Automated Testing & Verification

Automatic Test Generation

Galeotti/Gorla/Rau
Saarland University

Testing & Debugging

- Testing : Find inputs that make the program fail

- Debugging : The process of finding the cause of a failure.

Test Case

- Test Case Values/Test Input/Test Data : Values that satisfy a test requirement

- Expected output: The result of the test case if the software under test behaves as expected.
  - Test Oracle

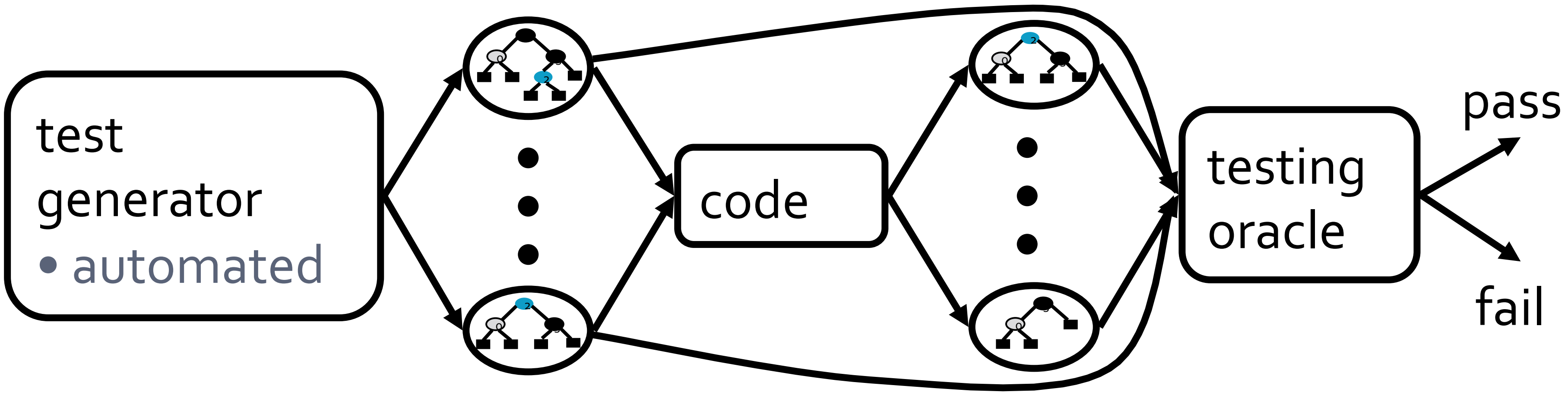
Adequacy criteria

- Goals of a Test Suite
  - All instructions
  - All conditions (decisions)
  - All paths
  - All inputs
- Goals might not be feasible
  - Example: dead code
  - Detection of unfeasible goals is undecidable in general
- 100% goal coverage might not be possible

Our ideal

- To generate automatically tests
  - Satisfying adequacy criteria
  - Avoid writting the tests ourselves
  - Find bugs faster
- Update tests automatically if the software evolves

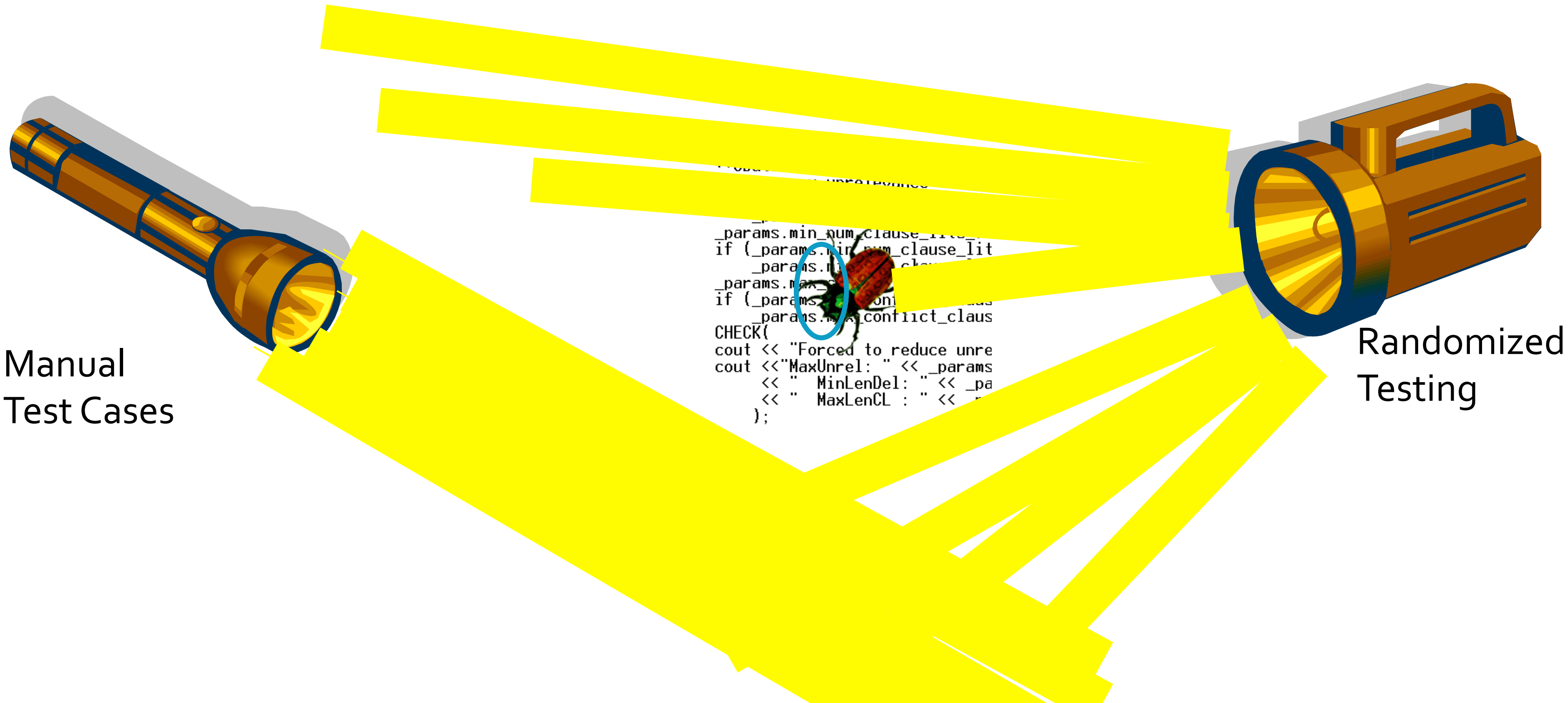
Automatic test case generation



- It requires:
  - Choosing adequacy criteria
  - Generate inputs
  - Check output

Motivation

- Extend the spectra of possible inputs in a test case
  - We have no idea where the failure could be



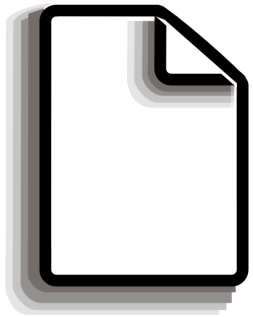
Fuzz testing

- Idea: See how an application endures "noise" in the input

- Inspired in real life:
  - Unix `ssh` and `vi` use in a dial-up connection.
- **Fuzzers**
  - Tools that create mal-formed random inputs to a program
    - To crash the program
    - To find security flaws

Fuzz Testing - Typical Targets

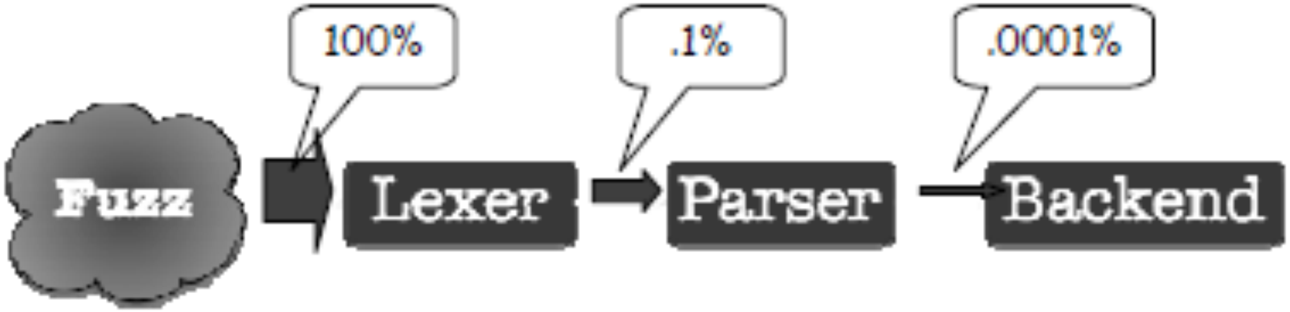
- File formats
  - e.g. documents, pictures



- Network protocols
  - servers and clients

- Specific interfaces
  - e.g. DOM

Fuzz Testing - Example



- The Lexer can be exercised in detail
- But few inputs reach the parser…
- And even less inputs reach the runtime …

Random Testing

- Create program inputs randomly
- Observe if the program behaves "correctly"
  - Using explicit contracts (pre & posts)
  - Implicitly: runtime undeclared exceptions
- Advantages:
  - Easy to implement
  - Good coverage if the test suite is big enough

Ramdom Testing - Example

```
int double (int v) {
    return 2*v;
}

void testMe (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

- Possible inputs:
  - x = 2, y = 10
    - First IF is false
  - x = 102, y = -10
    - First IF is false
  - x = 4, y = 2
    - First IF is true
  - x = 34, y = 0
    - First IF is false
  - x = 40, y = 20
    - It hits ERROR

Simple Random Generation

- Given m(x1:T1,…,xk:Tk) a method under test
- For each xi:
  - If Ti is a primitive data type
    - A random primitive value
  - If Ti is a reference, choose randomly among:
    - The null value
    - The constructor with no arguments (if it exists)

An example

- Given this method:

```
public static int max(int a, int b) {
        if (a>b) {
                return a;
        }
        else {
                return b;
        }
}
```

- We generate random values for a and b:
  - a=124, b=3
  - a=-15, b=0

Another example

- Given this method:

```
public static Integer largest(LinkedList<Integer> list) {
        int index = 0;
        int max = Integer.MIN_VALUE;
        while (index <= list.size()-1) {
                if (list.get(index) > max) {
                        max = list.get(index);
                }
                index++;
        }
        return max;
}
```

- We can use LinkedList() or null
  - list=null
  - list=[] (the empty list)

Simple Random Testing - Limitations

- Each test sequence is independent from the other sequences (redundancy)
- Complex objects tend to be rather simple

(Complex) Random Testing

- Given m(x1:T1,…,xk:Tk):Tk+1 a method under test
- For each xi:
  - If Ti is a primitive data type
    - A random primitive value
    - A method m’(y1:T’1,…yj:T’j): Ti (the return value is Ti)
      - Recursively apply the same procedure to y1,…,yj
  - If Ti is a reference, choose randomly among:
    - The null value
    - A method m’(y1:T’1,…yj:T’j): Ti (the return value is Ti)
      - Recursively apply the same procedure to y1,…,yj

Example 1

- Program under test:

```
public static int max(int a, int b) {
        if (a>b) {
                return a;
        }
        else {
                return b;
        }
}
```

- Random test sequence:

```
int b = 254;
LinkedList<Integer> l = new LinkedList<Integer>();
l.add(0,2);
Integer x = l.get(0);
int a = x.intValue();
max(a,b)
```

Example 2

- Method under test:

```
public static Integer largest(LinkedList<Integer> list) {
        int index = 0;
        int max = Integer.MIN_VALUE;
        while (index <= list.size()-1) {
                if (list.get(index) > max) {
                        max = list.get(index);
                }
                index++;
        }
        return max;
}
```

- Random test sequence:

```
LinkedList<Integer> listo = new LinkedList<Integer>();
listo.add(0,2);
int into = listo.getFirst();
listo.add(0,into);
largest(l);
```

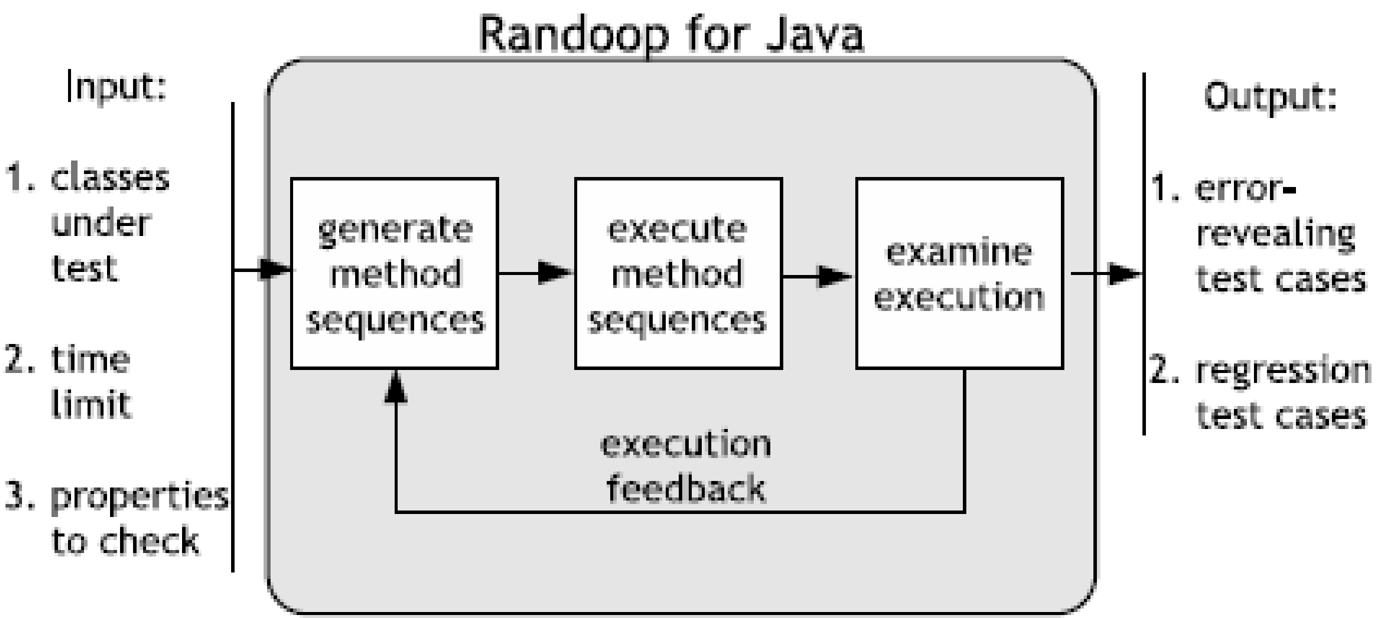
Some problems

- How do we know a test is “interesting”
  - We know some sort of Test Oracle

- How do we select non-redundant tests?
  - More generation time => More generated tests
  - Test Selection is very expensive

Randoop

- Randomized Test Generator for Object-Oriented Programs
- Execution Feedback



Randoop: Contracts

- Object level:
  - o.equals(o) returns true
  - o.equals(o) throws no exception
  - o.hashCode() throws no exception
  - o.toString() throws no exception
- Method level:
  - Method throws no AssertionError
  - Method throws no NullPointerException if all args != null
- User can add more contracts (Randoop interface)

Randoop: Test Selection

- Randoop classifies generated tests into:
  - Failing Test Cases:
    - Execution led to a contract failure
  - Normal Test Cases:
    - Execution did not violate any contract
- All failing tests are collected.
- Normal tests are filtered

Randoop: Filtering

- Equality
  - The equals() method is used to see if a given object was created before
  - A pool with all created objects is kept alive.
- Null
  - Avoid using a null return value (use “null” directly instead).
- Exceptions
  - If the test leads to a exception, do not extend that.

Randoop: Filtering

- The execution of these two sequences is redundant

```
Date d = new Date(2006,2,14);
d.setMonth(-1);
```

```
Date d = new Date(2006,2,14);
d.setMonth(-1);
d.setDay(5);
```

- Smaller size improves program understanding.

Randoop: Filtering

- The execution of these two sequences is redundant

```
HashSet s0 = new HashSet();
Integer int0 = new Integer(1);
Integer int1 = new Integer(0);
s0.add(int0);
s0.add(int1);
```

```
HashSet s0 = new HashSet();
Integer int0 = new Integer(0);
Integer int1 = new Integer(1);
s0.add(int0);
s0.add(int1);
```

- Both hashSets are equal w.r.t. the equals() method.

Randoop: Example

- Test case exhibiting a failure
  - Fails on Sun 1.5, 1.6.

```
public static void test1() {
  LinkedList l1 = new LinkedList();
  Object o1 = new Object();
  l1.addFirst(o1);
  TreeSet t1 = new TreeSet(l1);
  Set s1=Collections.unmodifiableSet(t1);
  Assert.assertTrue(s1.equals(s1));
}
```

Randoop: Example

- Regression Test
  - Passes on Sun 1.5, fails on Sun 1.6 Beta 2.

```
public static void test2() {
  BitSet b = new BitSet();
  Assert.assertEquals(64, b.size());
  b.clone();
  Assert.assertEquals(64, b.size());
}
```

Random Testing

- Advantages:
  - Few requirements
  - Unbiased
- Disavantages:
  - It does not benefit from source code information.
  - It is difficult to find “deep” errors.

Exhaustive Testing - Idea

- Generate all non-isomorphic valid inputs up to a given size.
- Use programmatic contracts to decide if an input is valid.
- Prune search space efficiently.

Exhaustive Testing - Example

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

- Type information
  - The class declaration states the values a field can take

- Enumerate all possible values up to a given length.

Naïve Algorithm

1. Select k the maxium input size
2. Generate all inputs up to size k
3. Discard all inputs that do not satisfy the precondition
4. Execute the program
5. Check the postcondition

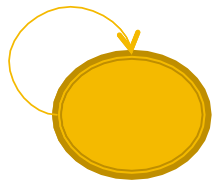
Example

- Binary trees
- How many instances of k<=3?
  - 3 nodes
  - 2 values per node (left, right)
  - 4 possible values (consider also null) for each node and the tree instance.
  - 4 * (4 * 4) ^3 = 16.384 instances!
  - For 4 nodes it grows to more than 1.000.000!
- Number of instances grow exponentially.

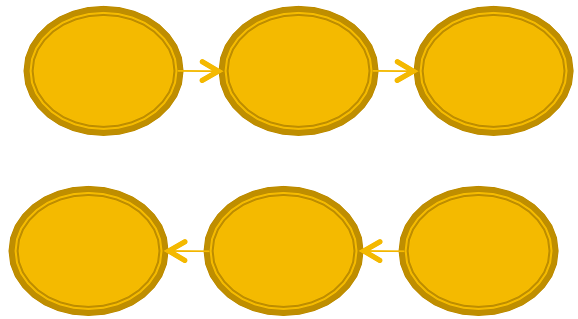
```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```

Relevant inputs

- This enumeration does not take into account many important aspects:



  - Many inputs are not trees



  - Many inputs are isomorphic instances.

Korat

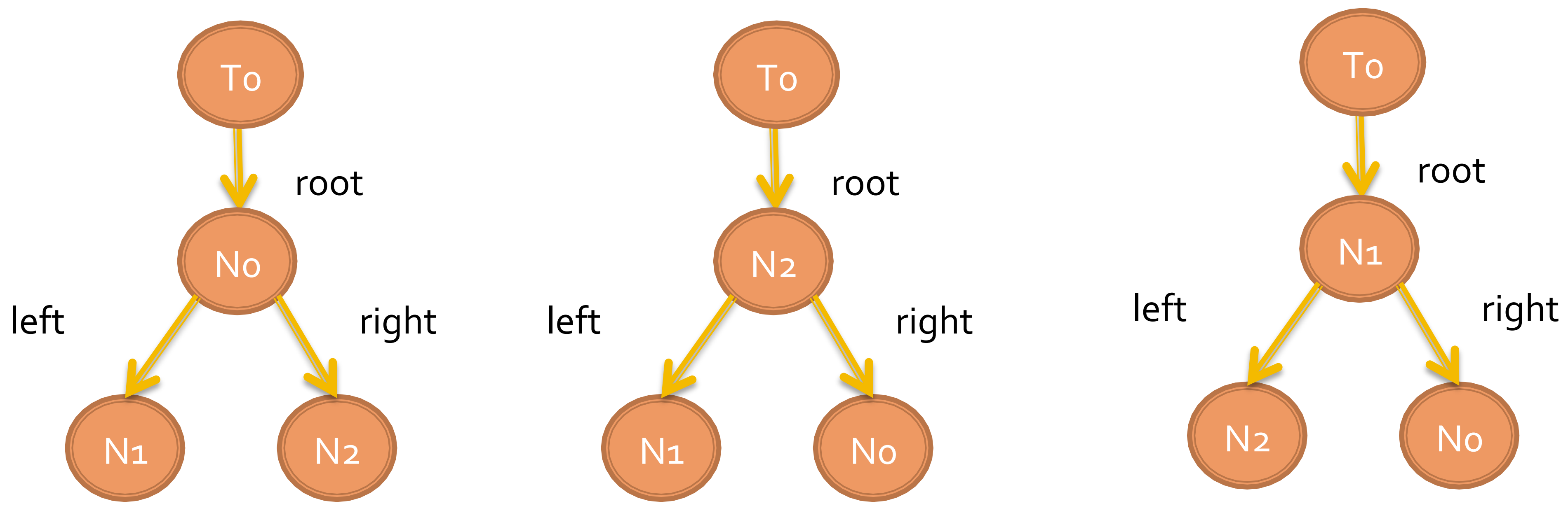
- Automated Testing Based on Java Predicates
- http://korat.sourceforge.net/index.html
- Efficiently enumerates instances by:
  - Monitoring which fields are accessed
    - Which values should I change?
  - Avoiding isomorphic instances
    - Vector representation of the heap configuration.

Binary Trees - Example

```
boolean repOk() {
  if (this.root == null)
       return true;
  Set visited = new HashSet();
  visited.add(t.root);
  List workList = new LinkedList();
  workList.add(t.root);
  while (!workList.isEmpty()) {
    Node current = (Node)workList.removeFirst();
    if (current.left != null) {
      if (!visited.add(current.left))
        return false;
      workList.add(current.left); }
    if (current.right != null) {
      if (!visited.add(current.right))
        return false;
      workList.add(current.right); }
  }
  return true;
}
```

Binary Trees Example

- Isomorphic instances
  - Identical except object names

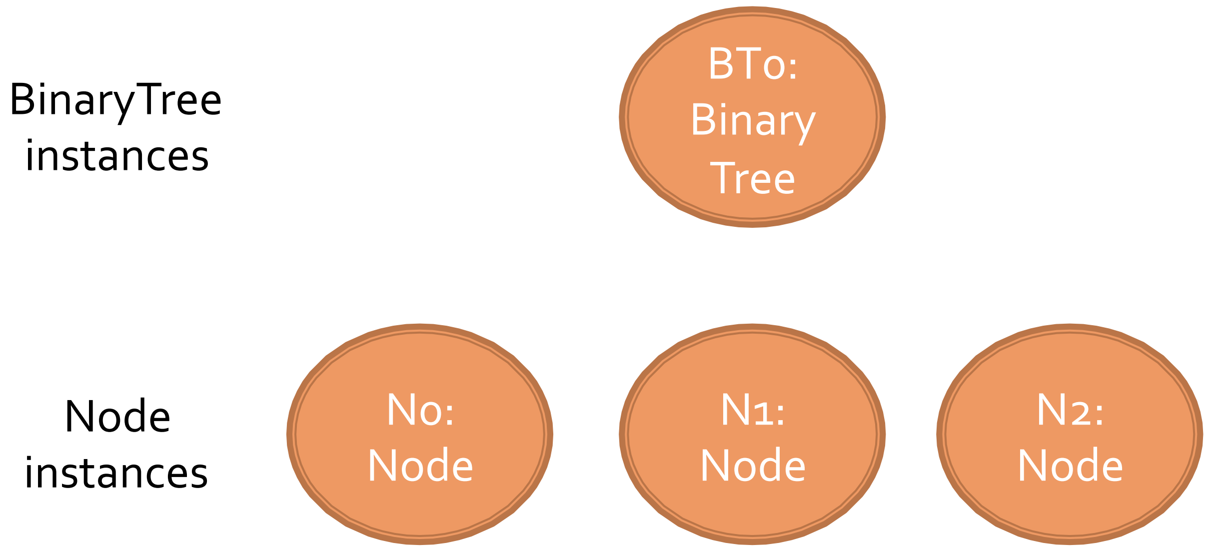


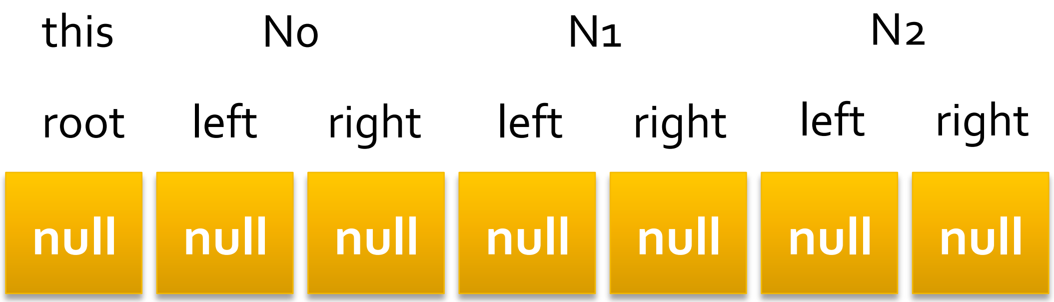
Binary Trees Finitization

```
public static Finitization finBinaryTree(int NUM_Node) {
      Finitization f = new Finitization(BinaryTree.class);
      ObjSet nodes = f.createObjects("Node", NUM_Node);
      nodes.add(null);
      f.set("root", nodes); // root in null + Node
      f.set("Node.left", nodes); // Node.left in null + Node
      f.set("Node.right", nodes); // Node.right in null+ Node
      return f;
}
```

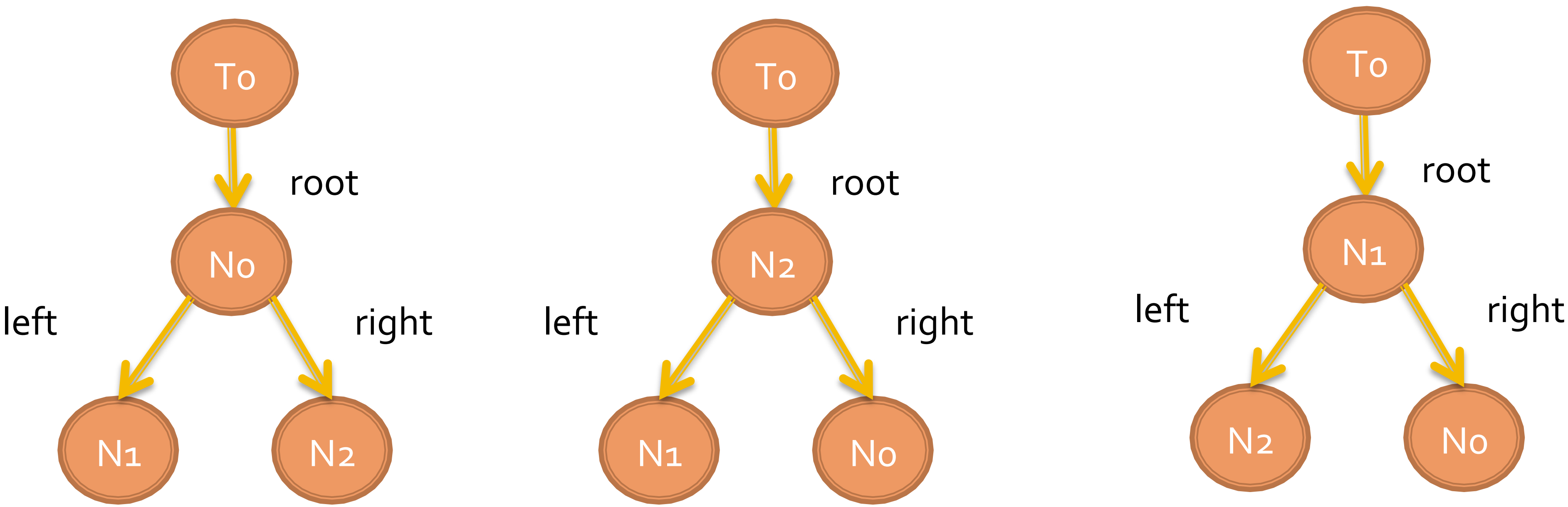
Binary Trees – Example (3 nodes)

```
class BinaryTree {
  Node root;
  class Node {
    Node left;
    Node right;
  }
}
```





Isomorphic enumeration

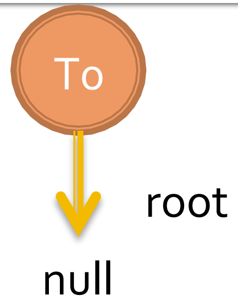


	N0		N1		N2	
root	left	right	left	right	left	right
N0	N1	N2	null	null	null	null

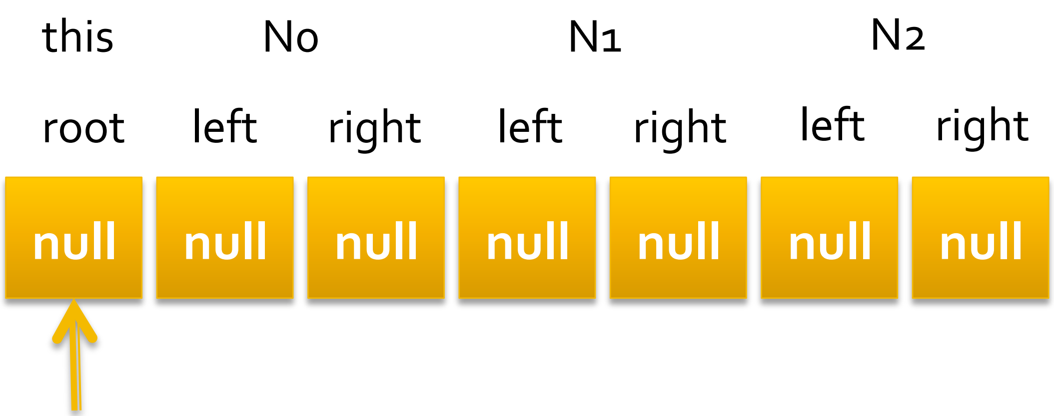
	N0		N1		N2	
root	left	right	left	right	left	right
N1	null	null	N2	N0	null	null

	N0		N1		N2	
root	left	right	left	right	left	right
N2	null	null	null	null	N1	N0

Binary Trees Example

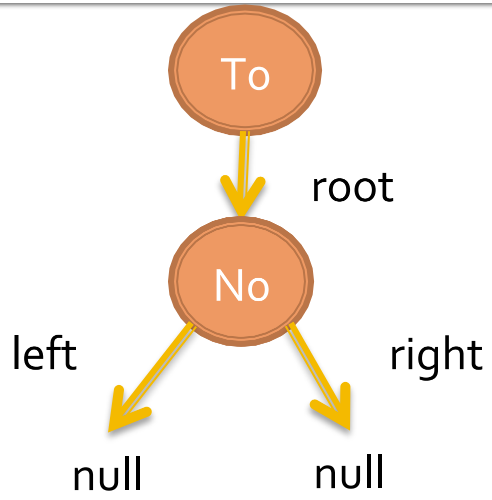


this.root



this.repOk()==true
(move right)

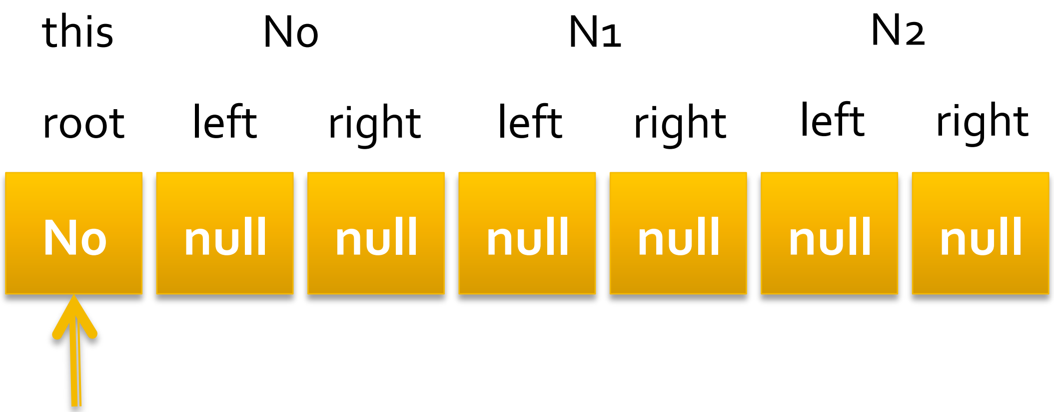
Binary Trees Example



this.root

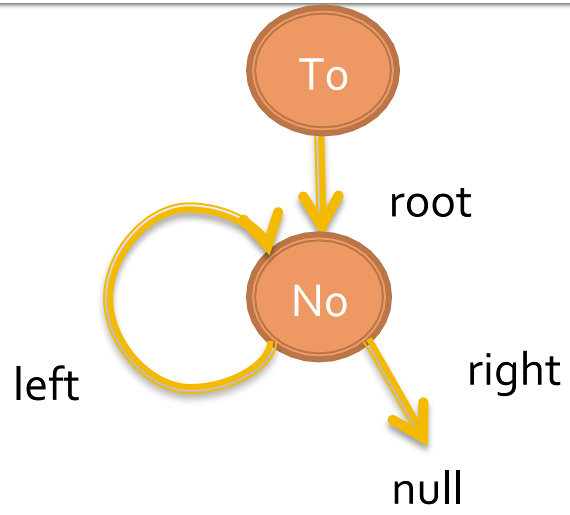
No.left

No.right



this.repOk()==true
(move right)

Binary Trees Example



this.root

No.left

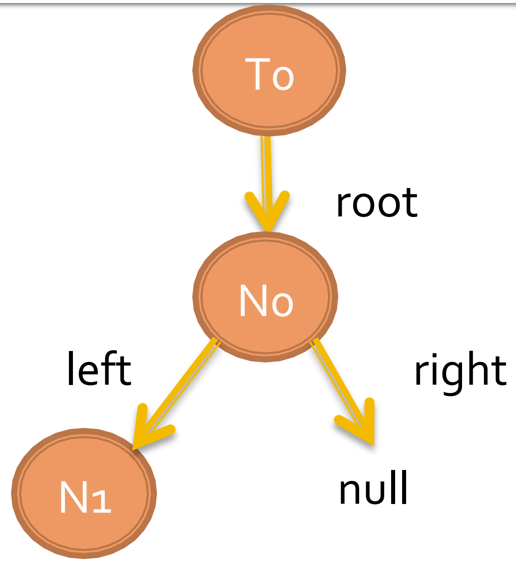
No.right

No.left

this	No		N1		N2	
root	left	right	left	right	left	right
No	**No**	**null**	**null**	**null**	**null**	**null**

this.repOk()==false
(increase index)

Binary Trees Example



this	N0		N1		N2	
root	left	right	left	right	left	right
N0	N1	null	null	null	null	null

this.root

N0.left

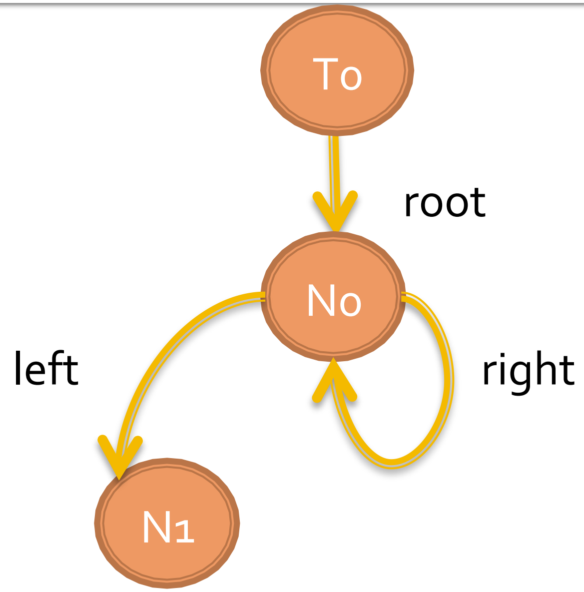
N0.right

N1.left

N1.right

this.repOk()==true
(move right)

Binary Trees Example



this.root

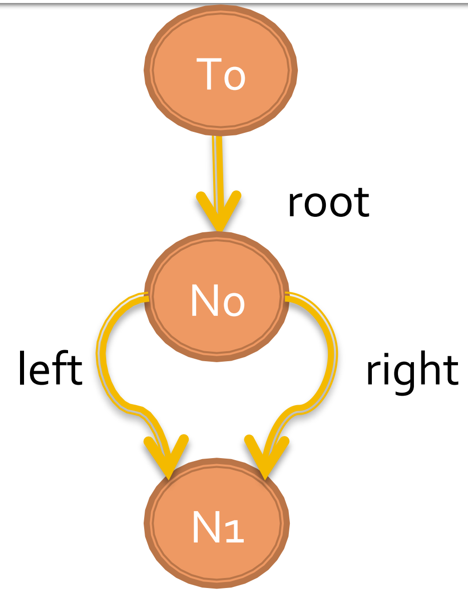
N0.left

N0.right

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	N1	N0	null	null	null	null

this.repOk()==false
(increase index)

Binary Trees Example



this.root

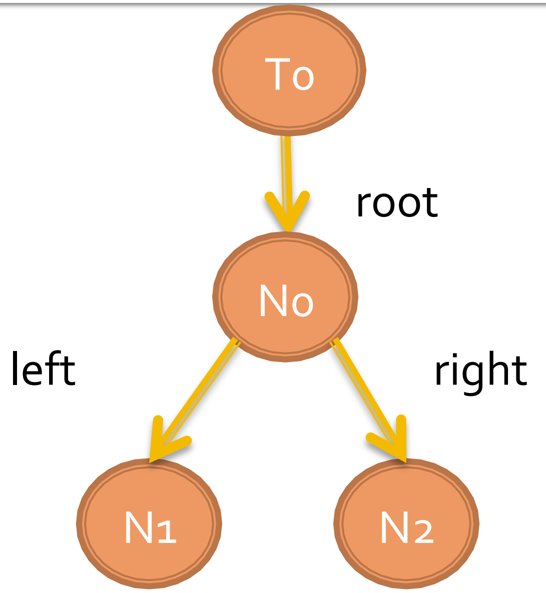
N0.left

N0.right

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	**N1**	**N1**	**null**	**null**	**null**	**null**

this.repOk()==false
(increase index)

Binary Trees Example



this.root

N0.left

N0.right

N1.left

N1.right

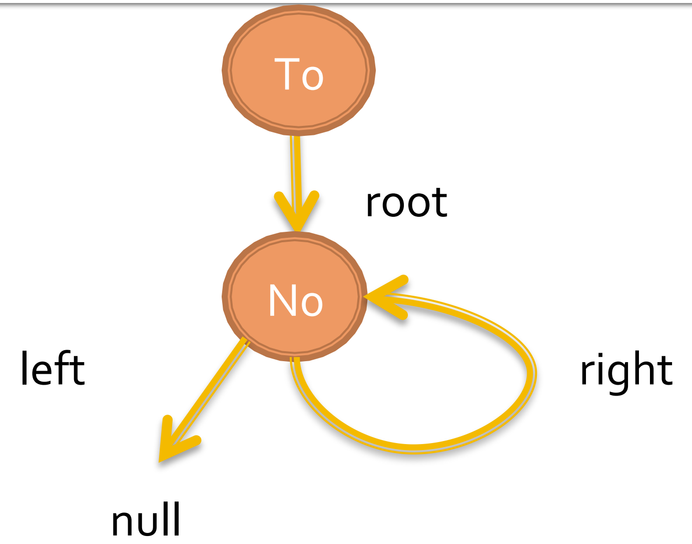
N2.left

N2.right

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	**N1**	**N2**	**null**	**null**	**null**	**null**

this.repOk()==true
(backtrack)

Binary Trees Example



this.root

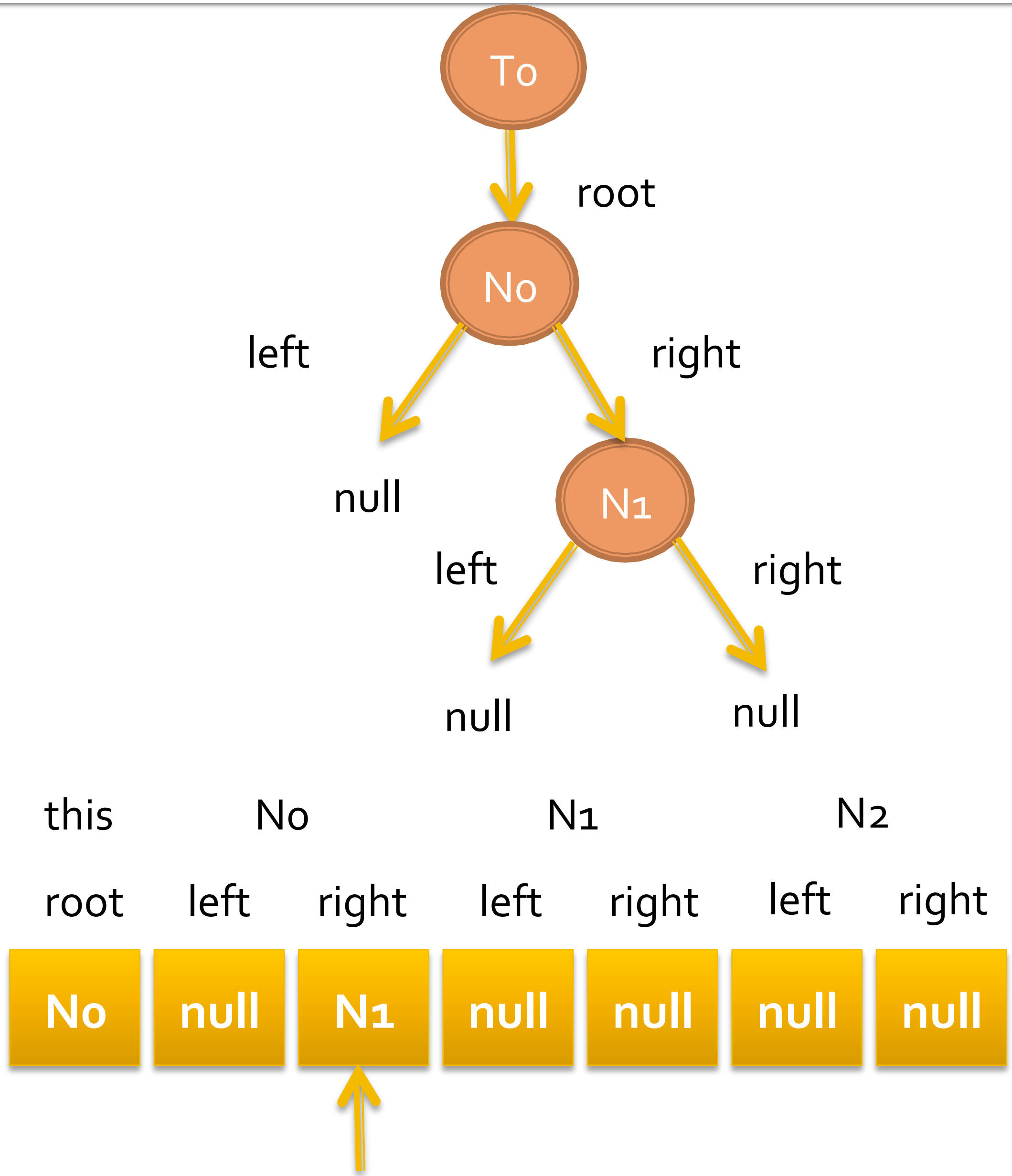
N0.left

N0.right

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	**null**	**N0**	**null**	**null**	**null**	**null**

this.repOk()==true
(move right)

Binary Trees Example



this.root

No.left

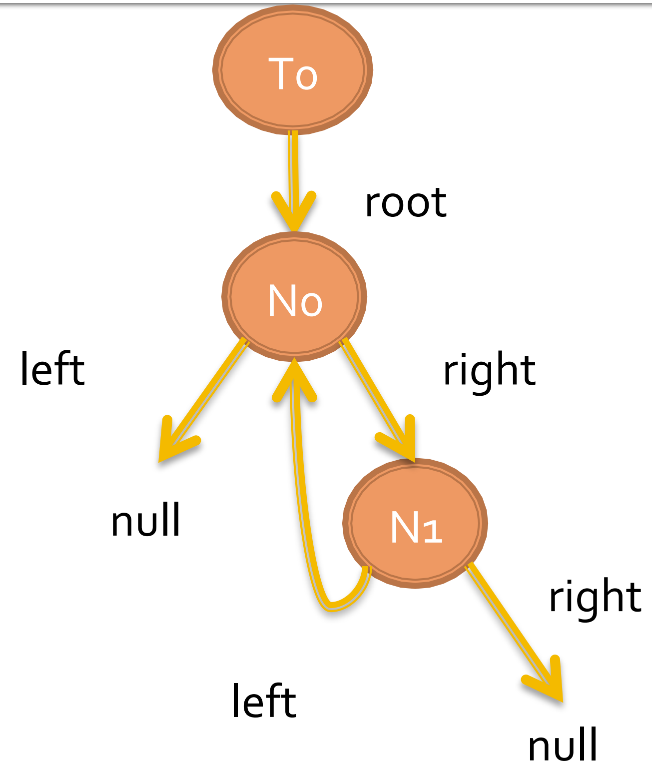
No.right

N1.left

N1.right

this.repOk()==true
(move right)

Binary Trees Example



this.root

N0.left

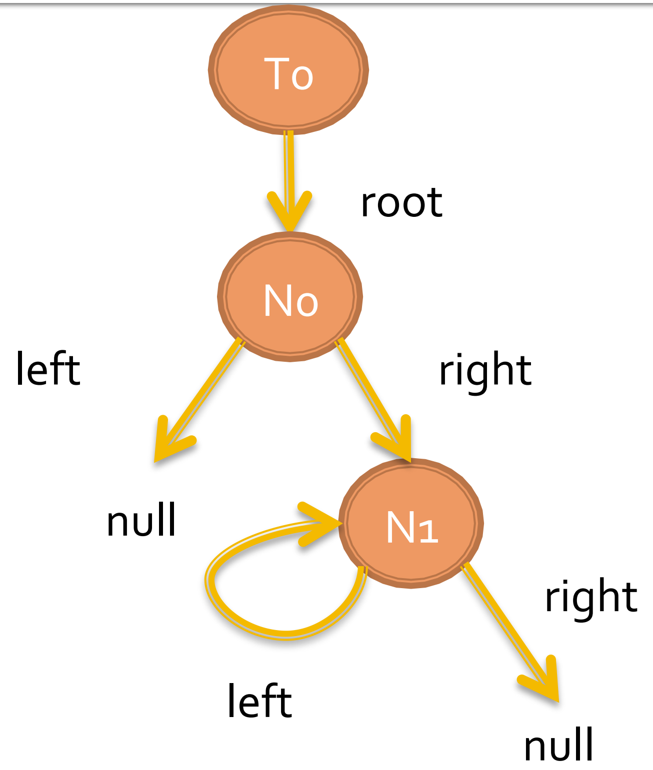
N0.right

N1.left

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	**null**	**N1**	**N0**	**null**	**null**	**null**

this.repOk()==false
(increase index)

Binary Trees Example



this.root

N0.left

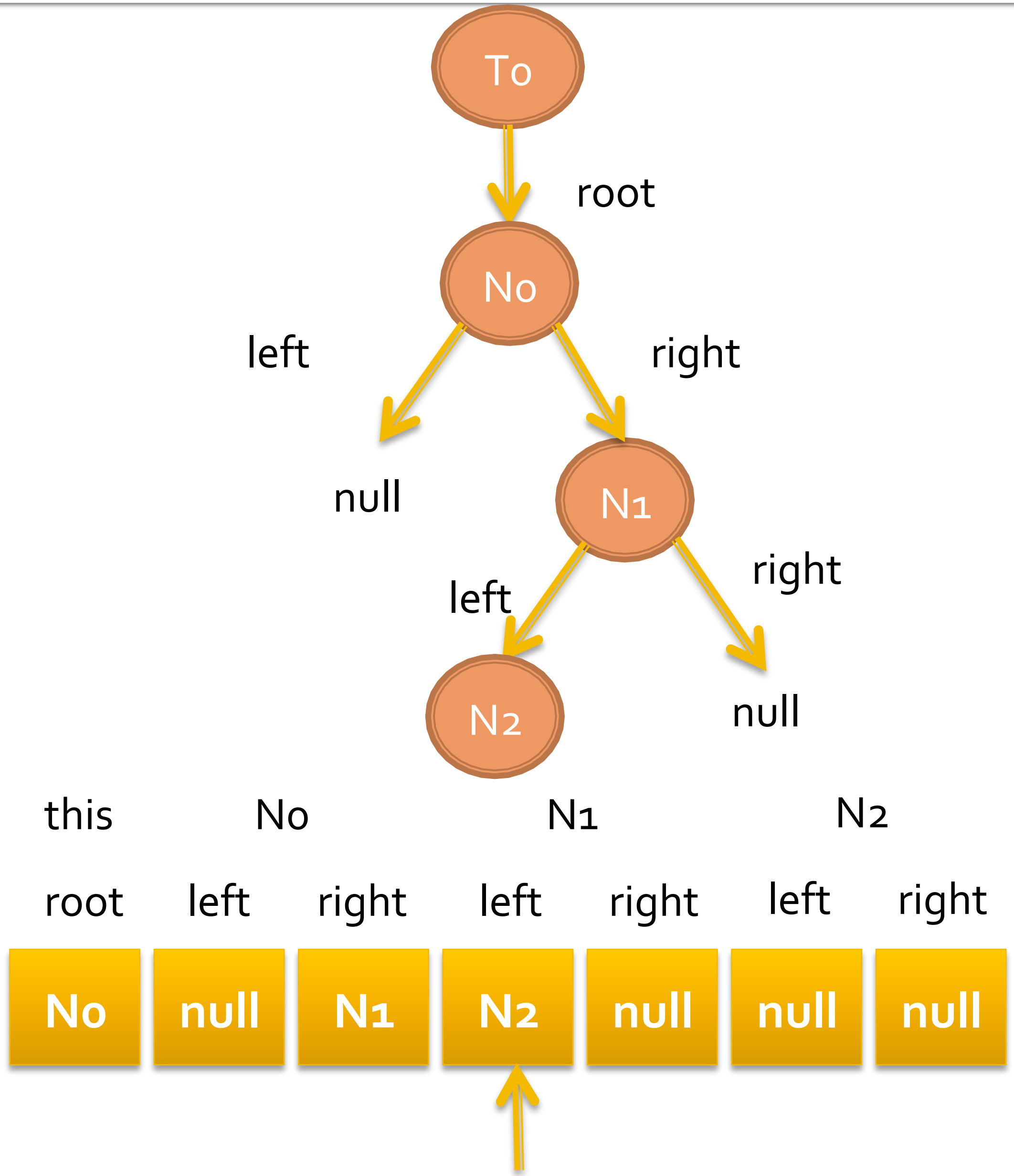
N0.right

N1.left

this	N0		N1		N2	
root	left	right	left	right	left	right
N0	**null**	**N1**	**N1**	**null**	**null**	**null**

this.repOk()==false
(increase index)

Binary Trees Example



this.root

N0.left

N0.right

N1.left

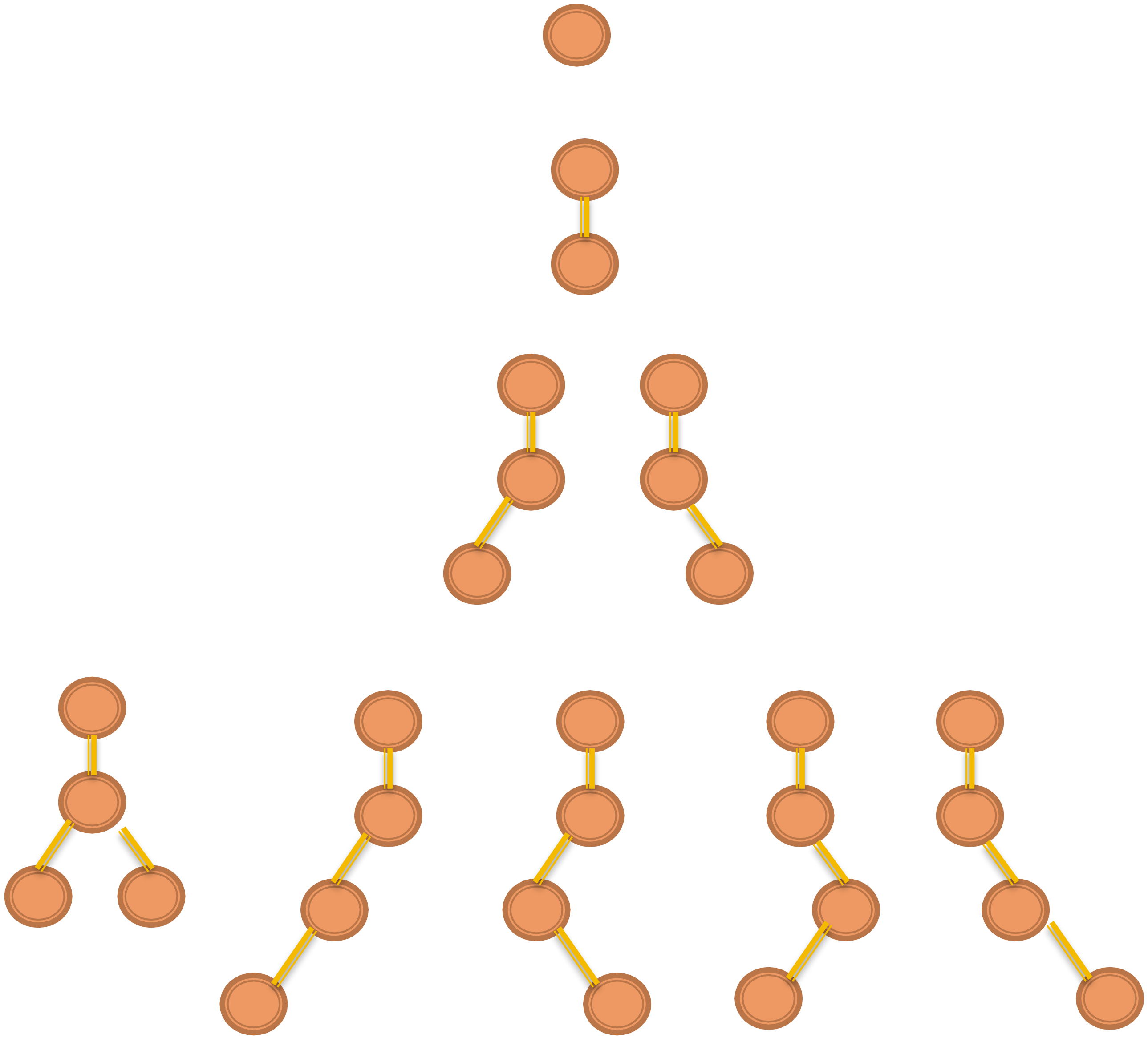
N1.right

N2.left

N2.right

this.repOk()==true
(backtrack)

Binary Trees Example



The Technique

- Isomorphic instances enumeration:
  - Given all the already assigned values, Korat can only use the same Restriction on values (only previous values +1)
- Prune un-interesting search space:
  - Korat monitors the field accesses to judge which fields must be modified during backtracking.

Korat – Systematic Testing

- For each non-isomorphic instance up to the chosen finitization satisfying the Java predicate
  - Creates the instance using Java reflection
  - Execute the selected method on the instance
  - Check the Java predicates on exit

Korat – Limitations

- How do we deal with Strings, floats, etc.?
- Korat does not build the instance using a method sequence
- The performance of the Java predicate directly affects the Test Case Generation process.
- Exhaustive generation can be infeasible for bigger scopes (100? 1000? 100000?)

Recap

- **Randoop**
  - Random Testing with Feedback loop
- **Korat**
  - Efficient non-isomorphic test input generation

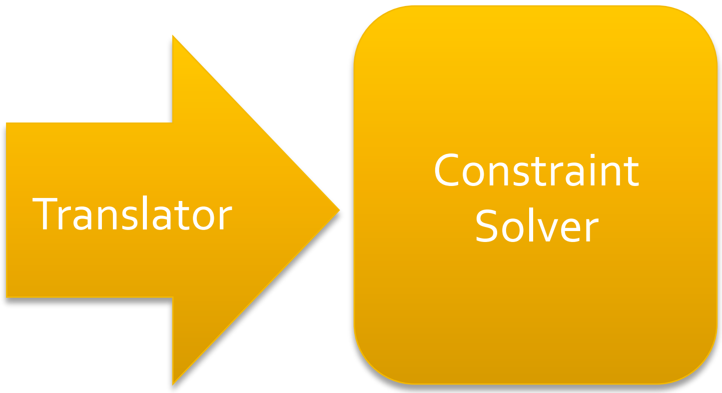
- How can we use the source code to generate inputs?
  - **Pex:** white-box test case generation using concolic execution.

Verifying program behavior

- Examples: ESC/Java2, JMLForge

Verifying program behavior

```
//@ requires …
//@ ensures …
procedure m(x,y,z) {
   if (x>y) {
     …
   } else {
     …
   }
   if (z==y) {
     …
   }
}
```

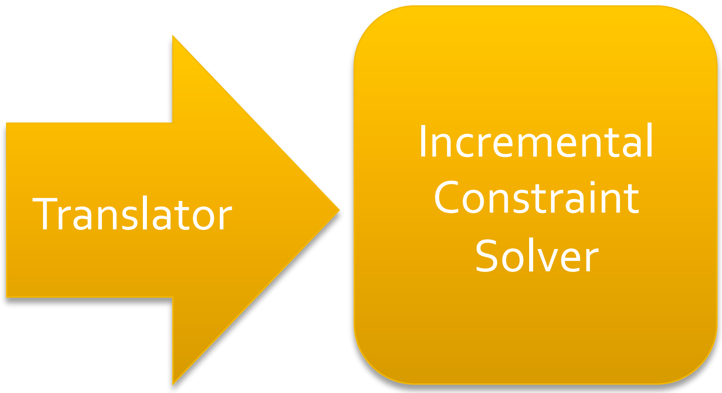


Incremental Constraint Solvers

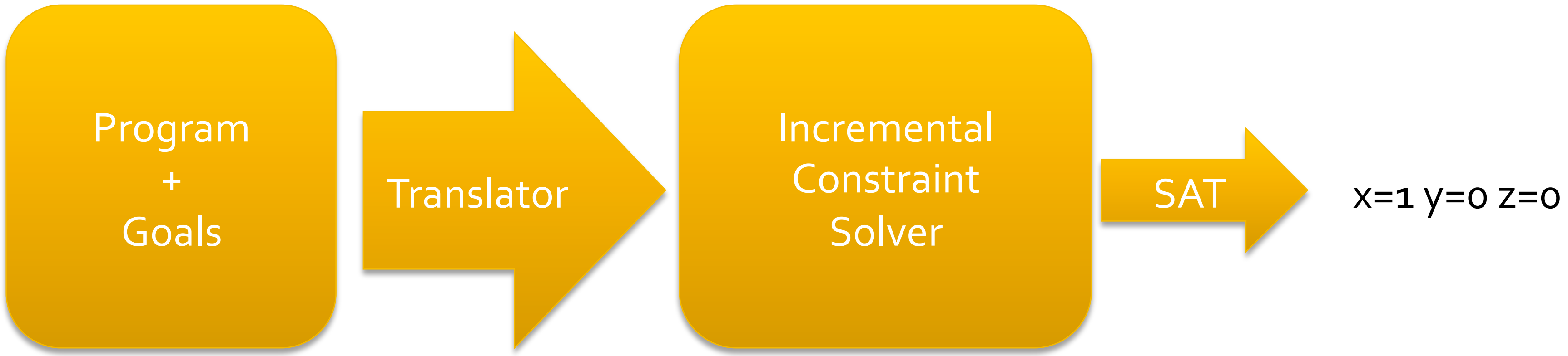
- Incremental solvers (SMT/SAT) allow the user to add new clauses/axioms when a solution is produced
  - This could be use to enumerate more solutions
  - Or it could be use to enumerate “interesting” executions of the program

Constraint Generation

```
//@ requires …
//@ ensures false;
procedure m(x,y,z) {
  if (x>y) {
    GOAL_0
    …
  } else {
    GOAL_1
    …
  }
  if (z==y) {
    GOAL_2
    …
  } else {
    GOAL_3
  }
}
```

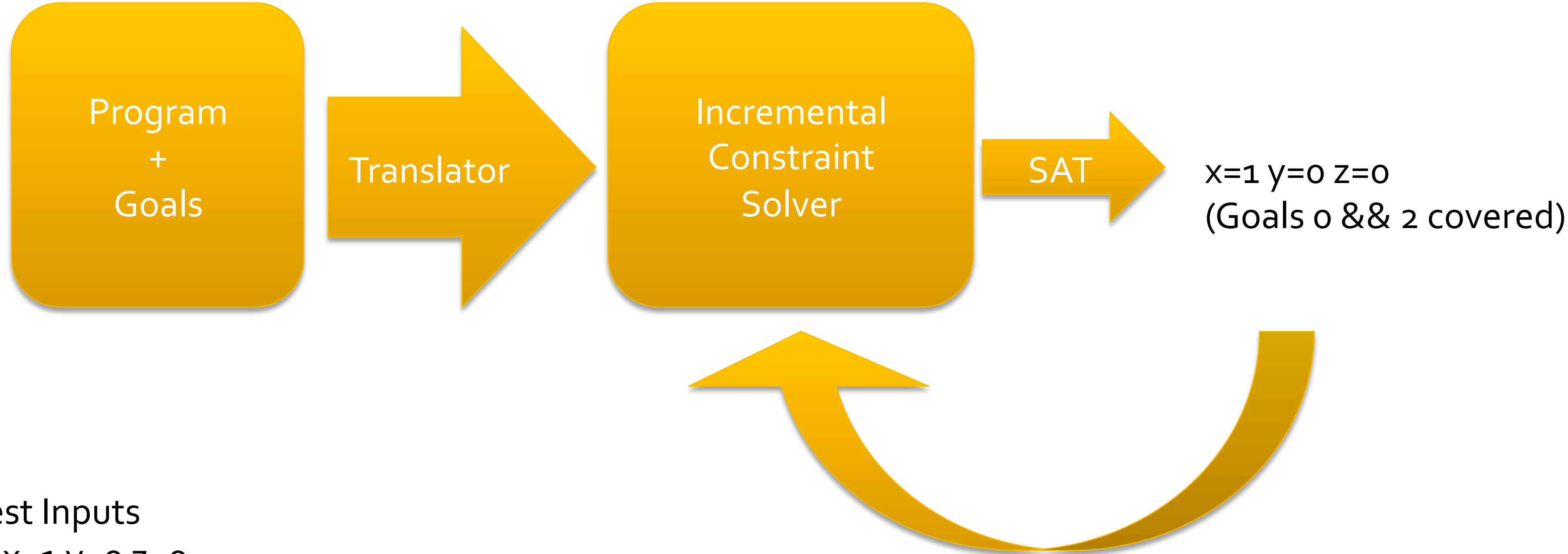


Constraint Generation



- Counterexample:
  - an execution trace (inputs) s.t. the postcondition fails (ensures “false”)
  - some goals were covered

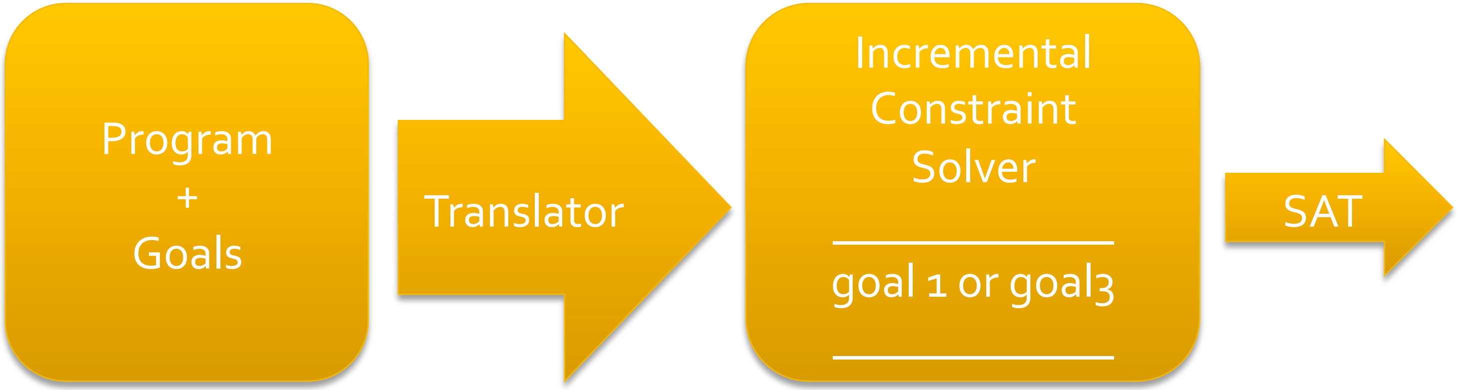
Constraint Generation



Test Inputs
1) x=1 y=0 z=0

we push a new axiom to the
Constraint solver
Goal 1 OR Goal 2 should be
covered now

Constraint Generation



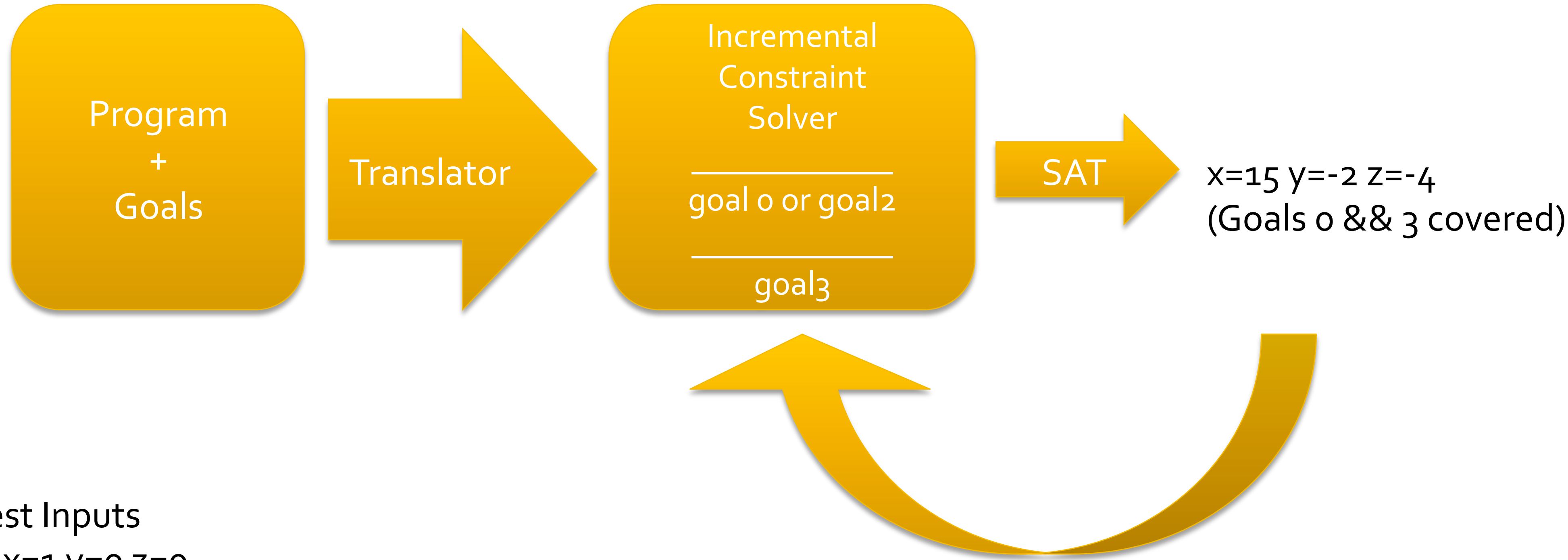
x=9 y=9 z=9
(Goals 1 && 2 covered)

Test Inputs
1) x=1 y=0 z=0

2) x=9 y=9 z=9

we push a new axiom to the Constraint solver
Goal 1 OR Goal 3 should be covered now

Constraint Generation



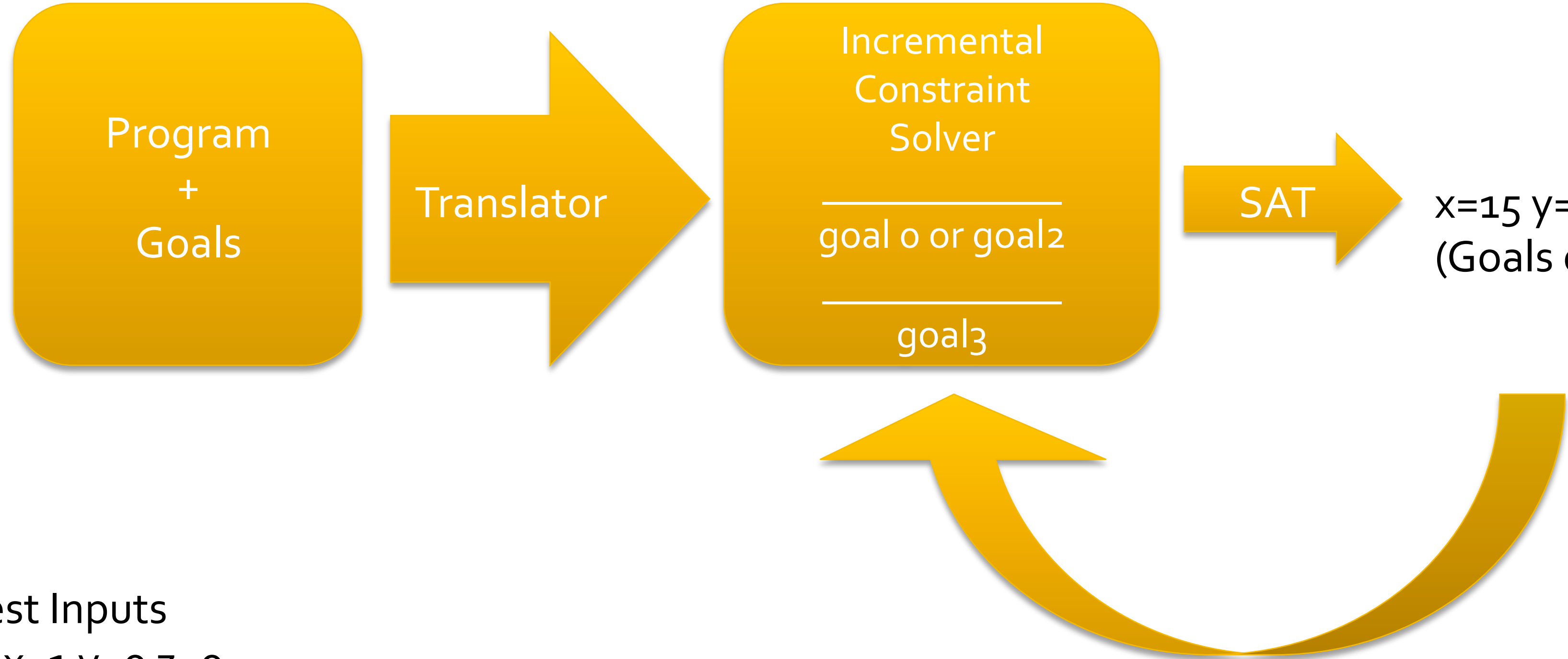
Test Inputs
1) x=1 y=0 z=0

2) x=9 y=9 z=9

3) x=15 y=-2 z=-4

we push a new axiom to the
Constraint solver
Goal 3 should be covered now

Constraint Generation



x=15 y=-2 z=-4
(Goals 0 && 3 covered)

Test Inputs
1) x=1 y=0 z=0

2) x=9 y=9 z=9

3) x=15 y=-2 z=-4

No more goals to cover!

FAJITA

- JML input programs
- Branch/Goal/Path coverage
- Incremental SAT-Solving
- http://www.dc.uba.ar/fajita

- Advantages
  - Fewer calls to the constraint solver in worst case
- Disadvantages
  - Complete CFG has to be encoded as a formula (more complex that only a path)