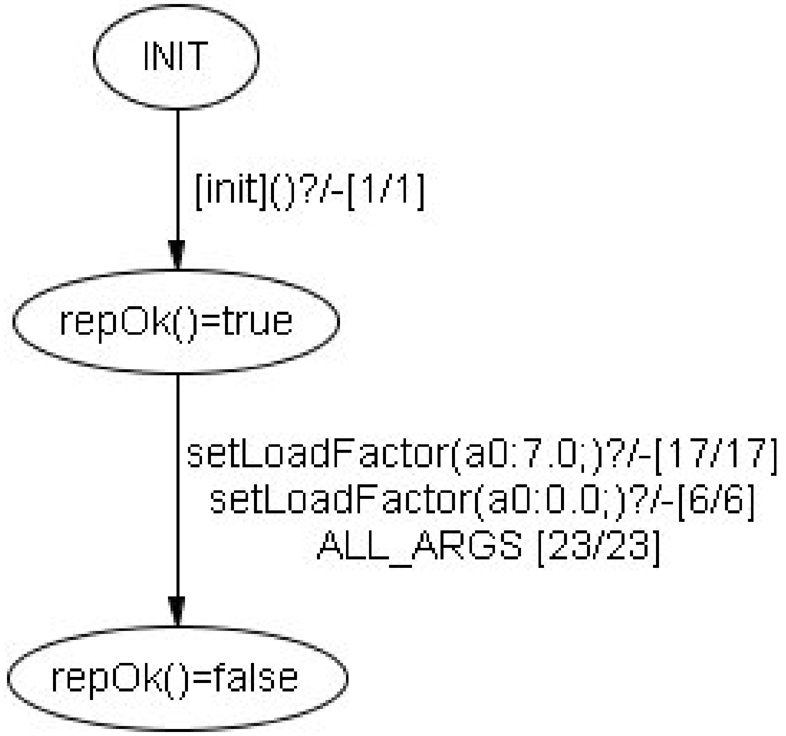
```
Test 1 (T1):
BSTree b1 = new BSTree();
b1.remove(null);

Test 2 (T2):
BSTree b1 = new BSTree();
MyInput m1 = new MyInput(0);
b1.add(m1);
b1.remove(null);
```

- **Bug/illegal-input isolation**
  - `add(null)`
  - `remove(null)` `when !isEmpty()`



exception/repOk OSM of HashMap

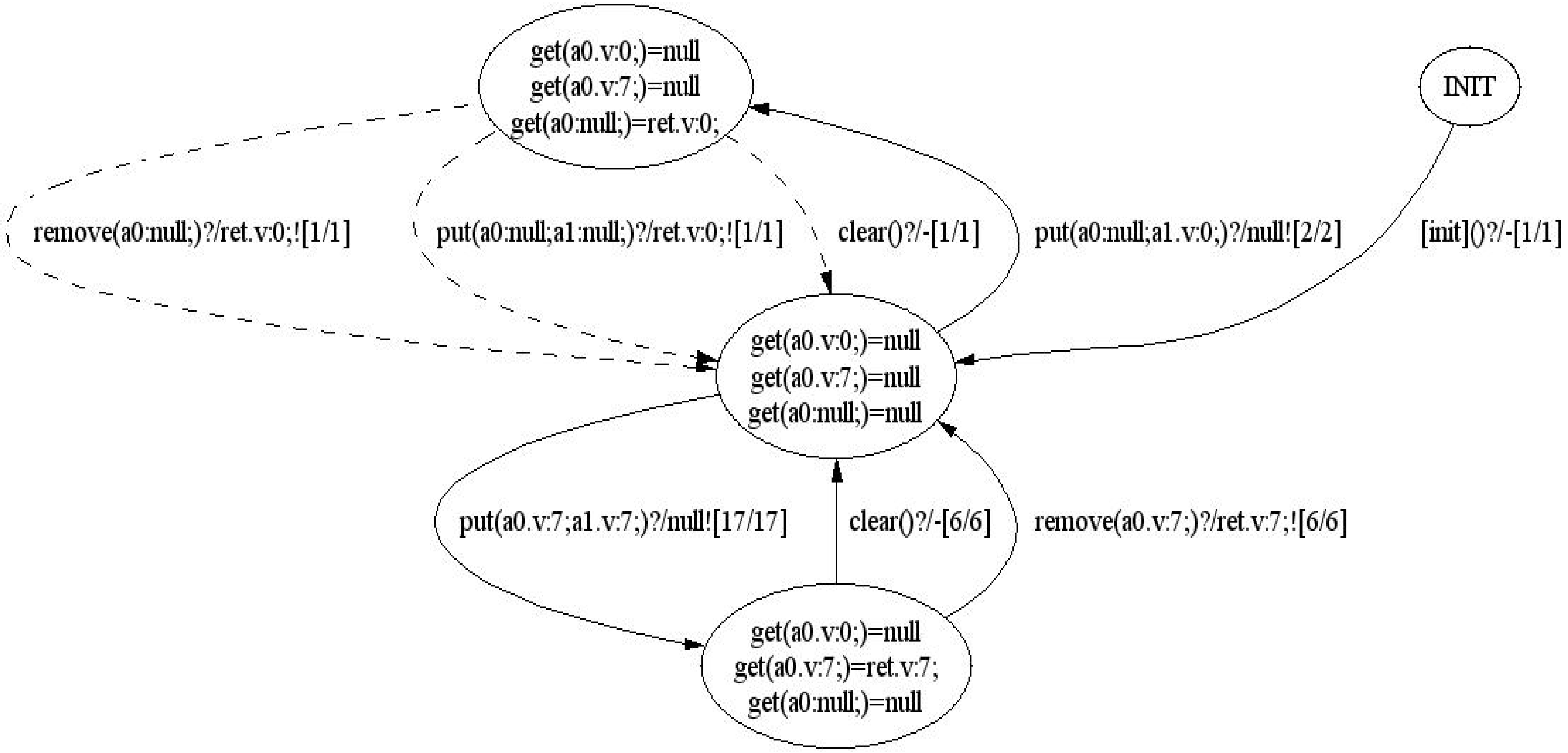



INIT
[init]()?/-[1/1]
repOk()=true
setLoadFactor(a0:7.0;)?/-[17/17]
setLoadFactor(a0:0.0;)?/-[6/6]
ALL_ARGS [23/23]
repOk()=false

- **Illegal input:** `putAll(null)`

•**Class invariant:** `threshold` shall be `(int)(capacity * loadFactor)`.
`setLoadFactor` sets `loadFactor` without updating `threshold`

get OSM of HashMap

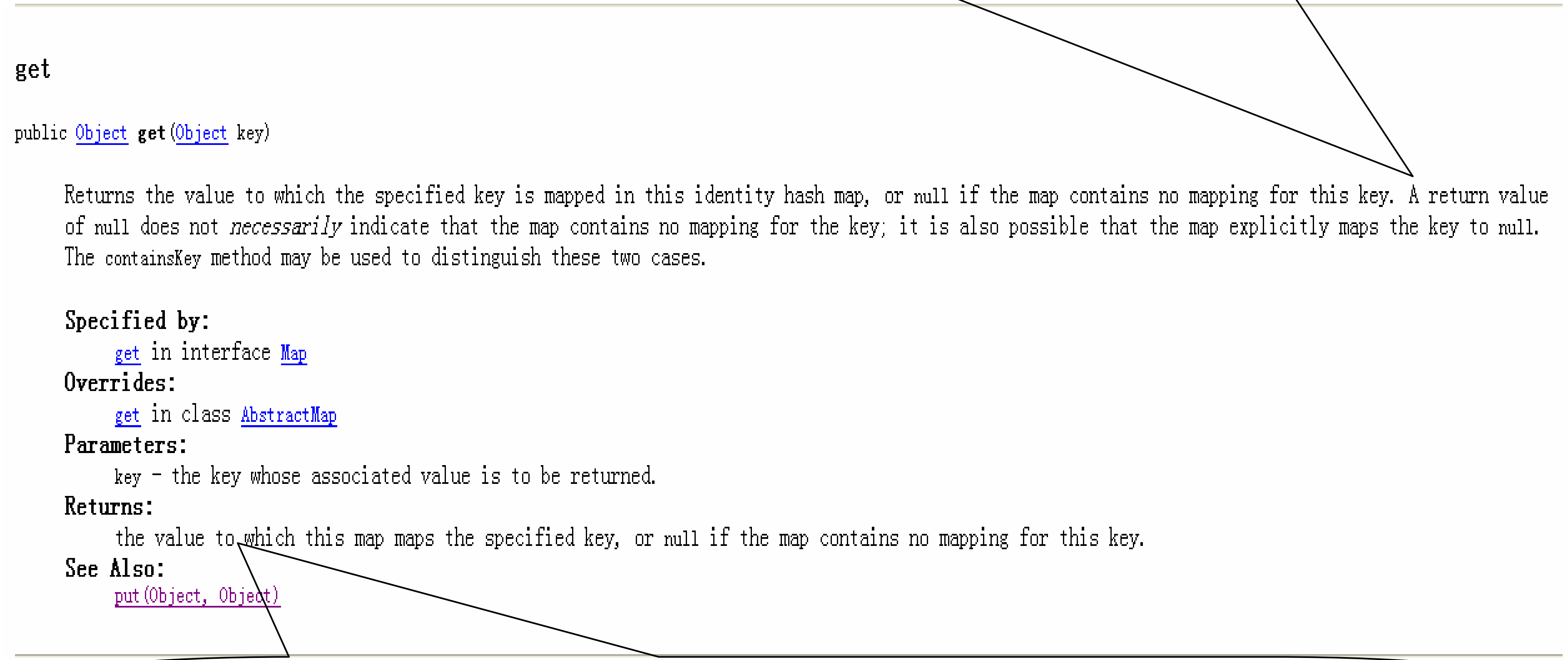


- **Suspicious transition:** `put(a0:null;a1:null;)?/ret.v:0![1/1]`
- **Expose an error in Java API doc for HashMap**

Java API Doc for HashMap

http://java.sun.com/j2se/1.4.2/docs/api/java/util/HashMap.html

- A return value of null does not *necessarily* indicate that the map contains no mapping for the key; it is also possible that the map explicitly maps the key to null.



get

```
public Object get(Object key)
```

Returns the value to which the specified key is mapped in this identity hash map, or `null` if the map contains no mapping for this key. A return value of `null` does not *necessarily* indicate that the map contains no mapping for the key; it is also possible that the map explicitly maps the key to `null`. The `containsKey` method may be used to distinguish these two cases.

Specified by:
`get` in interface `Map`

Overrides:
`get` in class `AbstractMap`

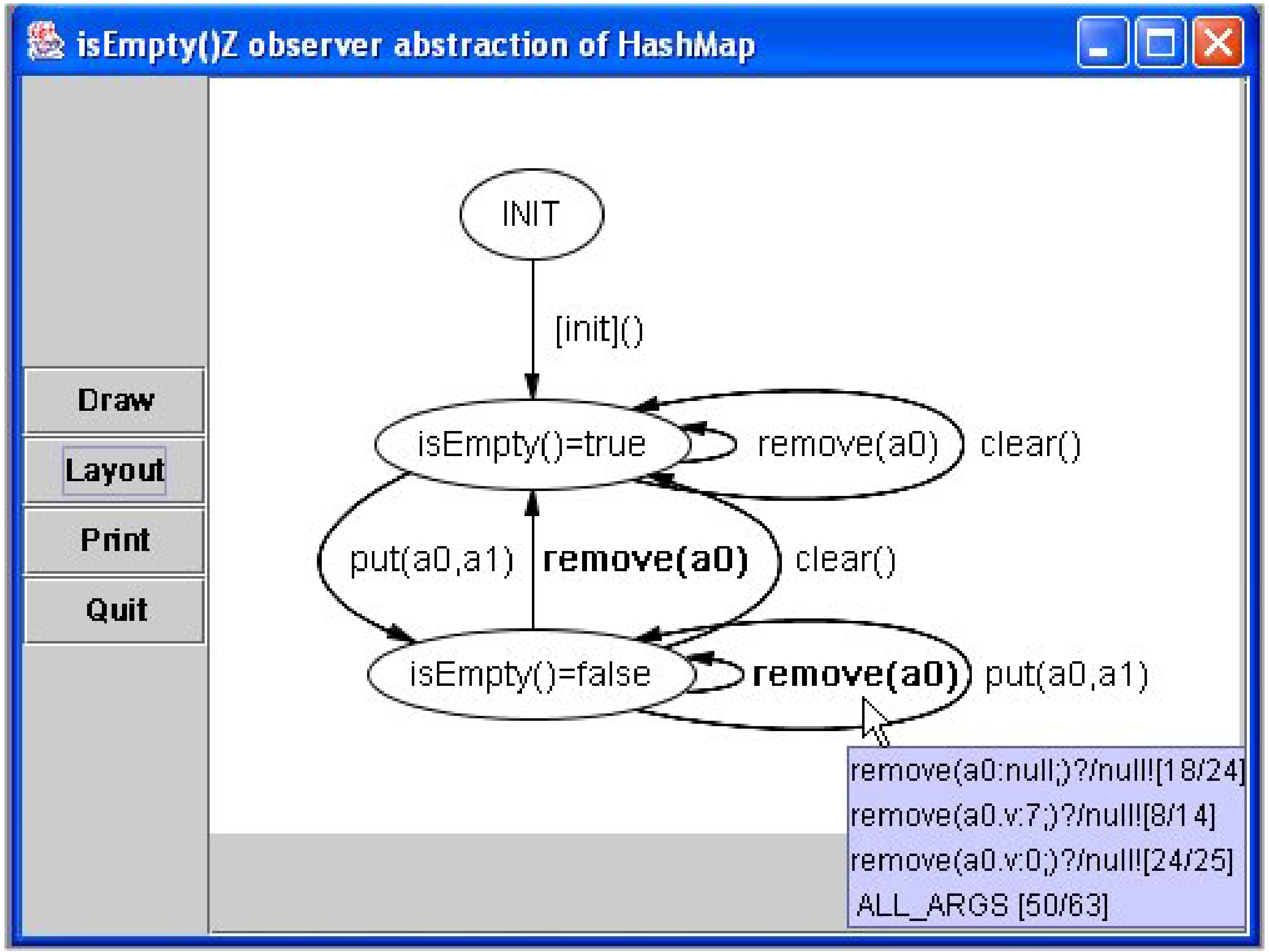
Parameters:
`key` - the key whose associated value is to be returned.

Returns:
the value to which this map maps the specified key, or `null` if the map contains no mapping for this key.

See Also:
`put(Object, Object)`

- **Returns**: the value to which this map maps the specified key, or null if the map contains no mapping for this key.

isEmpty OSM of HashMap



```
remove(a0:null;)?/null![18/24]
remove(a0.v:7;)?/null![8/14]
remove(a0.v:0;)?/null![24/25]
 ALL_ARGS [50/63]
```

- Almost the same as a manually created state machine for a container structure [Nguyen 98]

Lessons

- Extracted observer abstractions help
  - investigate causes of uncaught exceptions
  - identify weakness of an initial test suite
  - find bugs in a class implementation or its documentation
  - understand class behavior

- But some observer abstractions are complex
  three observers of HashMap produce 43 abstract states (e.g., `Collection values()`)
  - user-specified filtering criteria to display a portion of a complex observer abstraction
  - extraction based on a user-specified subset of the initial tests

Related Work

- Sliced OSM extraction [Xie&Notkin SAVCBS 04]
- Daikon [Ernst et al. 01] and algebraic spec discovery [Henkel&Diwan 03]
  - Focus on intra-method or method-pair properties
- Component interface extraction [Whaley et al. 02] and specification mining [Ammons et al. 02]
  - Assume availability of "good" system tests
  - Extract complete graphs from generated unit tests
- Predicate abstraction [Graf&Saidi 97, Ball et al. 00]
  - Returns of predicates are limited to boolean values
  - Focus on program states between program statements
- FSM generation from ASM [Grieskamp et al. 02]
  - Require user-defined indistinguishability properties

Conclusion

- Need tool support to help test inspection
  - too many automatically generated test inputs
- Extract observer abstractions from test executions
  - succinct and useful object-state-transition information for inspection
- Provide some benefits of formal methods without the pain of writing specifications.

Questions?