Automatic Extraction of Sliced Object State Machines for Component Interfaces

Tao Xie David Notkin

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

SAVCBS 04
Oct. 2004

Motivation

- Software components are building blocks of a software system in component-based software development
- Behavior of component interfaces needs to be understood

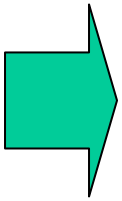
- However, behavioral specifications for component interfaces are often not written down

Synopsis

- Dynamically extract sliced object state machines for component interfaces
  - Component: Java class
  - Component interface: public methods
  - Focus on object-state transitions

- Succinct and useful for understanding how method executions affect object states
    - Component understanding, test inspection, etc.

Outline



- Motivation
- Object State Machine (OSM)
- Sliced Object State Machine
- Discussion
- Related Work
- Conclusion

Object State Machine (OSM)

M = (I, O, S, δ, λ, $INIT$) 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, δ, λ, $INIT$) 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

States can be abstract or concrete

Concrete-Object State Rep

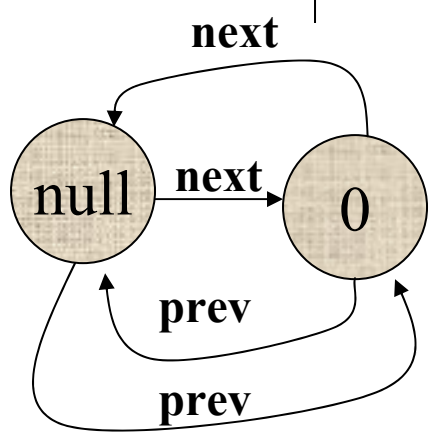
- Rostra includes five techniques for state representation [Xie, Marinov, and Notkin ASE 04]
- WholeState technique
  - Traversal: collect the values of all the fields transitively reachable from the object
  - Linearization: remove reference addresses but keep reference relationship
- State comparison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

Concrete-Object State Rep

Test 1:

```
MyInput t0 = new MyInput(0);
LinkedList THIS = new LinkedList();
boolean RETVAL = THIS.add(t0);
```

```
size=1;
modCount=1;
serialVersionUID=876323262645176354;
header.element=null;
header.next.element.v=0;
header.next.next=header;
header.next.previous=header;
header.previous=header.next;
```

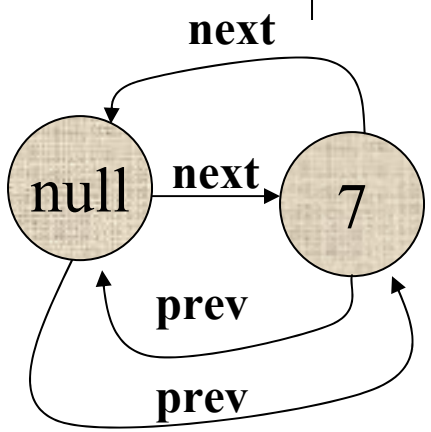


Concrete-Object State Rep

Test 2:

```
MyInput t0 = new MyInput(7);
LinkedList THIS = new LinkedList();
boolean RETVAL = THIS.add(t0);
```

```
size=1;
modCount=1;
serialVersionUID=876323262645176354;
header.element=null;
header.next.element.v=7;
header.next.next=header;
header.next.previous=header;
header.previous=header.next;
```

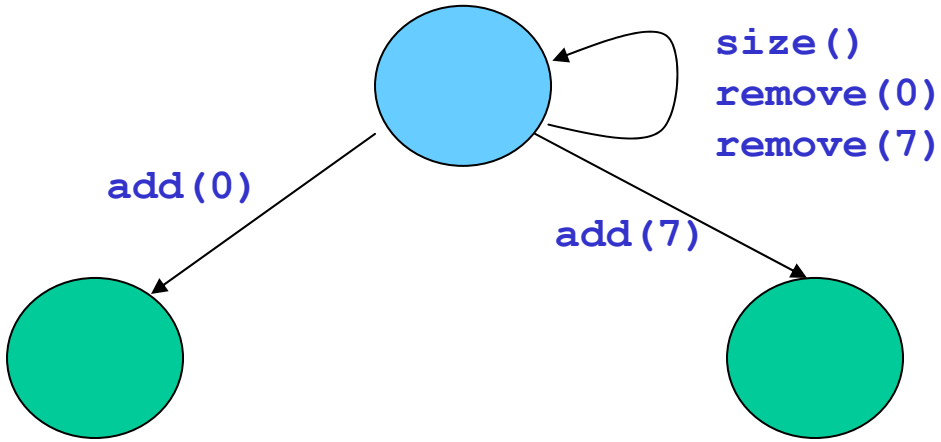


Concrete-Object State Exploration -Rostra Test Generation

Method args: **`add(0), add(7), remove(0), remove(7), size()`**

after **`new LinkedList()`**

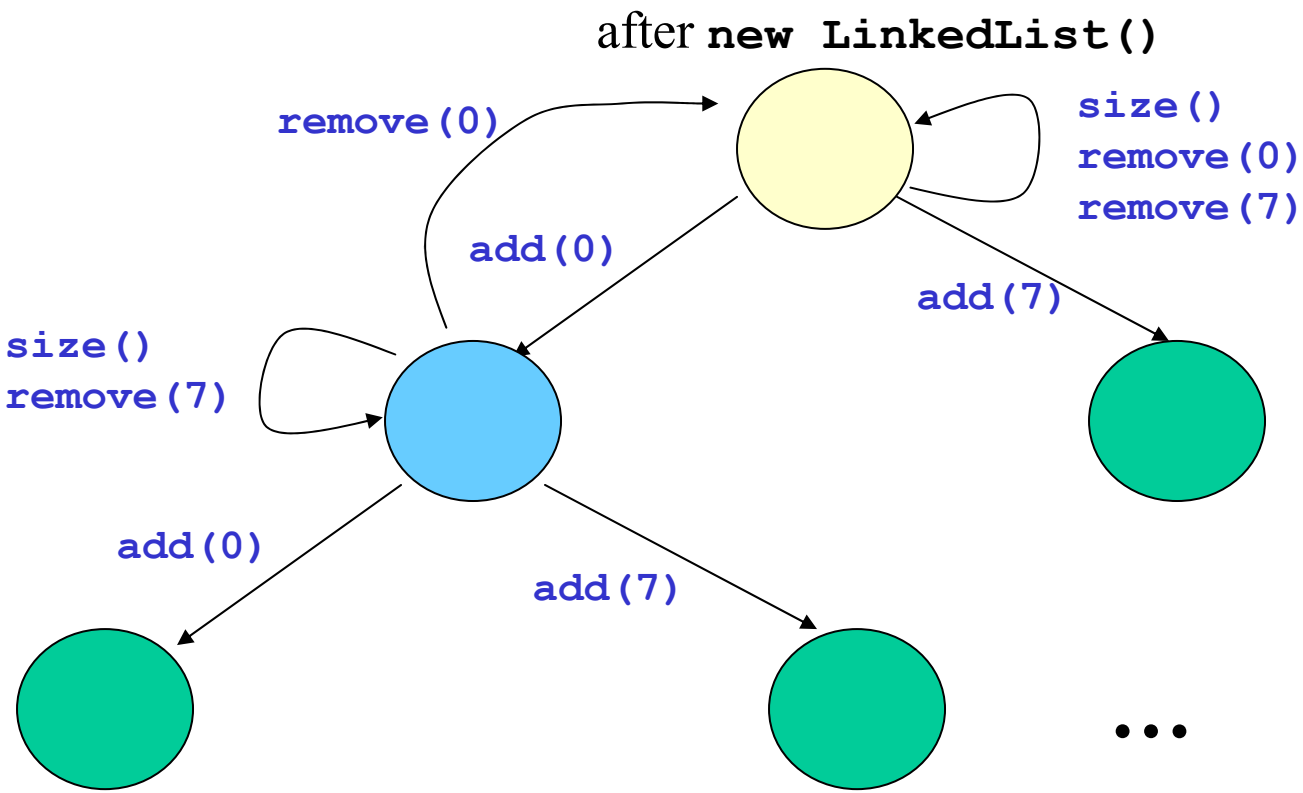
Parasoft Jtest 5.1



The 1st Iteration

Concrete-Object State Exploration -Rostra Test Generation

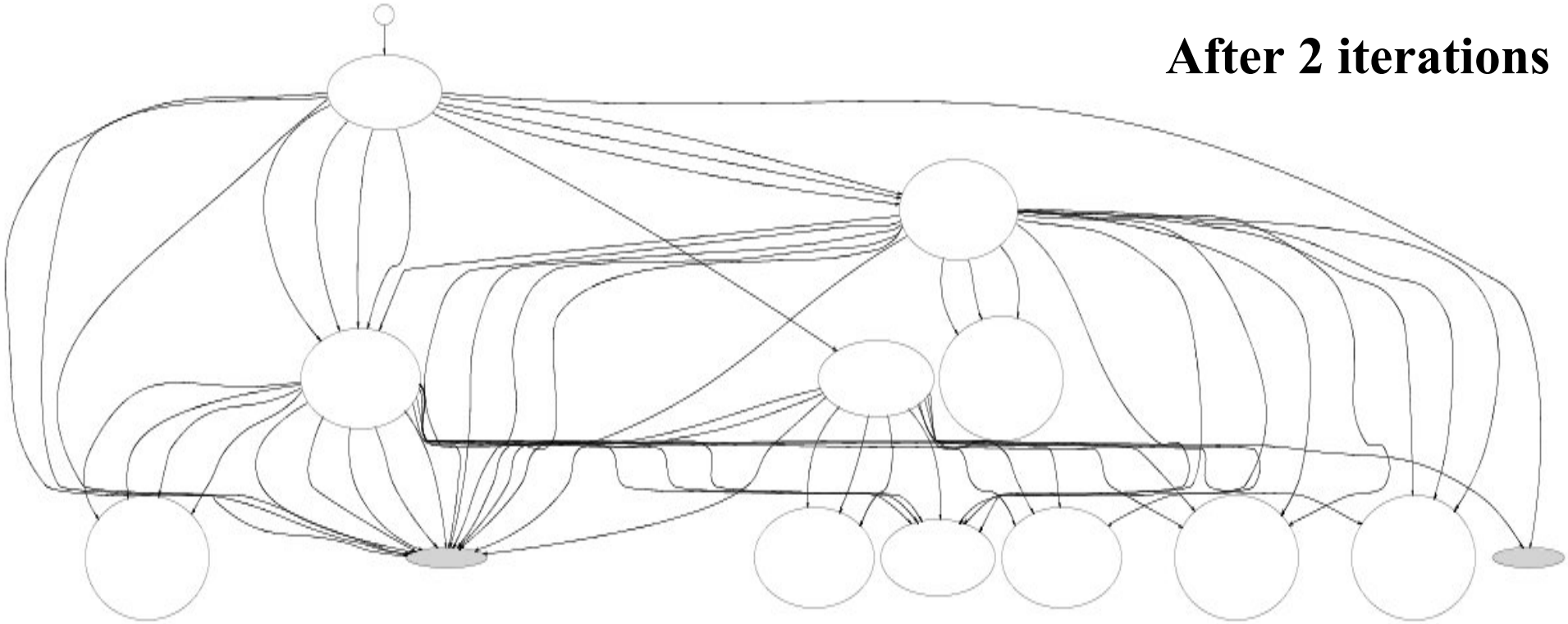
Method args: `add(0)`, `add(7)`, `remove(0)`, `remove(7)`, `size()`



The 2nd Iteration

Example of *LinkedList* Concrete OSM

After 2 iterations

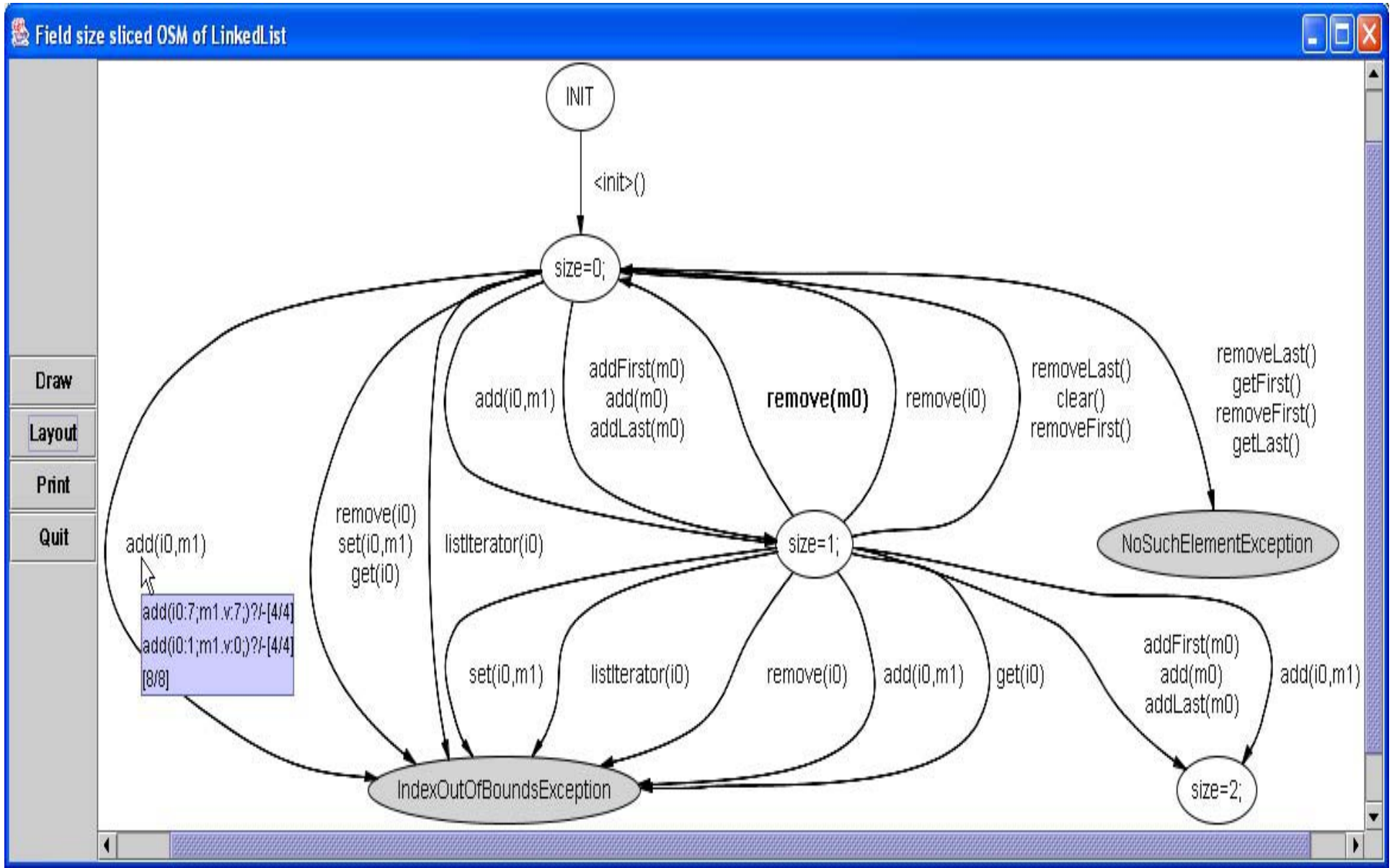


Too complex to learn useful behavior

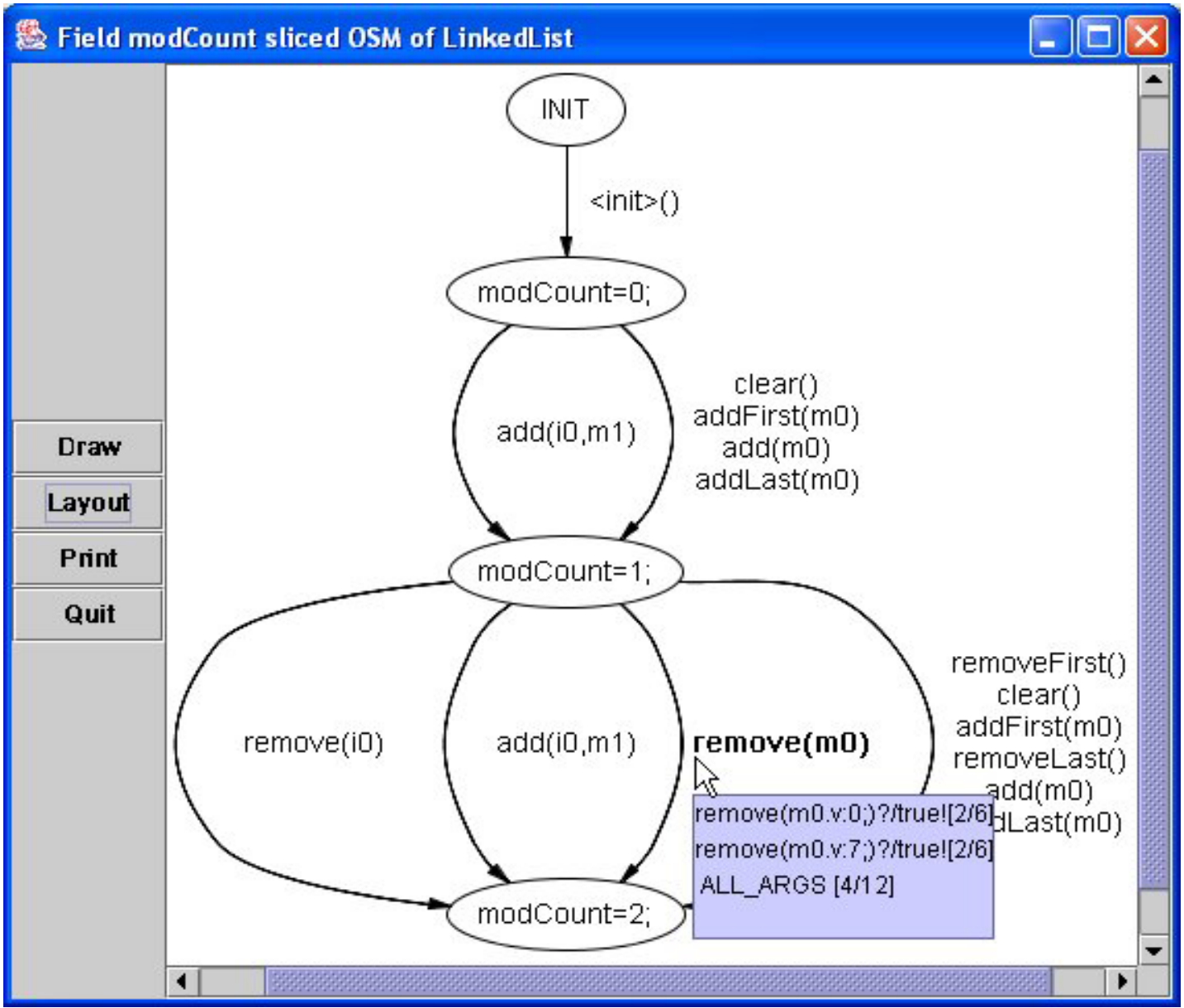
State Slicing by Fields

- The extracted concrete OSM is too complex to be useful — we need to reduce the size
- Slice a concrete state by fields
  - Inspired by Whaley et al. [Whaley et al. 02]
  - Project a concrete state to a specific field
- Construct sliced OSM's
- Extract multiple OSM's instead of one single OSM

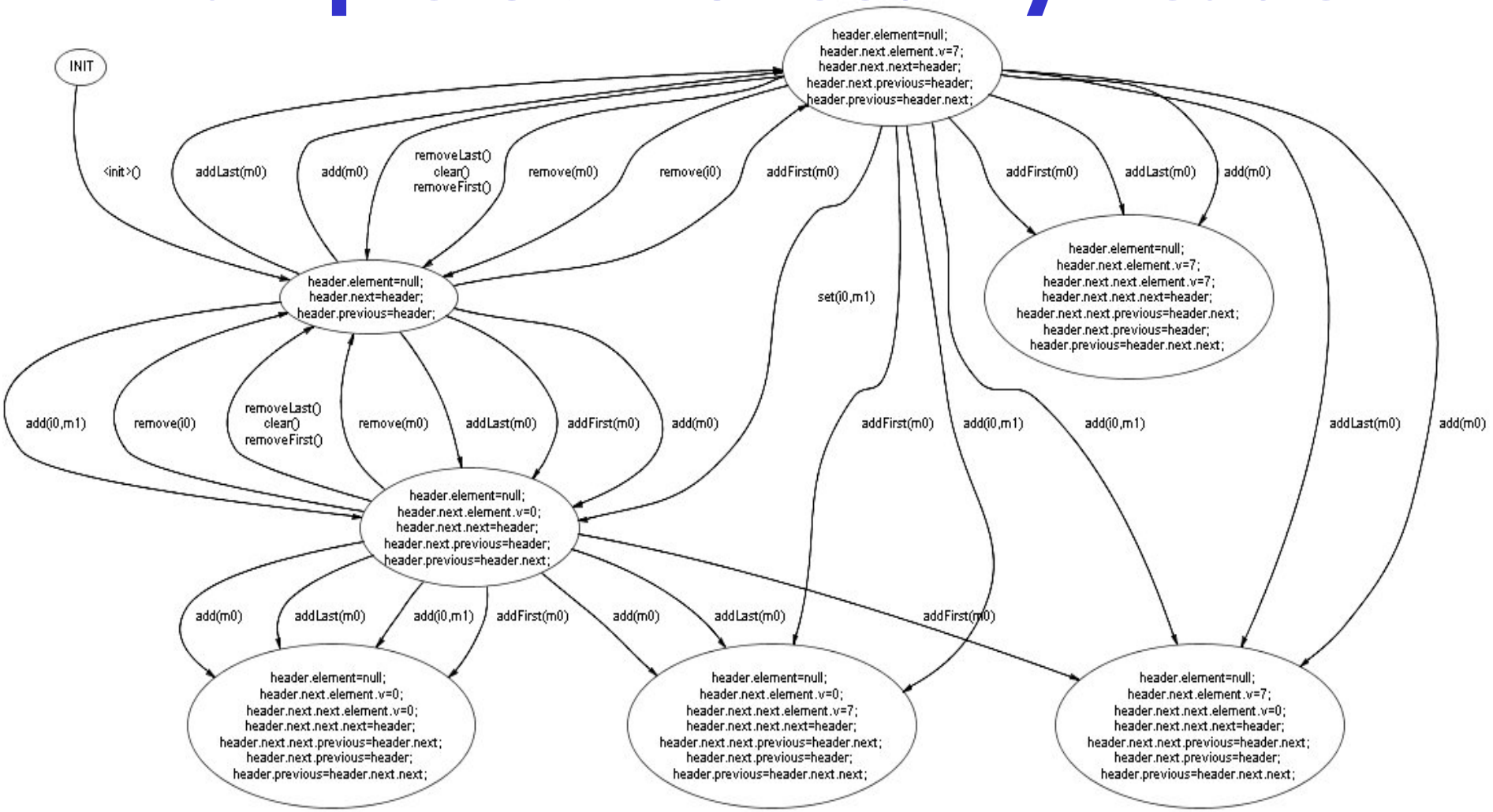
Example OSM sliced by *size*



Example OSM sliced by *modCount*



Example OSM sliced by *header*



- ***header*: sentinel node leading to key content of the linked list**
- **sliced states include fields reachable from *header***

Structural Abstraction

- Inspired by Korat [Boyapati et al. 02]

Test 1:

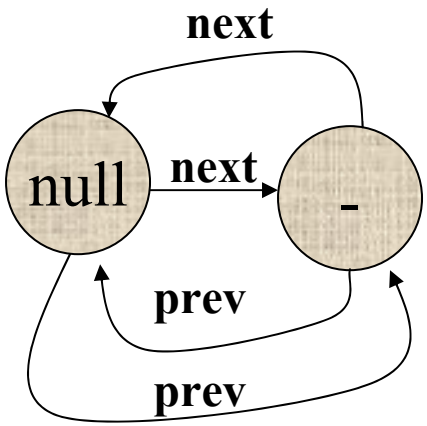
```
MyInput t0 = new MyInput(0);
LinkedList THIS = new LinkedList();
boolean RETVAL = THIS.add(t0);
```

```
size=1;
modCount=1;
serialVersionUID=876323262645176354;
header.element=null;
header.next.element.v=0;
header.next.next=header;
header.next.previous=header;
header.previous=header.next;
```

Test 2:

```
MyInput t0 = new MyInput(7);
LinkedList THIS = new LinkedList();
boolean RETVAL = THIS.add(t0);
```

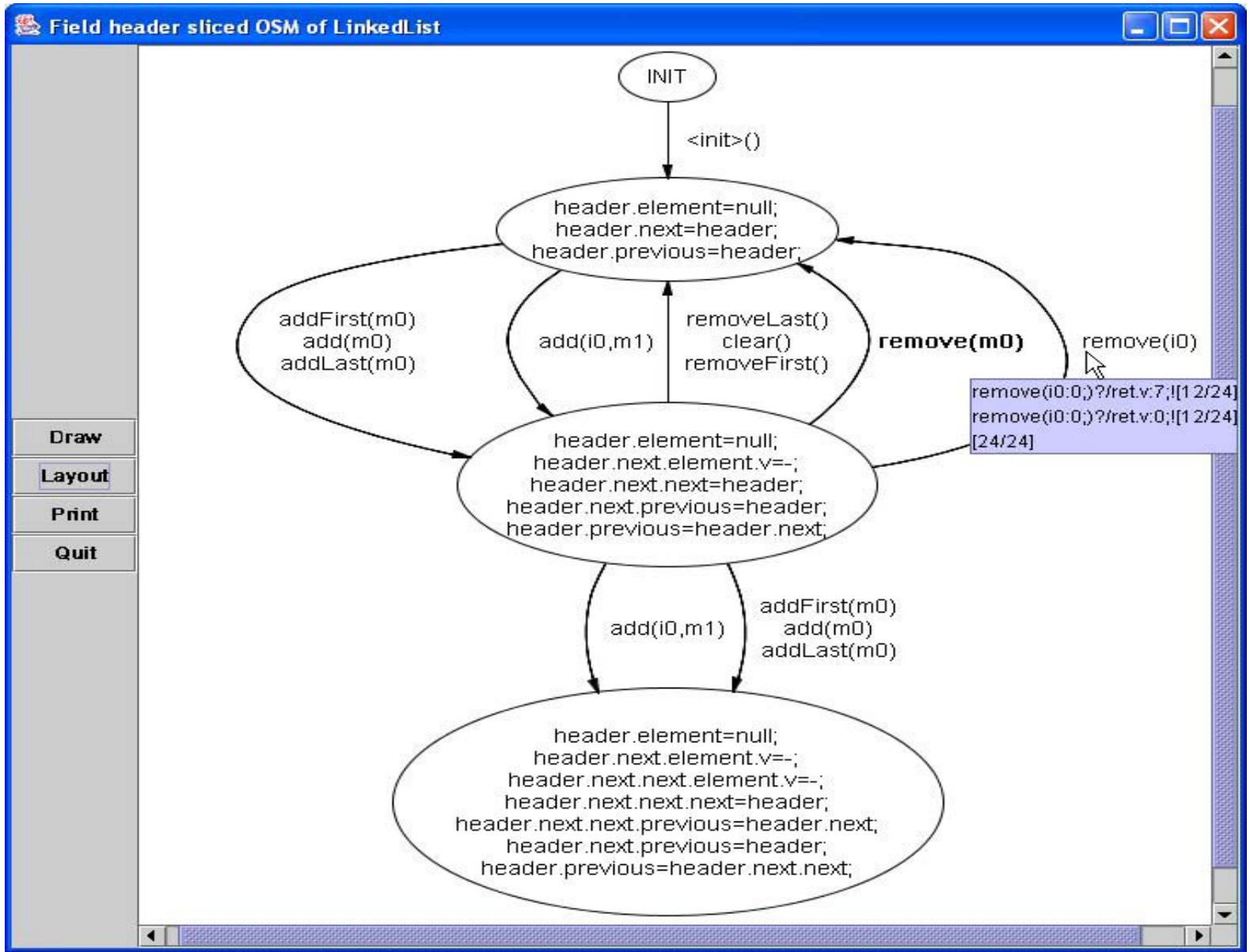
```
size=1;
modCount=1;
serialVersionUID=876323262645176354;
header.element=null;
header.next.element.v=7;
header.next.next=header;
header.next.previous=header;
header.previous=header.next;
```




```
size=1;
modCount=1;
serialVersionUID=876323262645176354;
header.element=null;
header.next.element.v=-;
header.next.next=header;
header.next.previous=header;
header.previous=header.next;
```

Object graphs share the same shape

Example OSM sliced by *header* after structural abstraction



Outline

- Motivation
- Object State Machine (OSM)
- Sliced Object State Machine
- Discussion
- Related Work
- Conclusion

Member Fields

- Member fields as abstraction functions
- Over-abstract: coupled member fields
  - Projection on multiple fields
  - Concept analysis to group fields [Dekel&Gil 03]
- Under-abstract: complex member field
  - Human inputs for better abstraction functions [Grieskamp et al. 02]
  - Better than requiring human inputs upfront

Generated Tests

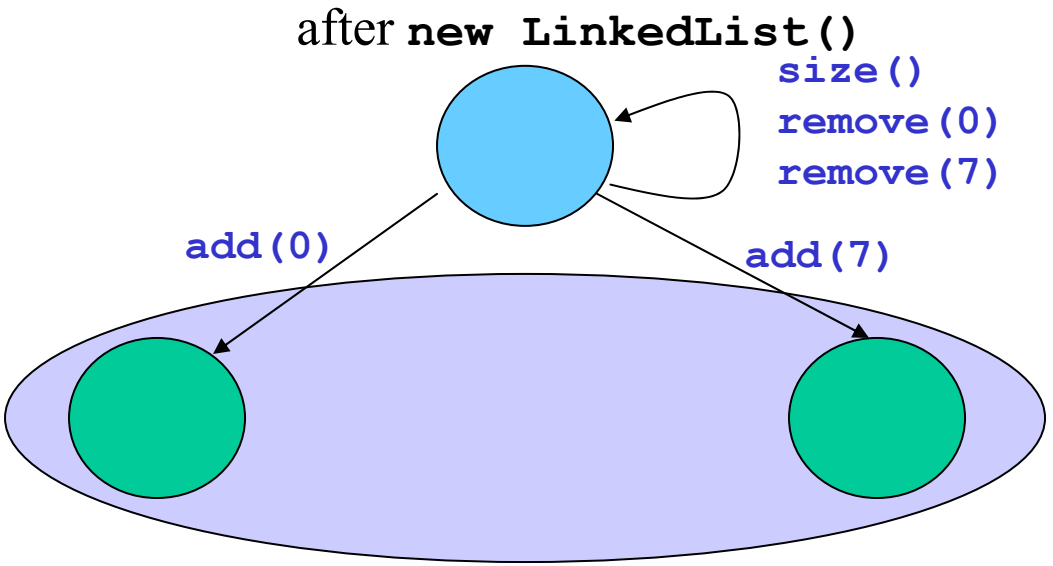
- Increase #method arguments
  - OSM's sliced by *size, modCount* remain the same (edge details grow)
  - OSM's sliced by *header* grow rapidly (after structural abstraction, remain the same)
- Increase #iteration
  - OSM's sliced by *size*, *modCount*, *header* (after structural abstraction) grow linearly
- To manage the complexity, the user can configure to use fewer arguments or iterations
  - But might miss some interesting cases

Other Potential Applications

- FSM-guided test generation [Lee&Yannakakis 96]
- Feedback loop [Xie&Notkin FATES 03]
  - Test generation & OSM extraction
  - Deviation-based test selection [Xie&Notkin ASE 03]
- "Conformance" testing
  - Inspect and confirm extracted OSM's
  - Extrapolate extracted OSM's to predict unobserved behavior
  - Generate more tests and check against predicted behavior

Sliced-State Exploration

Method args: `add(0)`, `add(7)`, `remove(0)`, `remove(7)`, `size()`



State sliced by *size*, *modCount*, or *header*

- **Heuristics to guide test generation or model checking**
- **More investigations are needed**

Related Work

- Dynamically extract observer abstractions [Xie&Notkin ICFEM 04]
  - Abstraction functions: returns of observers
  - Capture behavior of observer returns
  - Require the availability of "good" observers
  - Sometimes too many observers for an interface (18 observers for *LinkedList*)
  - Two approaches are complementary

Related Work (cont.)

- Whaley et al. [Whaley et al. 02]
  - Abstraction functions: immediately preceding state-modifying method
- Ammons et al. [Ammons et al. 02]
  - Sequence order among method calls

Both

- Assume availability of "good" system tests
- Extract complete graphs from generated unit tests

Related Work (cont.)

- Bandera [Corbett et al. 00]
  - Slice control points, variables, and data structures w.r.t. a given property
- AsmLT [Grieskamp et al. 02]
  - Abstraction function: user-defined indistinguishability properties
- Statically extract object state model [Kung et al. 94]
  - Abstraction function: value intervals related to path conditions
  - *LinkedList*
    - No value intervals for *header*
    - Only (*size* == 0) path condition for *size*

Conclusion

- Lack of specs for a component interface poses a barrier to component reuse
- Extract sliced OSM's to capture the object-state-transition information
- Sliced OSM's are often succinct and useful for inspection
- Sliced OSM's have other potential applications in testing and verification

Questions?

Generated Tests (cont.)

- Poor-quality tests → Poor-quality OSM's
  - Lack sufficient arguments
  - Lack sufficient iterations
- Static analysis can help identify some insufficient cases
  - `addAll(int index, Collection c)` identified to be state-preserving
- Inspection of OSM's can also help