Falling Back on Executable Specifications

Hesam Samimi
Ei Darli Aung
Todd Millstein

University of California, Los Angeles
{hesam,eidarli,todd}@cs.ucla.edu

Specifications for Reliability

```
class LinkedList {
  void sort() {

              < Implementation >


  }
}
```

Oops! Failure / Subtle bug!

Specifications for Reliability

```
class LinkedList ensures isAcyclic()
  void sort() ensures
  isPermutationOf(old) && isSorted()
```

specification

```
              < Implementation >

  }
}
```

Static Verification

development

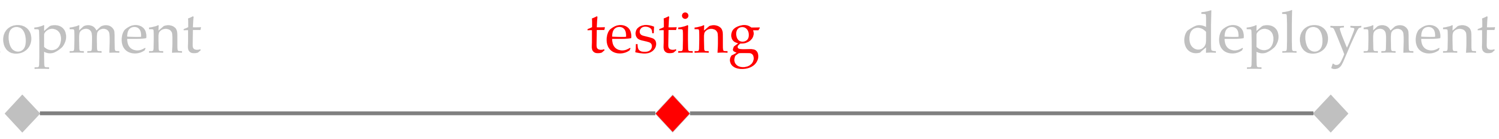
testing

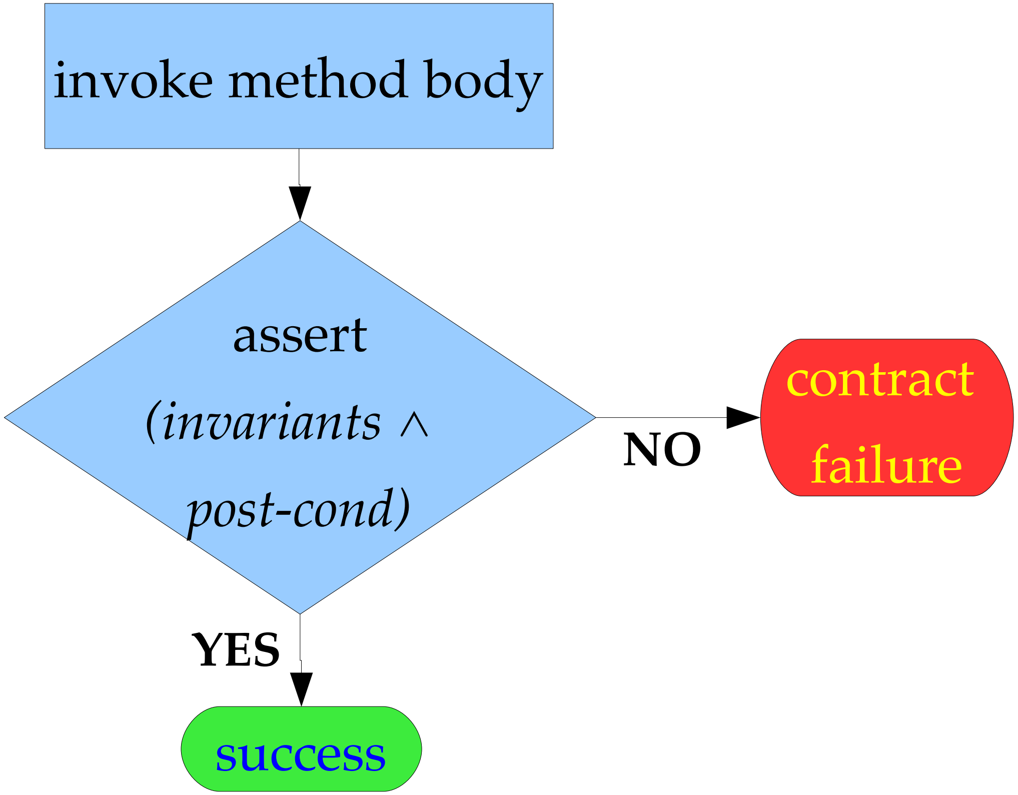
deployment

```
class LinkedList ensures isAcyclic()
  void sort() ensures
  isPermutationOf(old) && isSorted()


             < Implementation >


  }
}
```

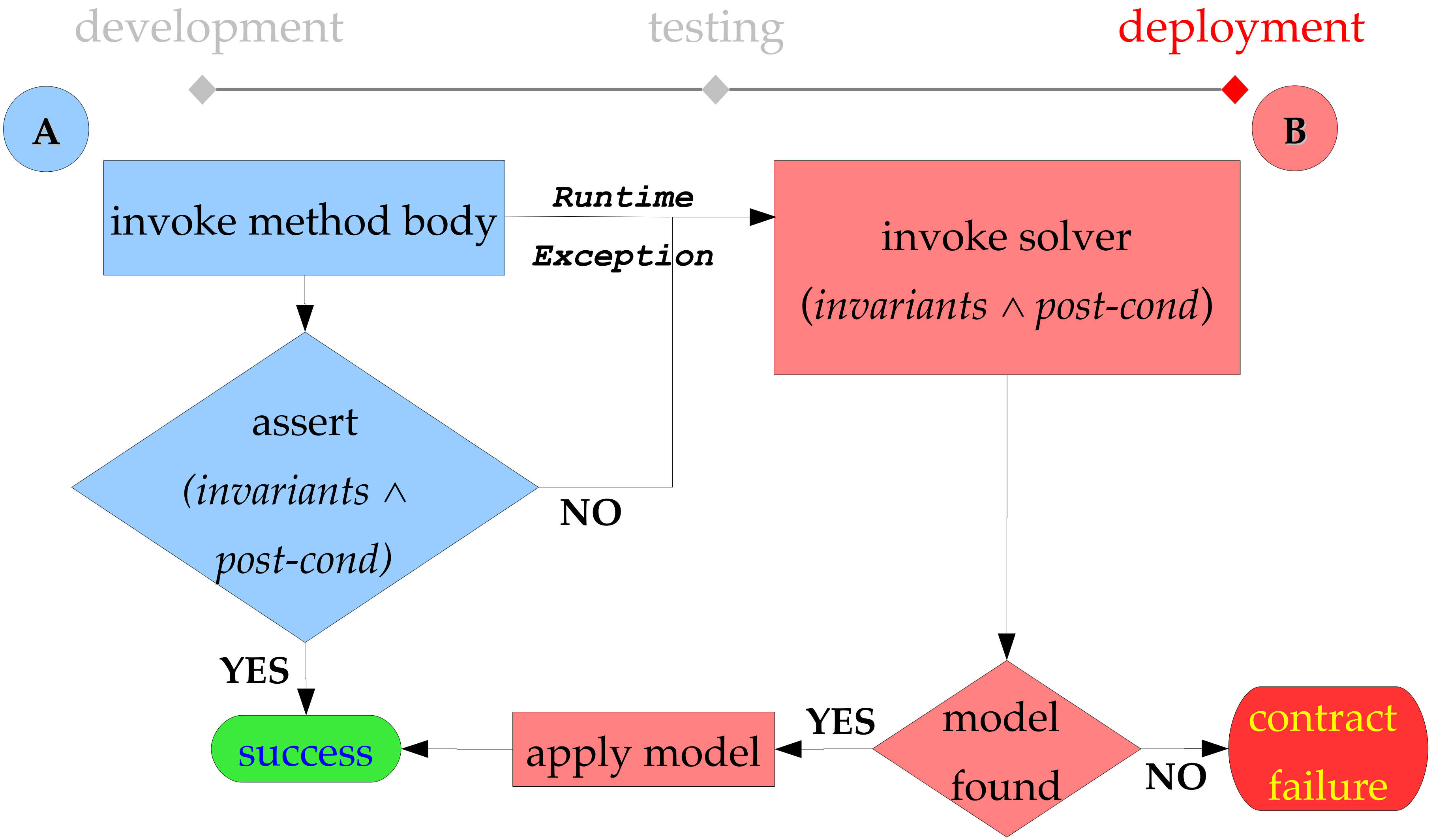
specification

Contract Checking





Plan B Fallback



UCLA COMPUTER SCIENCE DEPARTMENT

Plan B

- Idea
  - specs not only to validate, but to run as slower/reliable alternatives to failing implementations
  - use a constraint solver to find a model
    - satisfying specs non-deterministically

Plan B

- Idea
  - specs not only to validate, but to run as slower/reliable alternatives to failing implementations
  - use a constraint solver to find a model
    - satisfying specs non-deterministically
- Benefit
  - handles arbitrary errors, Runtime Exceptions

Plan B

- Idea
  - specs not only to validate, but to run as slower/reliable alternatives to failing implementations
  - use a constraint solver to find a model
    - satisfying specs non-deterministically
- Benefit
  - handles arbitrary errors, Runtime Exceptions
  - intentional fallback (declarative programming) for complex tasks

Demo

LinkedList sort

Demo

Data Structure Repair

- *[Demsky/Rinard '03]*   *[Elkarablieh/Khurshid '07]*
- ensures method does not violate data integrity constraints
- no guarantee to retain method functionality
- patch final state and continue execution
  - local search
- relies on implementation to be mostly correct
  - some data loss for regaining integrity

Plan B

- ensures method post condition is satisfied
  - while keeping integrity constraints (invariants)
- starts fresh
  - SAT-based constraint solving
- no dependency on implementation
  - full functional recovery

Contributions

- Plan B: Fallback for method recovery
- PBnJ: Extension of Java
  - Specifications
    - first order relational logic Alloy [*Jackson '02*]
  - Implementation
    - Kodkod [*Torlak '09*]
- Making fallback practical
- Experience

Contributions

- Plan B: Fallback for method recovery
- PBnJ: Extension of Java
  - Specifications
    - first order relational logic Alloy [*Jackson '02*]
  - Implementation
    - Kodkod [*Torlak '09*]
- Making fallback practical
- Experience

Specifications in PBnJ

```
class Node { int value; Node next; }
class LinkedList ensures isAcyclic() {
  Node head;









}
```

Specifications in PBnJ

```
class Node { int value; Node next; }
class LinkedList ensures isAcyclic() {
  Node head;
  spec Set<Node> nodes() { return head.*next; }





}
```

reflexive

transitive closure

Specifications in PBnJ

```
class Node { int value; Node next; }
class LinkedList ensures isAcyclic() {
  Node head;
  spec Set<Node> nodes() { return head.*next; }
  spec boolean isAcyclic() {
    return head == null ||
           some Node n : nodes() | n.next == null;
  }
}
```

existential quantification

Contributions

- Plan B: Fallback for method recovery
- PBnJ: Extension of Java
  - Specifications
    - first order relational logic Alloy [*Jackson '02*]
  - Implementation
    - Kodkod [*Torlak '09*]
- Making fallback practical
- Experience

Implementation

- Kodkod [*Torlak '09*]
  - bounded, relational SAT-based constraint solver

Implementation

- Kodkod [*Torlak '09*]
  - bounded, relational SAT-based constraint solver
- Relational
  - program states as relations, specs as relational op's
    - classes as unary relations
      - set of instances
    - fields as binary relations
      - [object, value] tuples

Implementation

- Kodkod [*Torlak '09*]
  - bounded, relational SAT-based constraint solver

- Bounded
  - requires bounds per relation
    - search space for each variable
    - spec unsataisfiable:
      - no solution within bounds (contract failure)
      - may miss solution outside bounds

Contributions

- Plan B: Fallback for method recovery
- PBnJ: Extension of Java
  - Specifications
    - first order relational logic Alloy [*Jackson '02*]
  - Implementation
    - Kodkod [*Torlak '09*]
- Making fallback practical
- Experience

Making Fallback Practical

- Problem: search space enormous
  - `LinkedList sort()` with 20 elements
    - space size ~ 10^{220}

Making Fallback Practical

- Problem: search space enormous
    - `LinkedList sort()` with 20 elements
        - space size ~ 10^{220}

- Approach: domain specific knowledge as annotations
    - such as “modifies clauses”
        - disallow spurious solutions
        - reduce space, improve solving efficiency

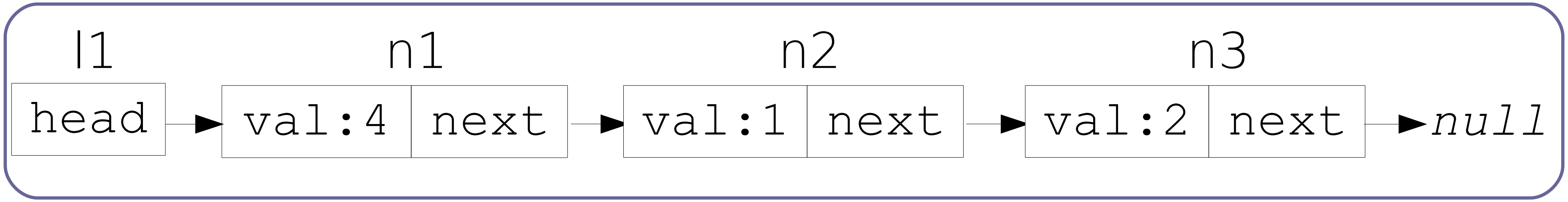
Modifies Clauses

- by default fallback is allowed to modify any field in specs

Modifies Clauses

- by default fallback is allowed to modify any field in specs
- limit modifiable fields:

```
void sort()
modifies fields LinkedList.head, Node.next {…}
```



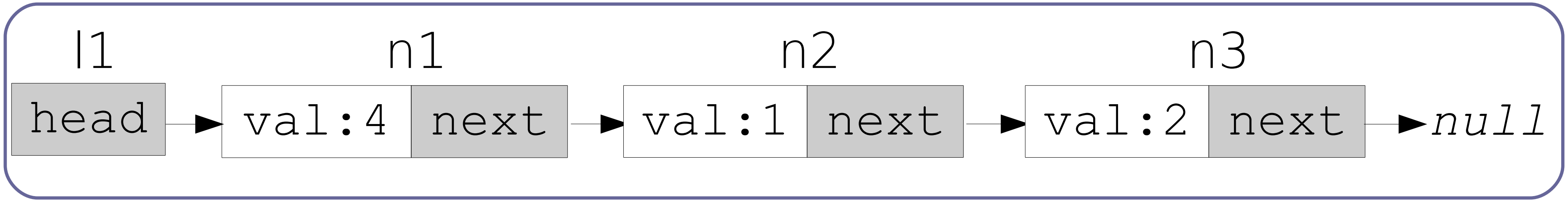
l1 head → n1 val:4 next → n2 val:1 next → n3 val:2 next → *null*

Modifies Clauses

- by default fallback is allowed to modify any field in specs

- limit modifiable fields:

```
void sort()
modifies fields LinkedList.head, Node.next {…}
```



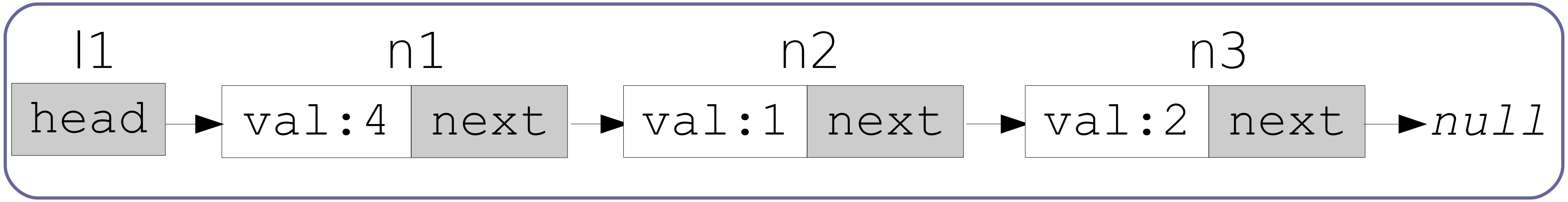
l1 n1 n2 n3

head → val:4 next → val:1 next → val:2 next → *null*

Modifies Clauses

- by default fallback is allowed to modify any field in specs
- limit modifiable fields:

```
void sort()
modifies fields LinkedList.head, Node.next {…}
```



- `LinkedList sort()` with 20 elements
  - space size ~ $\cancel{10^{220}}$ 10^{27}
  - demo fallback time ~ 4 sec.

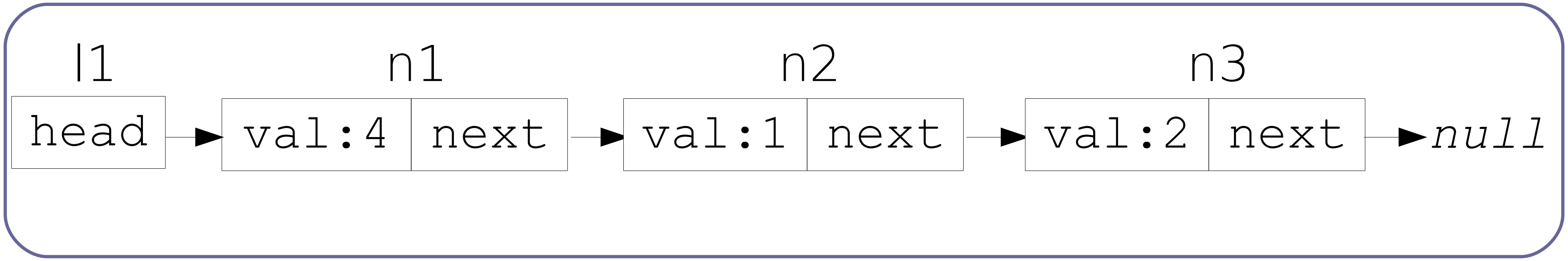
Modifies Clauses

- by default fallback is allowed to modify any reachable object

Modifies Clauses

- by default fallback is allowed to modify any reachable object
- limit modifiable objects:

```
  void add(Node n)
  modifies objects head == null ? this : tail() {
     ...
  }
```

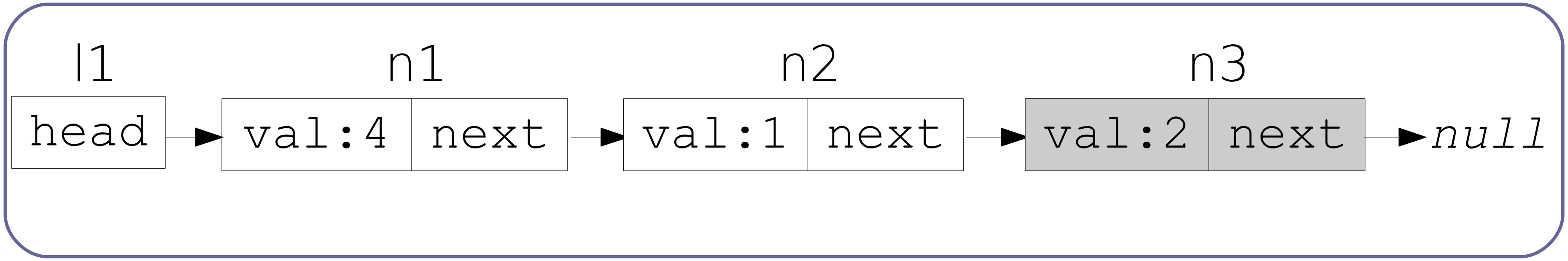


Modifies Clauses

- by default fallback is allowed to modify any reachable object
- limit modifiable objects:

```
  void add(Node n)
  modifies objects head == null ? this : tail() {
     ...
  }
```

eval → {n3}



l1

head

n1

val:4 | next

n2

val:1 | next

n3

val:2 | next

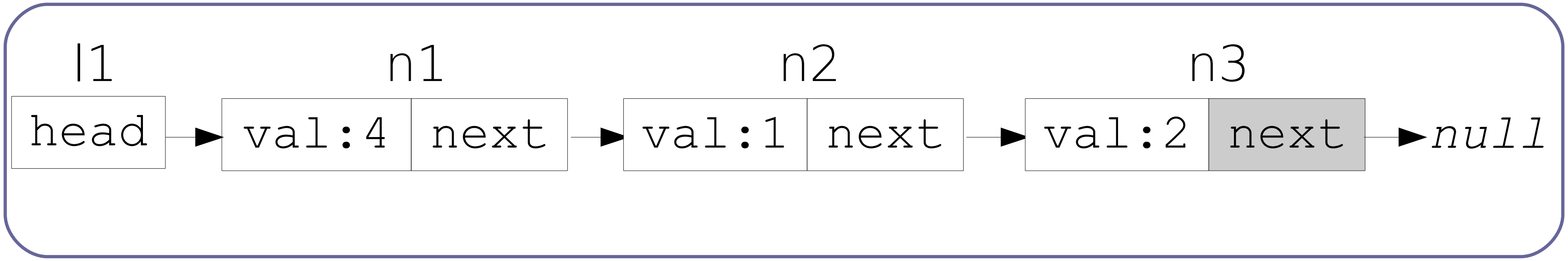
null

Modifies Clauses

- by default fallback is allowed to modify any reachable object
- limit modifiable objects:

```
void add(Node n)
modifies objects head == null ? this : tail() {
  ...
}
```

eval → {n3}



Contributions

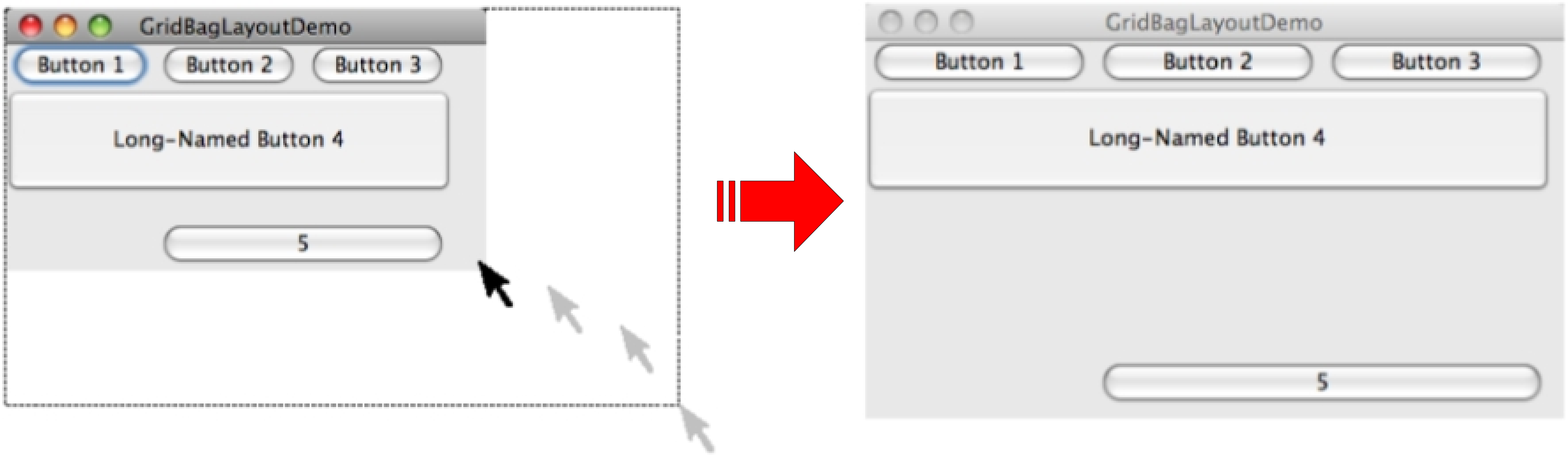
- Plan B: Fallback for method recovery
- PBnJ: Extension of Java
  - Specifications
    - first order relational logic Alloy [*Jackson '02*]
  - Implementation
    - Kodkod [*Torlak '09*]
- Making fallback practical
- Experience

Experience

- Stress Tests on binary trees
  - `Insert` operation
    - complex specs
    - modifies clauses
    - 200 nodes
    - Binary Search tree
      - 4 sec.
    - Red Black tree
      - 21 sec.
  - Kodkod's encoding step, not SAT-solving bottleneck

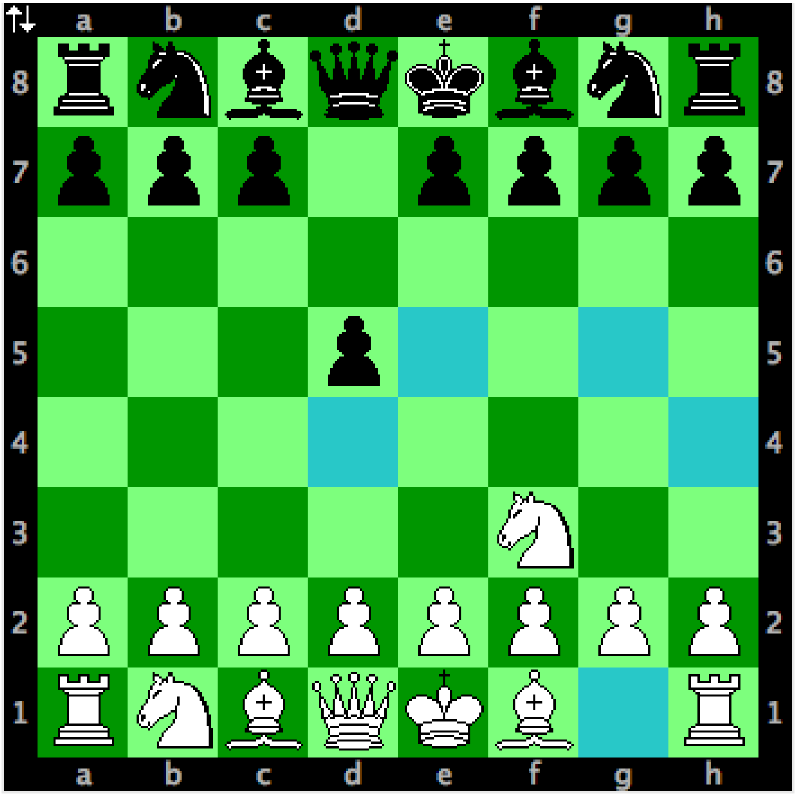
Experience

- Existing Software
  - expressiveness, ease of deployment, efficiency
  - `java.awt.GridBagLayout` Java layout manager



Experience

- Existing Software
  - JChessBoard Chess
    - valid moves



Related Work

- Executing Specifications via Constraint Solving
  - Specification Statement [*Morgan '88*]
  - jmle: Executable JML [*Krause/Wahls '06*]
  - Kaleidoscope [*Freeman-Benson/Borning '92*] Mixed Interpreter [*Rayside '09*]

Related Work

- Executing Specifications via Constraint Solving
  - Specification Statement [*Morgan '88*]
  - jmle: Executable JML [*Krause/Wahls '06*]
  - Kaleidoscope [*Freeman-Benson/Borning '92*] Mixed Interpreter [*Rayside '09*]
- Data Structure Repair [*Demsky/Rinard '03 '05 '09*]
  - Assertion-based Repair [*Elkarablieh/Khurshid '07 '08*]

Related Work

- Executing Specifications via Constraint Solving
  - Specification Statement [*Morgan '88*]
  - jmle: Executable JML [*Krause/Wahls '06*]
  - Kaleidoscope [*Freeman-Benson/Borning '92*] Mixed Interpreter [*Rayside '09*]
- Data Structure Repair [*Demsky/Rinard '03 '05 '09*]
  - Assertion-based Repair [*Elkarablieh/Khurshid '07 '08*]
- "Contract-based Data Structure Repair Using Alloy" [*Nokhbeh Zaeem/Khurshid '10*]
  - repair-oriented: iterative / heuristic instead of fallback-oriented: "modifies" annotations

Future Directions

- Other solvers
  - Kodkod with local search, cost optimizing
  - SMT vs. Relational solver

Future Directions

- Other solvers
  - Kodkod with local search, cost optimizing
  - SMT vs. Relational solver

- Aiding offline debugging
  - unreasonable to run Plan B next time on same error trace
    - error proof helps bug localization
    - can model from Plan B help in fixing bugs?

Conclusions

Plan B a practical use of executable specs:

- Static verification, synthesis major advances
  - unlikely to replace online repair and debugging soon
- Online SAT solving reasonable for failed/crashing case
- Declarative code within imperative on complex tasks
- Easy to enable existing software

Thank You!

http://www.cs.ucla.edu/~hesam/planb