Relational Tight Field Bounds for Distributed Analysis of Programs

Marcelo F. Frias

Instituto Tecnológico de Buenos Aires (ITBA), and CONICET, Argentina.

mfrias@itba.edu.ar

Roadmap

- Program Analyses we deal with
- Tight field bounds and their computation
- Distributing program analyses
- Some experimental data
- Conclusions

SAT-Based Bounded Bug Finding
and
Symbolic Execution

SAT-Based Bounded Bug Finding

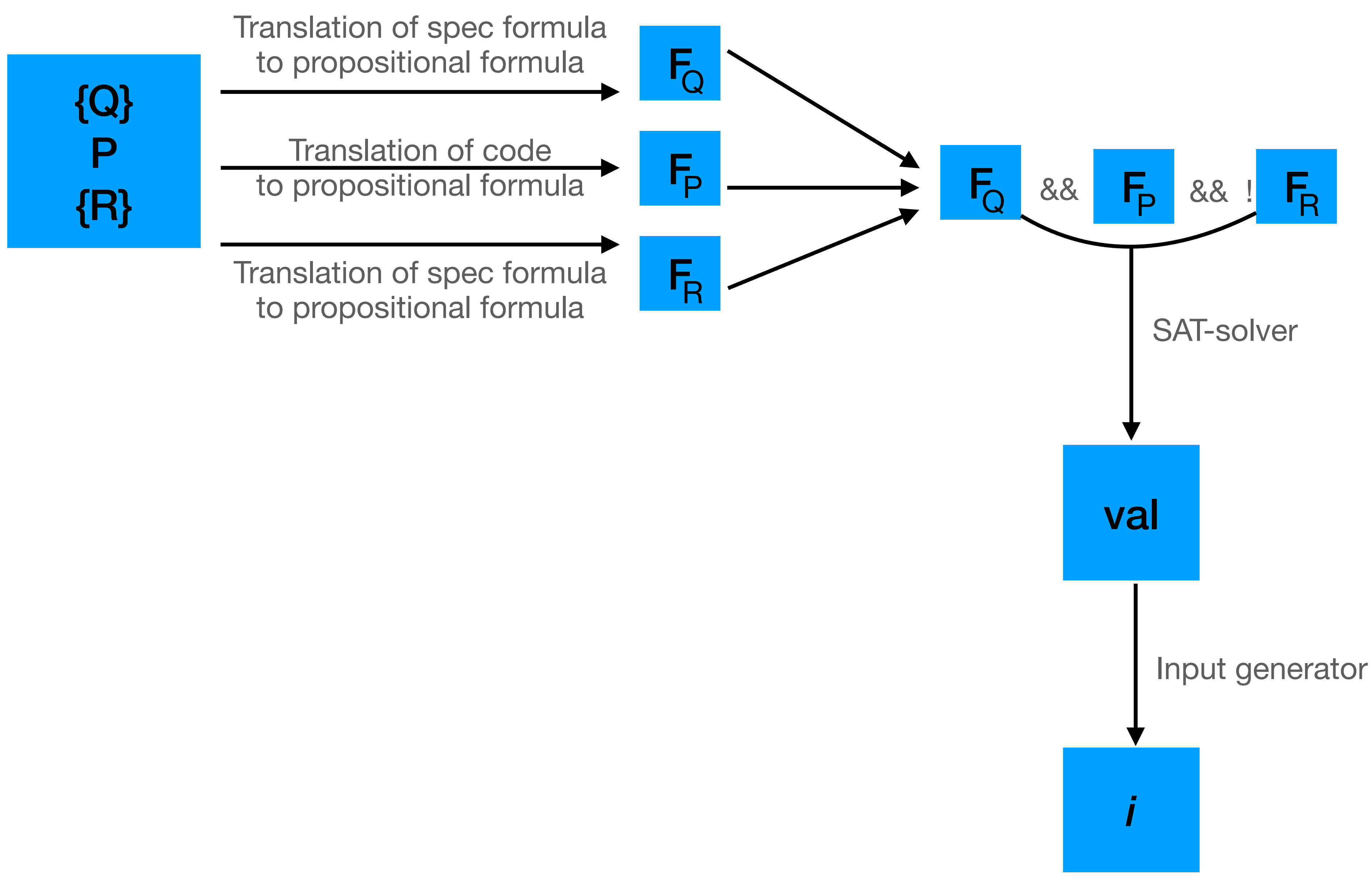
- Given a program with a contract (pre and post conditions, representation invariants, variant functions, etc.), we want to automatically detect the existence of faults.

- We automatically generate an input exposing the failure.

```
/*@
  @ requires true;
  @ ensures (\exists AvlNode n;
  @      \reach(root, AvlNode, left + right).has(n) == true;
  @      n.element==x) ==> \result==x;
  @ ensures (\forall AvlNode n;
  @      \reach(root, AvlNode, left + right).has(n) == true;
  @      n.element!=x) ==> \result==-1;
  @ signals (Exception e) false;
  @*/
  public int find(final int x) {
      AvlNode n = root;
      while (n != null) {
          if (x < n.element) {
              n = n.left;
          } else {
              if (x > n.element) {
                  n = n.right;
              } else {
                  return n.element;
              }
          }
      }
      return -1;
  }
```

TACO: Translation of Annotated Code

The process



TACO: Translation of Annotated COde: Demo

Galeotti J.P., Rosner N., Lopez Pombo C.G., Frias M.F., TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds, IEEE TSE, 39(9), 2013

Symbolic Execution

Symbolic Execution

Goals

- Automated test input generation
- Automated bug detection
- Quite scalable for certain domains (primitive types using simple constraints)
- Very active research topic when applied to complex objects (data structures).

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

(i=i0, j=j0, k=k0, true)

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

(i=i0, j=j0, k=k0, output=0, true)

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

(i=i0, j=j0, k=k0, output=0,
i0>j0 || i0>k0)

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

(i=i0, j=j0, k=k0, output=0,
(i0>j0 || i0>k0) && (i0<=j0 || k0<=j0))

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
    int output = 0;
    if (i <= j && i <= k)
        output = i;
    else
        if (i <= j || k <= j)
            output = k;
        else
            output = j;

    return output;
}
```

(i=i0, j=j0, k=k0, output=k0,
(i0>j0 || i0>k0) && (i0<=j0 || k0<=j0))

Symbolic Execution

An example

```
public int  min3(int i, int j, int k){
   int output = 0;
   if (i <= j && i <= k)
       output = i;
   else
       if (i <= j || k <= j)
           output = k;
       else
           output = j;

   return output;
}
```

Symbolic Execution

For Dynamically Allocated Structures

- Khurshid, Pasareanu and Visser proposed Lazy Initialization [TACAS'03].

- An object attribute is initialized at the time its value is accessed. Up tp that moment, the attribute value is kept symbolic.

Lazy Initialization (LI)

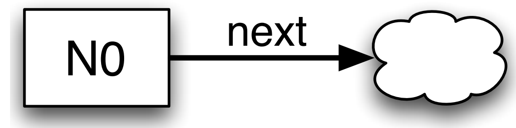
An example

```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```

Lazy Initialization (LI)

An example

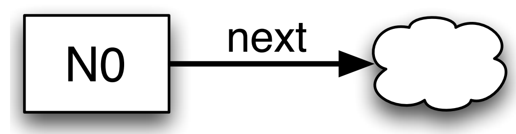
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

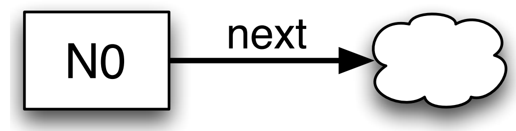
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```

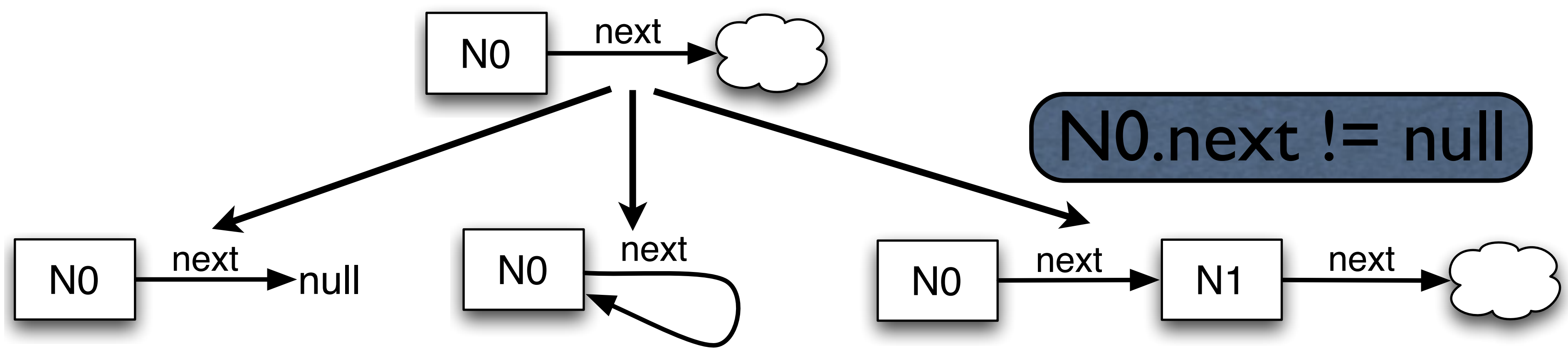


N0.next != null

Lazy Initialization (LI)

An example

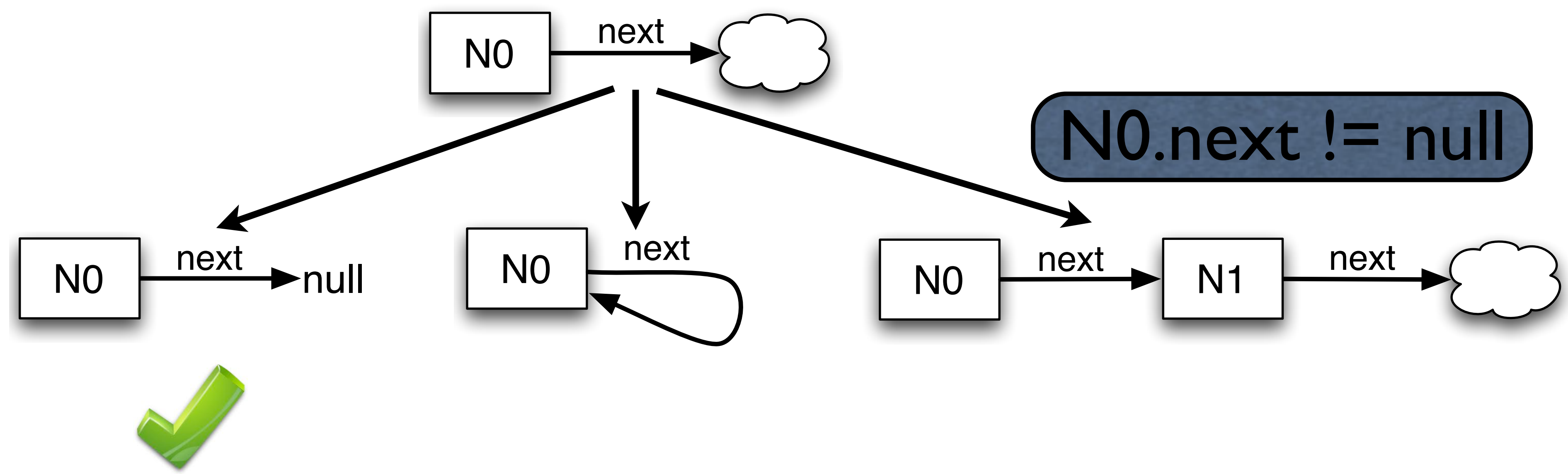
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

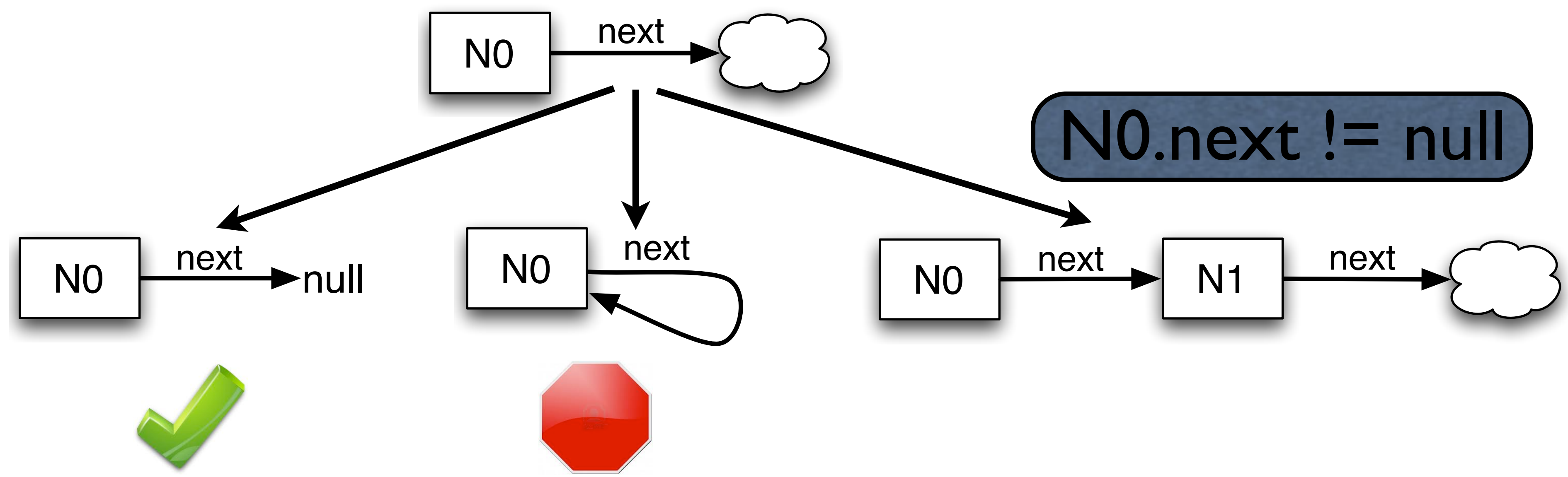
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

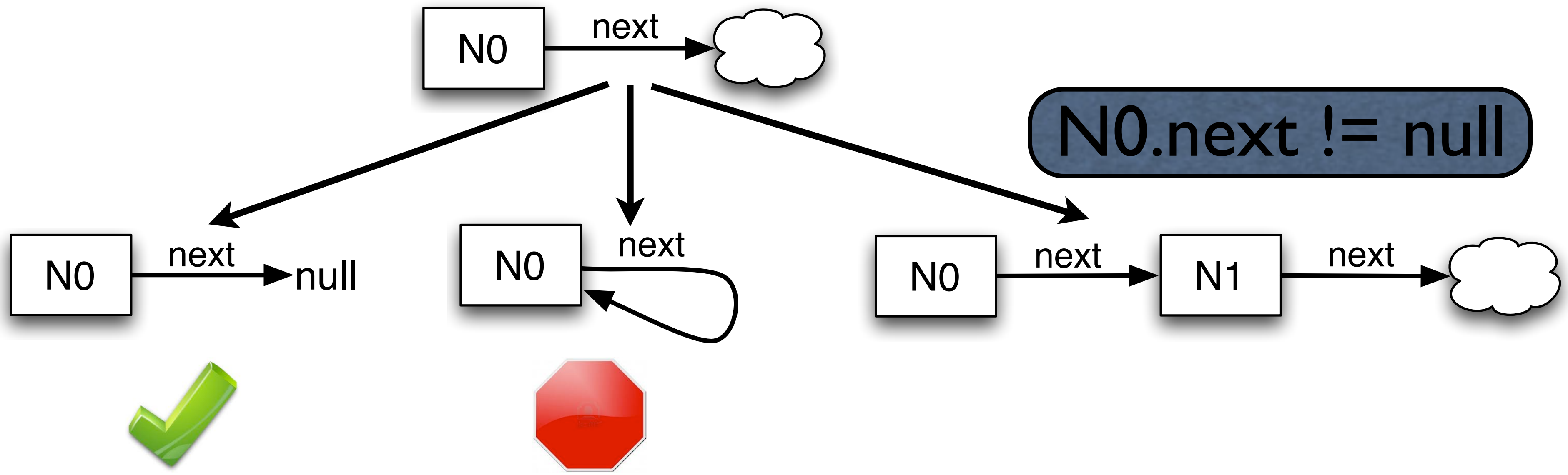
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

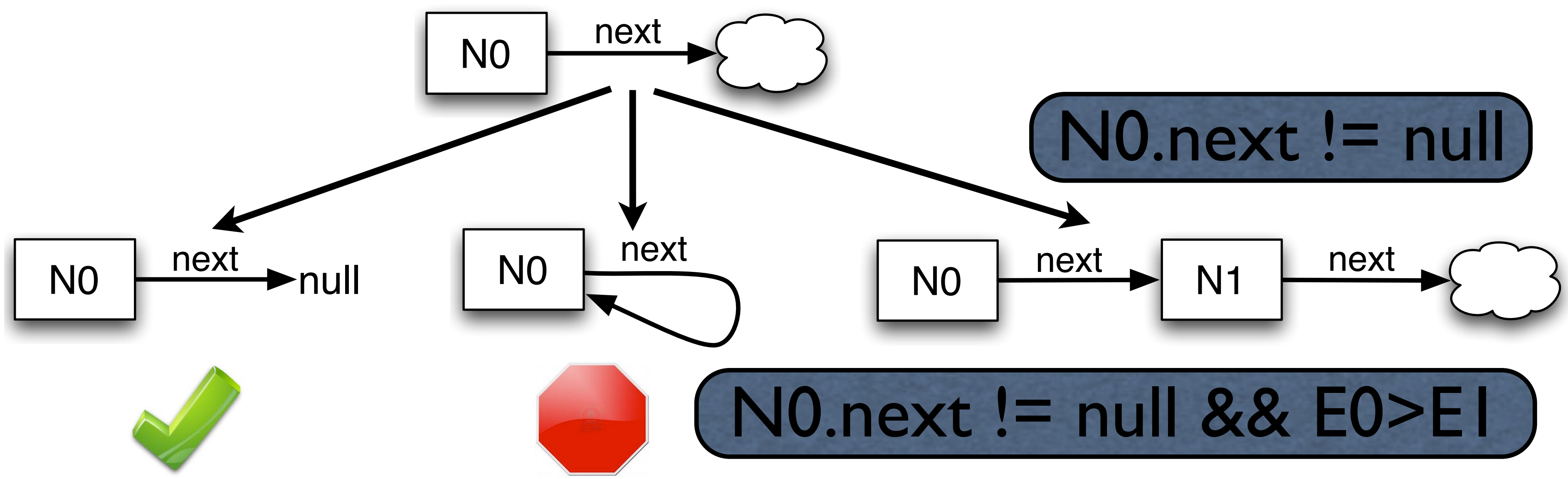
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

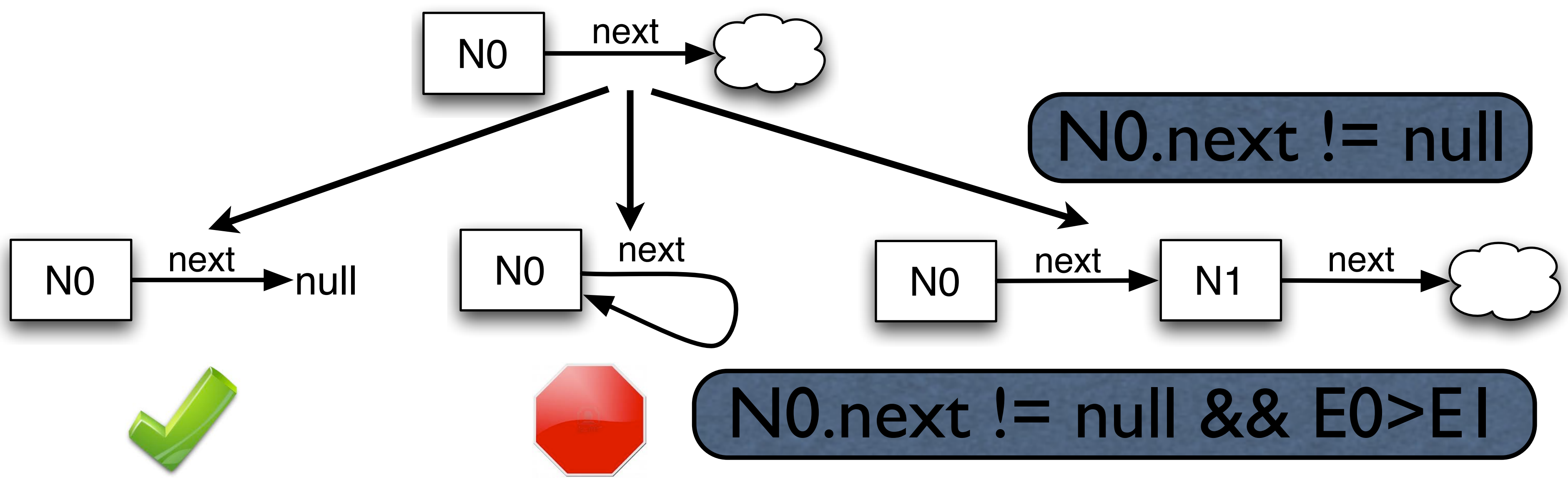
```
class Node {
   int elem;
   Node next;

   \requires Acyclic
   Node sortFirstTwo() {
      if (next != null)
        if (elem > next.elem) {
           Node t = next;
           next = t.next;
           t.next = this;
           return t;
        }
      return this;
   }
}
```



Lazy Initialization (LI)

An example

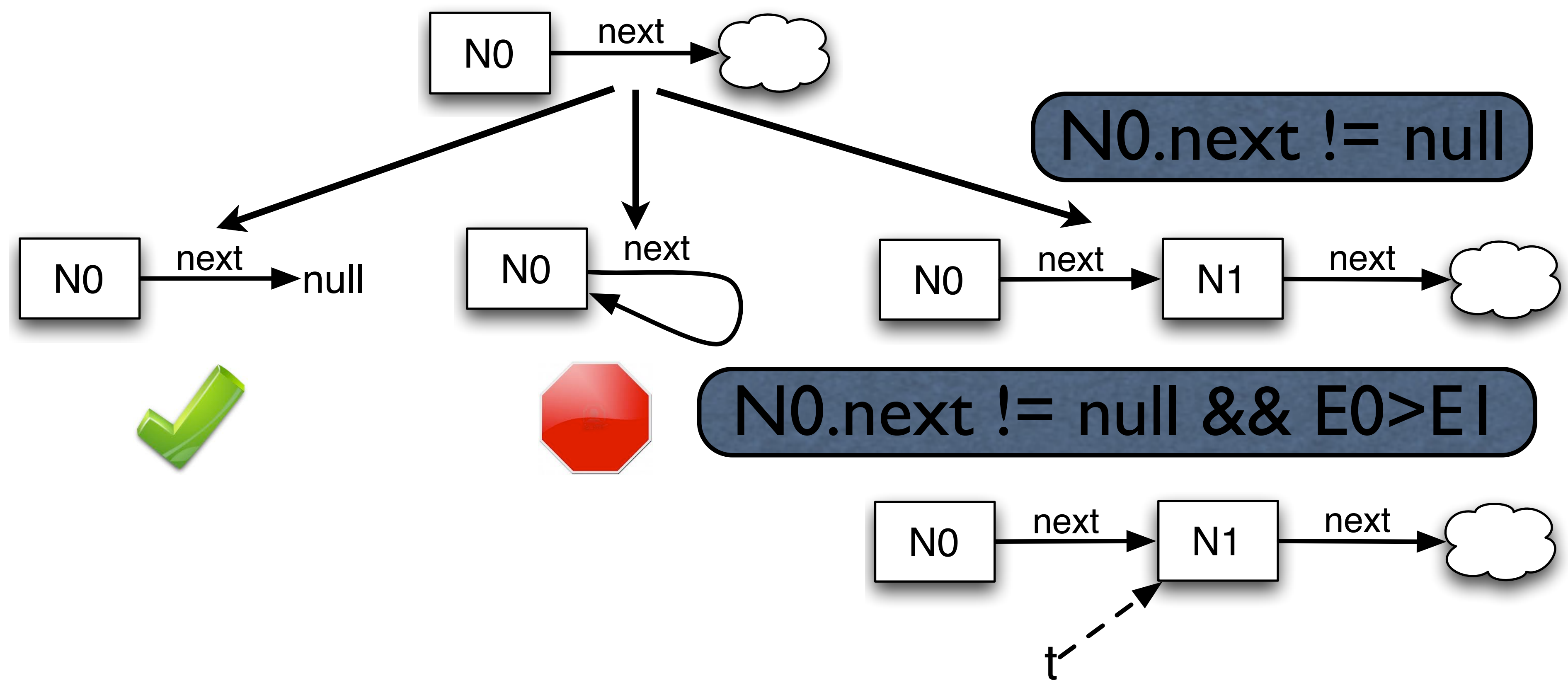
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

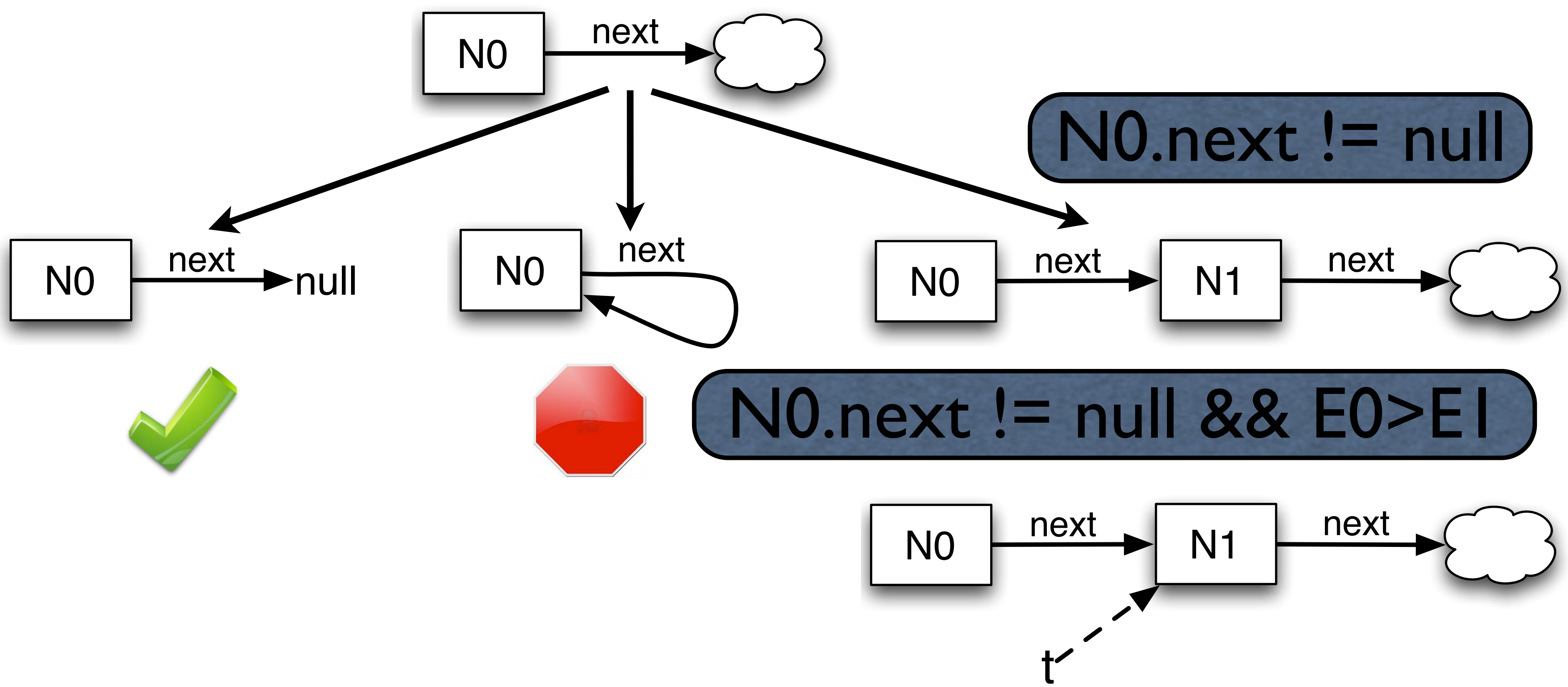
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

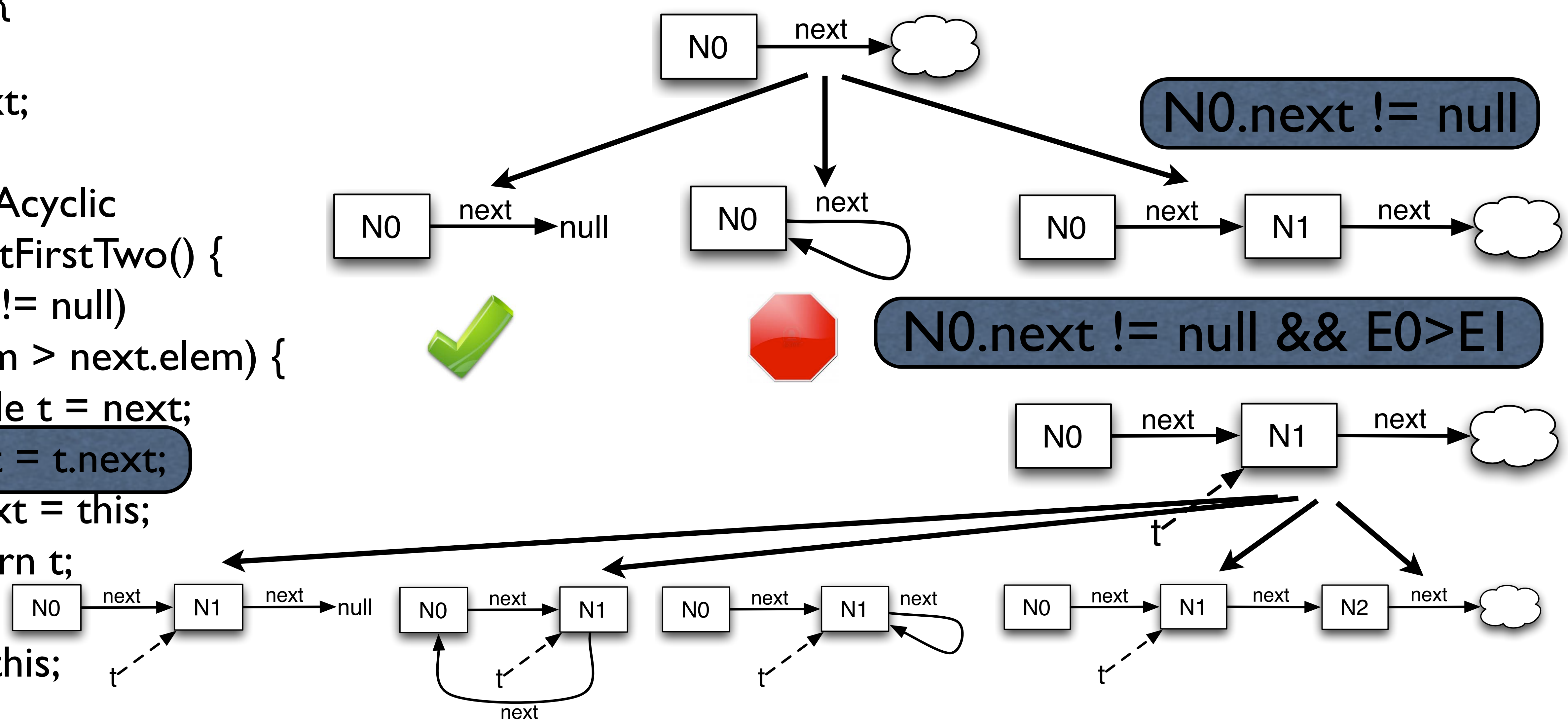
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

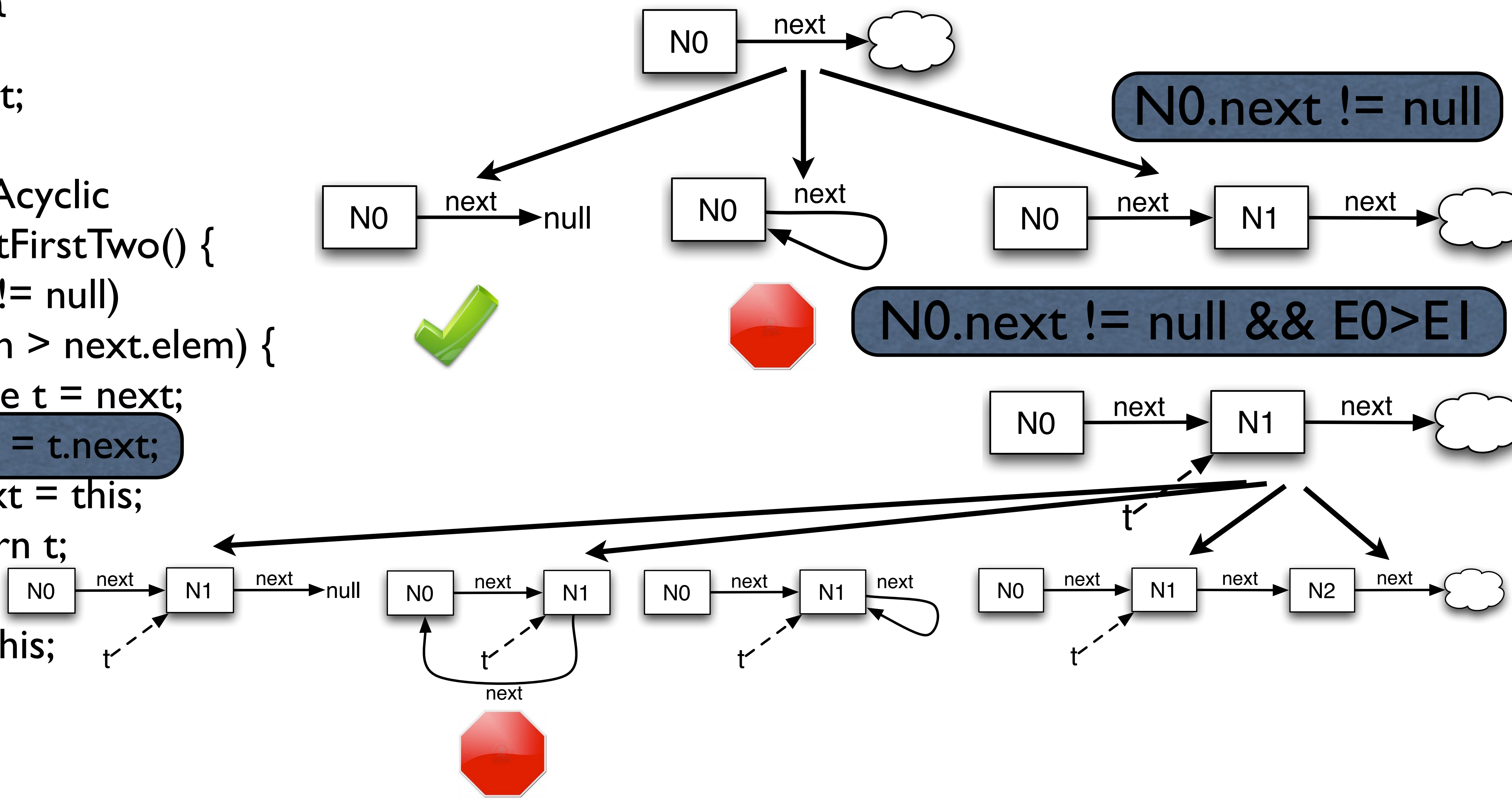
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

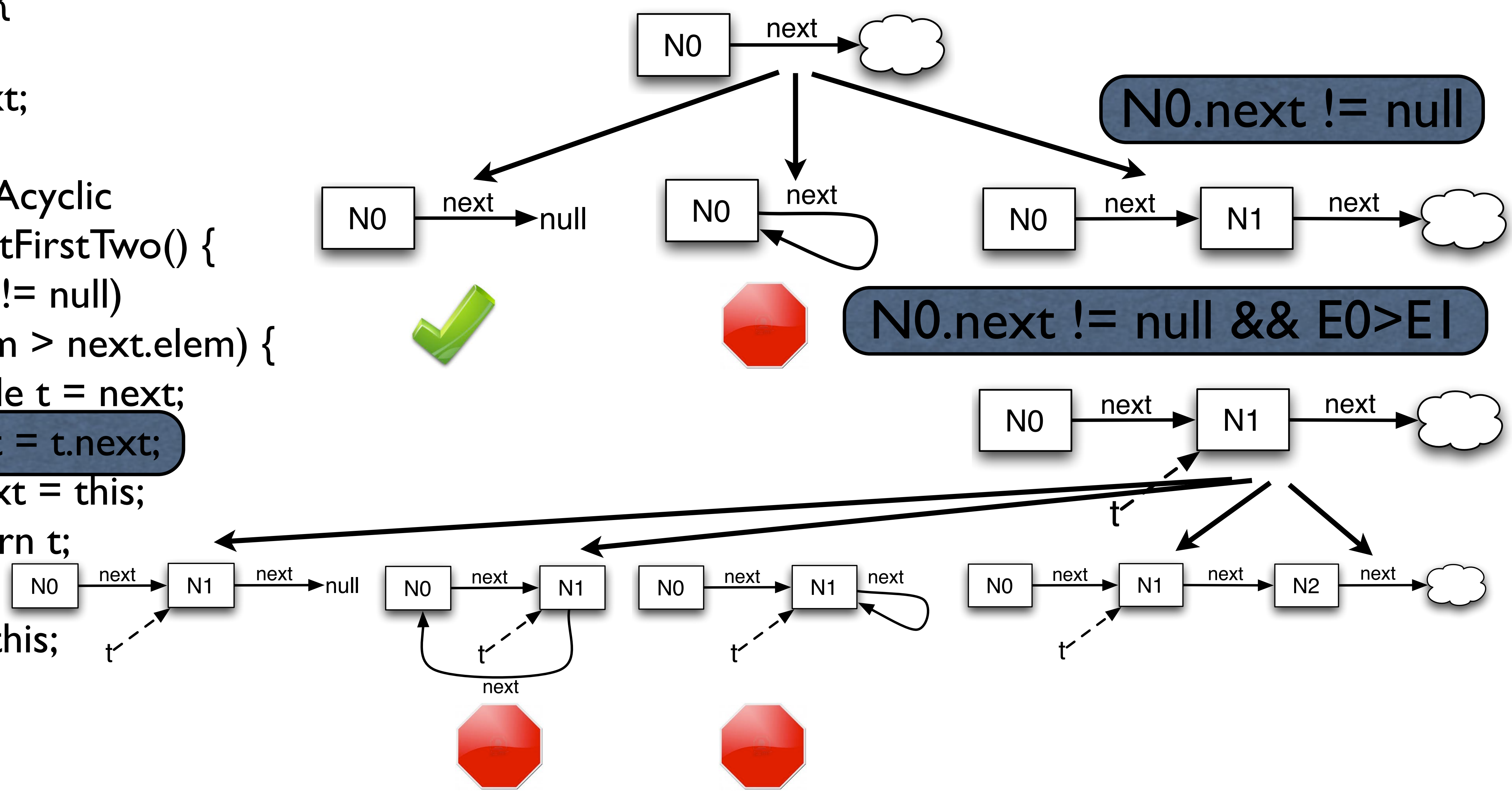
```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



Lazy Initialization (LI)

An example

```
class Node {
  int elem;
  Node next;

  \requires Acyclic
  Node sortFirstTwo() {
    if (next != null)
      if (elem > next.elem) {
        Node t = next;
        next = t.next;
        t.next = this;
        return t;
      }
    return this;
  }
}
```



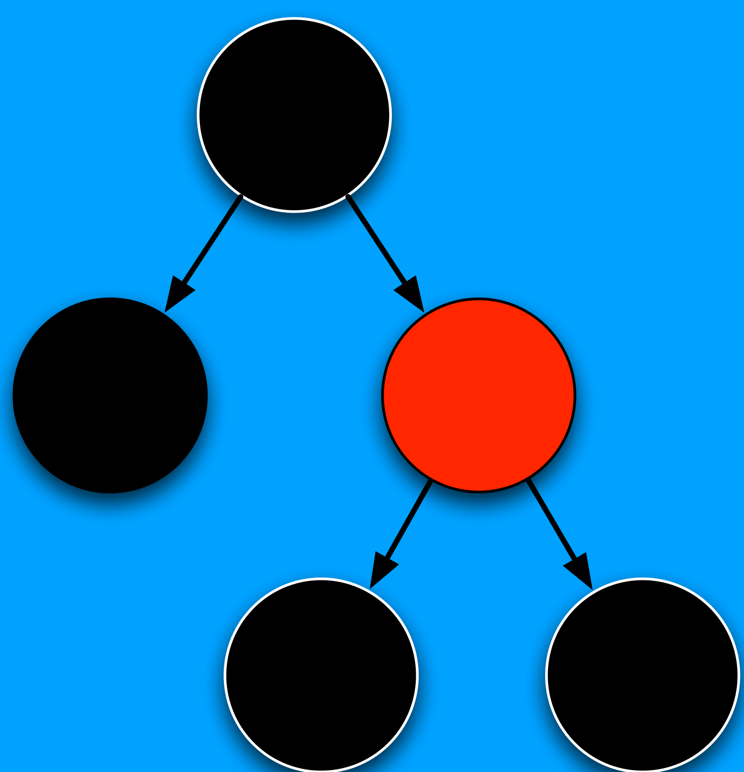
Lazy Initialization (LI)

Remarks

- Lazy Initialization does not generate isomorphic heaps.
- Constraints must be adapted so that they can be evaluated on partially symbolic heaps.

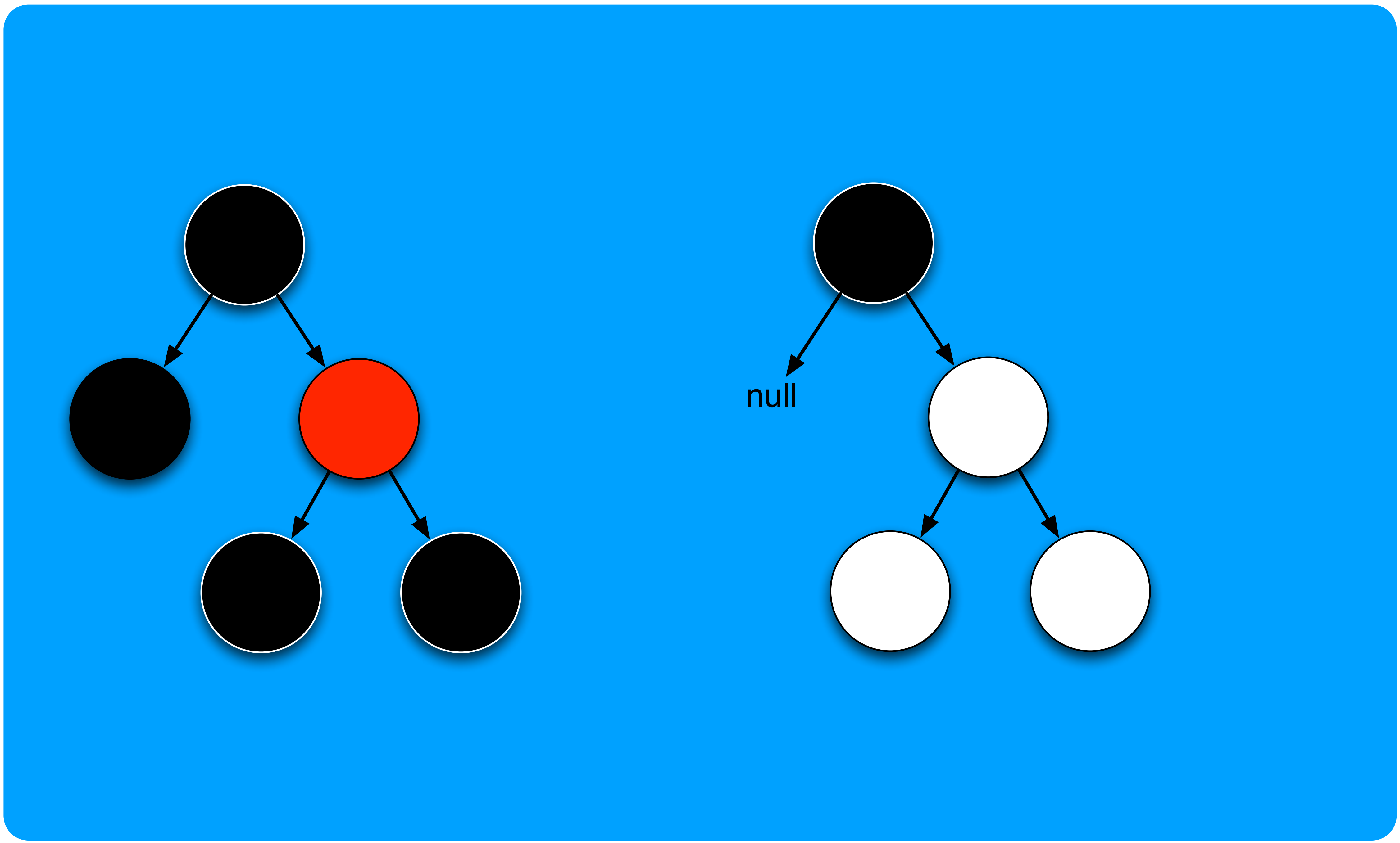
Symbolic Execution

For Dynamically Allocated Structures



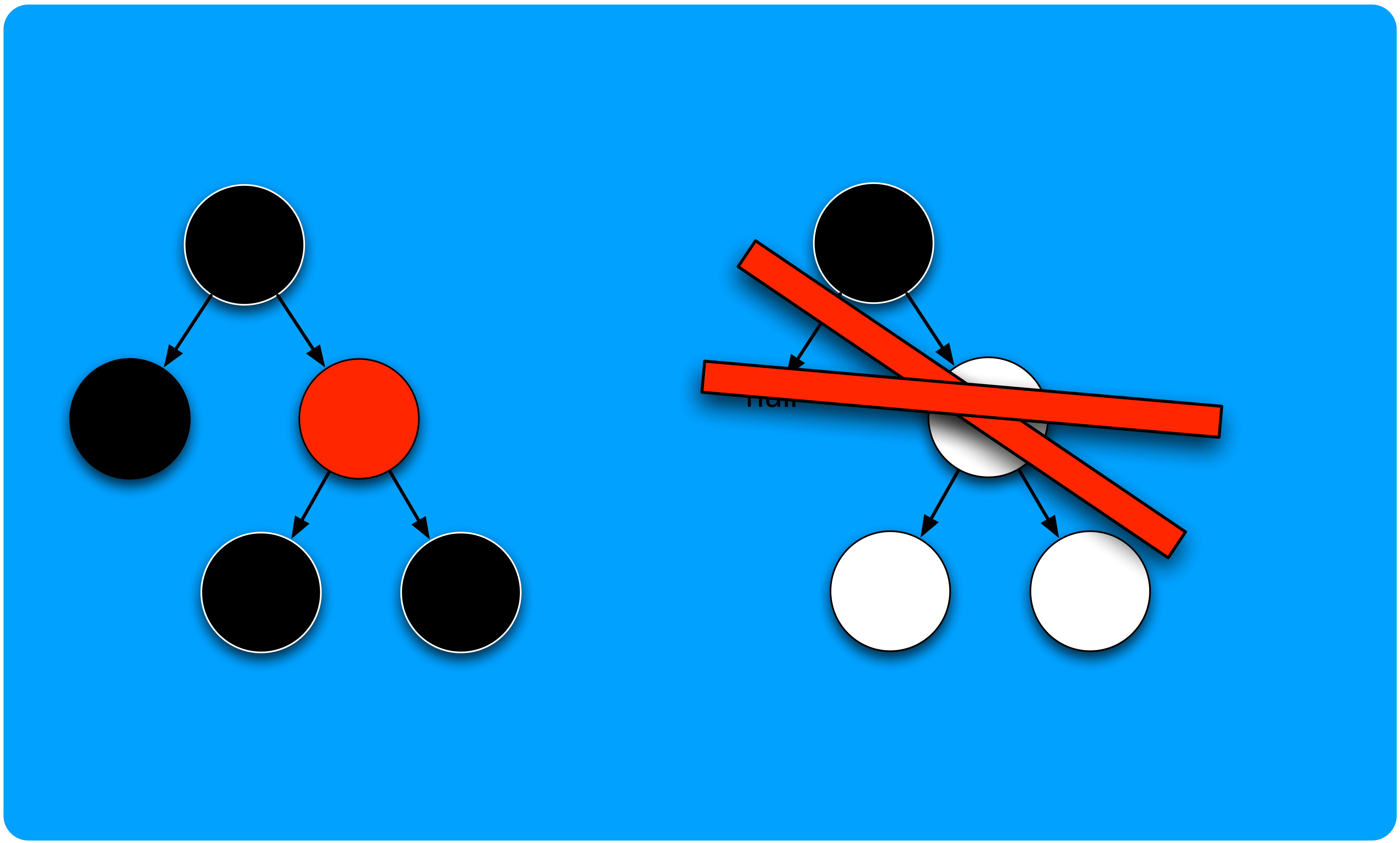
Symbolic Execution

For Dynamically Allocated Structures



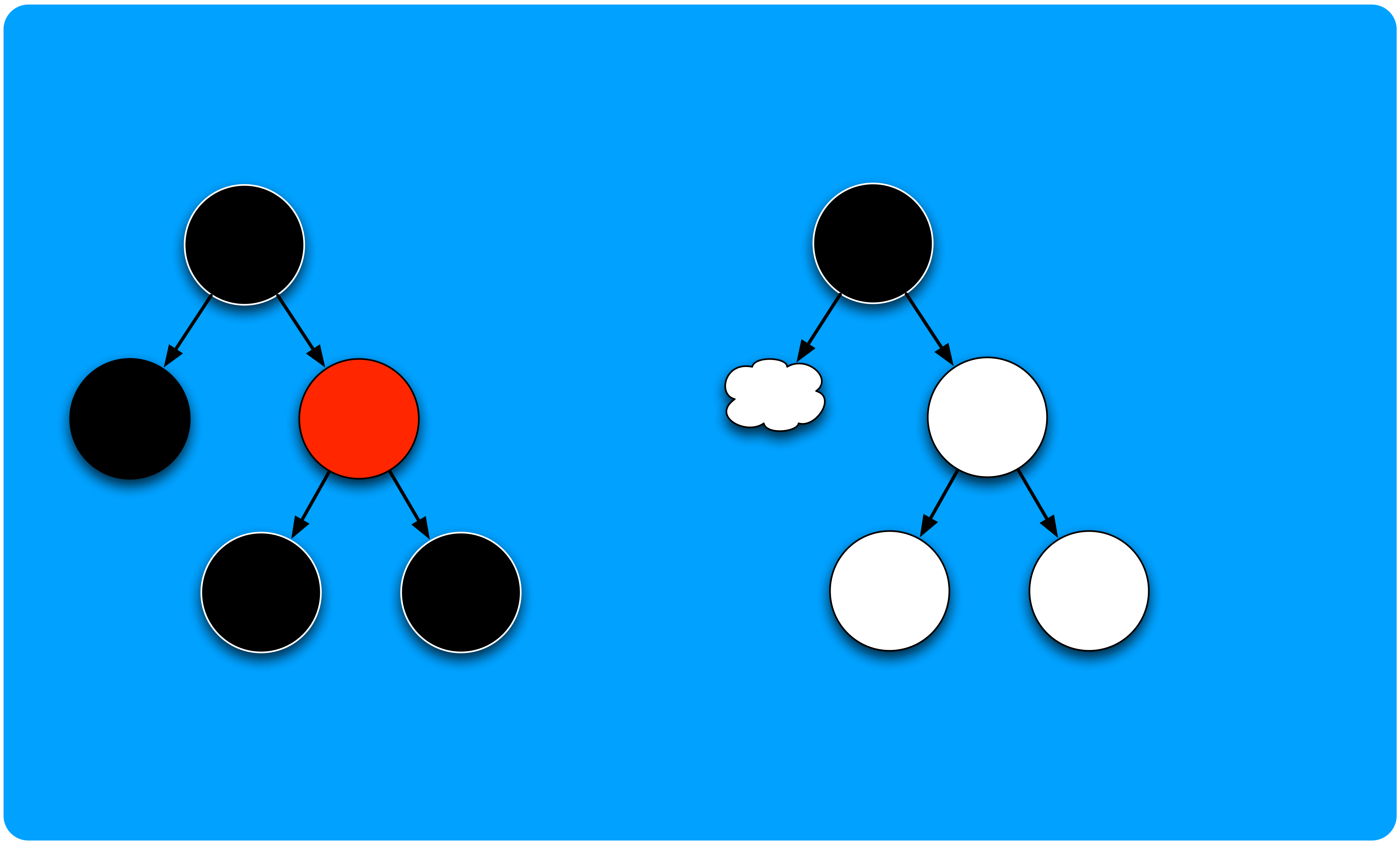
Symbolic Execution

For Dynamically Allocated Structures



Symbolic Execution

For Dynamically Allocated Structures



NASA's Symbolic Pathfinder Demo

Tight Field Bounds

Tight Field Bounds (1)

Intuition: propositional variables encode de relationship between objects in the memory heap, i.e., for each triple (O1, f, O2), where O1, O2 are objects and f is a field, propositional variable V(O1,f,O2) is true iff O1.f = O2.

Tight Field Bounds (2)

Example

If we use a scope of 3 nodes N0, N1, N2 in a data structure with a single field f, we will have variables:

V(N0,f,N0), V(N0,f,N1), V(N0,f,N2),
V(N1,f,N0), V(N1,f,N1), V(N1,f,N2),
V(N2,f,N0), V(N2,f,N1), V(N2,f,N2)

Tight Field Bounds (3)

The role of the SAT-Solver

The solver, while exploring the valuations for these variables, actually searches for a memory heap in which the program violates its contract.

V(N0,f,N0), V(N0,f,N1), V(N0,f,N2),
V(N1,f,N0), V(N1,f,N1), V(N1,f,N2),
V(N2,f,N0), V(N2,f,N1), V(N2,f,N2)

Tight Field Bounds (4)

Even more unfeasible valuations

If in any list instance node identifiers are chose from the head following the order N0, N1, N2,

~~V(N0,next,N0)~~, V(N0,next,N1), V(N0,next,N2),
V(N1,next,N0), ~~V(N1,next,N1)~~, V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (4)

Even more unfeasible valuations

If in any list instance node identifiers are chose from the head following the order N0, N1, N2,

V(N0,next,N0), V(N0,next,N1), V(N0,next,N2),
V(N1,next,N0), V(N1,next,N1), V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), V(N2,next,N2)

Tight Field Bounds (4)

Even more unfeasible valuations

If in any list instance node identifiers are chose from the head following the order N0, N1, N2,

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
~~V(N1,next,N0)~~, ~~V(N1,next,N1)~~, V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (4)

Even more unfeasible valuations

If in any list instance node identifiers are chose from the head following the order N0, N1, N2,

V(N0,next,N0), V(N0,next,N1), V(N0,next,N2),
V(N1,next,N0), V(N1,next,N1), V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), V(N2,next,N2)

Tight Field Bounds (4)

Even more unfeasible valuations

If in any list instance node identifiers are chose from the head following the order N0, N1, N2,

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
~~V(N1,next,N0)~~, ~~V(N1,next,N1)~~, V(N1,next,N2),
~~V(N2,next,N0)~~, ~~V(N2,next,N1)~~, ~~V(N2,next,N2)~~

Tight Field Bounds (4)

Even more unfeasible valuations

Symmetry-breaking predicates

If in any list in … are chose

from th… or N0, N1, N2,

V(N0,n… ,next,N1), V(N0,next,N2),
V(N1,ne…), V(N1,next,N1), V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), V(N2,next,N2)

Tight Field Bounds (5)

What's cool about unfeasible valuations?

Variables that are forced to be false can actually be removed and be substituted by their known value (false). In this way the SAT-solver has less work to do (exponential speedup).

~~V(N0,next,N0)~~, V(N0,next,N1), V(N0,next,N2),
V(N1,next,N0), ~~V(N1,next,N1)~~, V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (5)

What's cool about unfeasible valuations?

Variables that are forced to be false can actually be removed and be substituted by their known value (false). In this way the SAT-solver has less work to do (exponential speedup).

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
V(N1,next,N0), ~~V(N1,next,N1)~~, V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (5)

What's cool about unfeasible valuations?

Variables that are forced to be false can actually be removed and be substituted by their known value (false). In this way the SAT-solver has less work to do (exponential speedup).

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
~~V(N1,next,N0)~~, ~~V(N1,next,N1)~~, V(N1,next,N2),
V(N2,next,N0), V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (5)

What’s cool about unfeasible valuations?

Variables that are forced to be false can actually be removed and be substituted by their known value (false). In this way the SAT-solver has less work to do (exponential speedup).

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
~~V(N1,next,N0)~~, ~~V(N1,next,N1)~~, V(N1,next,N2),
~~V(N2,next,N0)~~, V(N2,next,N1), ~~V(N2,next,N2)~~

Tight Field Bounds (5)

What’s cool about unfeasible valuations?

Variables that are forced to be false can actually be removed and be substituted by their known value (false). In this way the SAT-solver has less work to do (exponential speedup).

~~V(N0,next,N0)~~, V(N0,next,N1), ~~V(N0,next,N2)~~,
~~V(N1,next,N0)~~, ~~V(N1,next,N1)~~, V(N1,next,N2),
~~V(N2,next,N0)~~, ~~V(N2,next,N1)~~, ~~V(N2,next,N2)~~

Tight Field Bounds (6)

Definition

Given object identifiers N0,…,Ni and a field f, the tight field bound tfb(f) is a minimal binary relation on N0,…,Ni,null such that (Ni, Nj) in tfb(f) iff there exists a valid memory heap satisfying the symmetry-breaking predicates in which Ni.f = Nj (idem for null).

Tight Field Bounds (7)

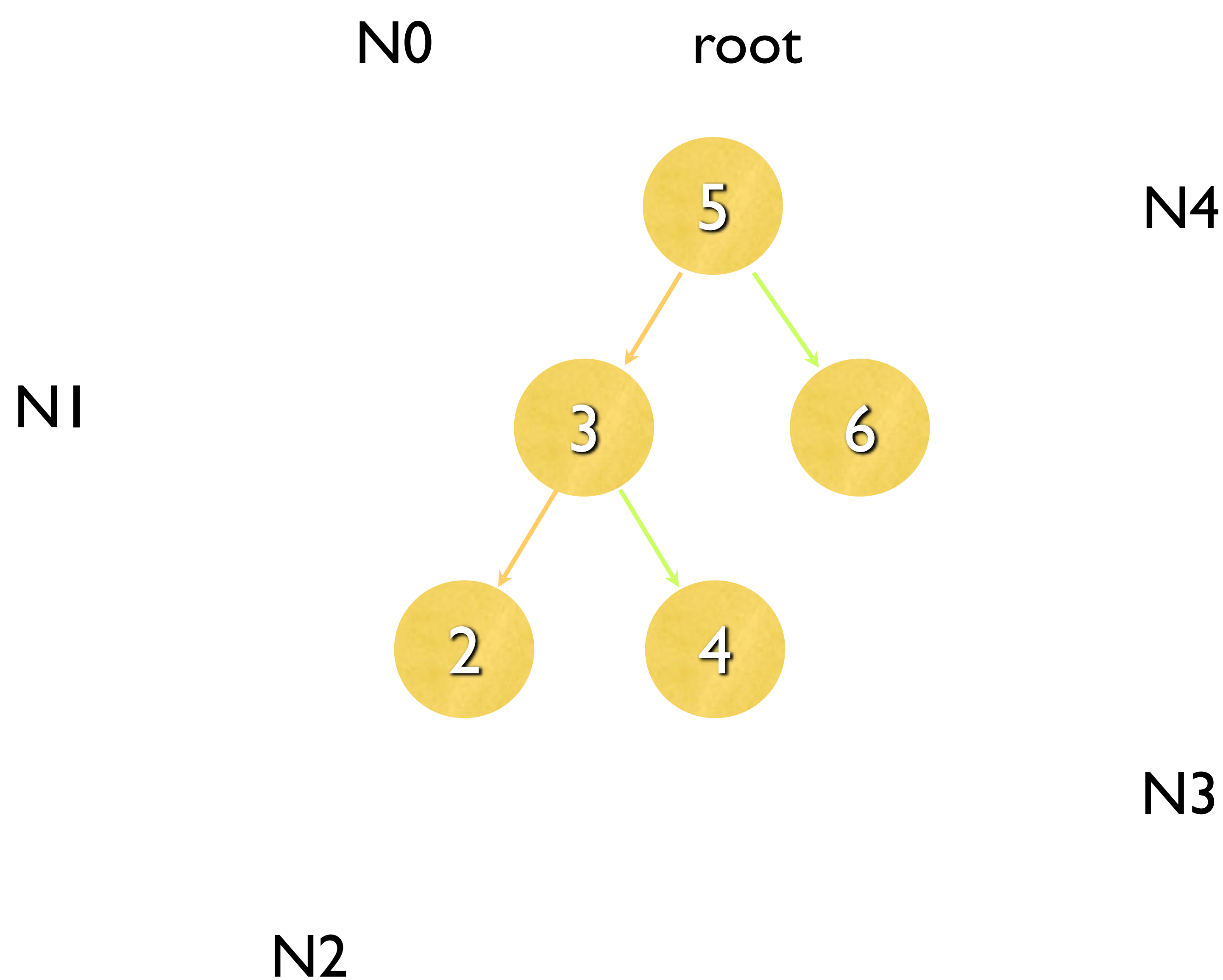
Valid

Satisfying the representation invariant (AVL? Red-Black Tree? BinHeap?)

Tight Field Bounds (7)

Satisfying Symmetry-Breaking

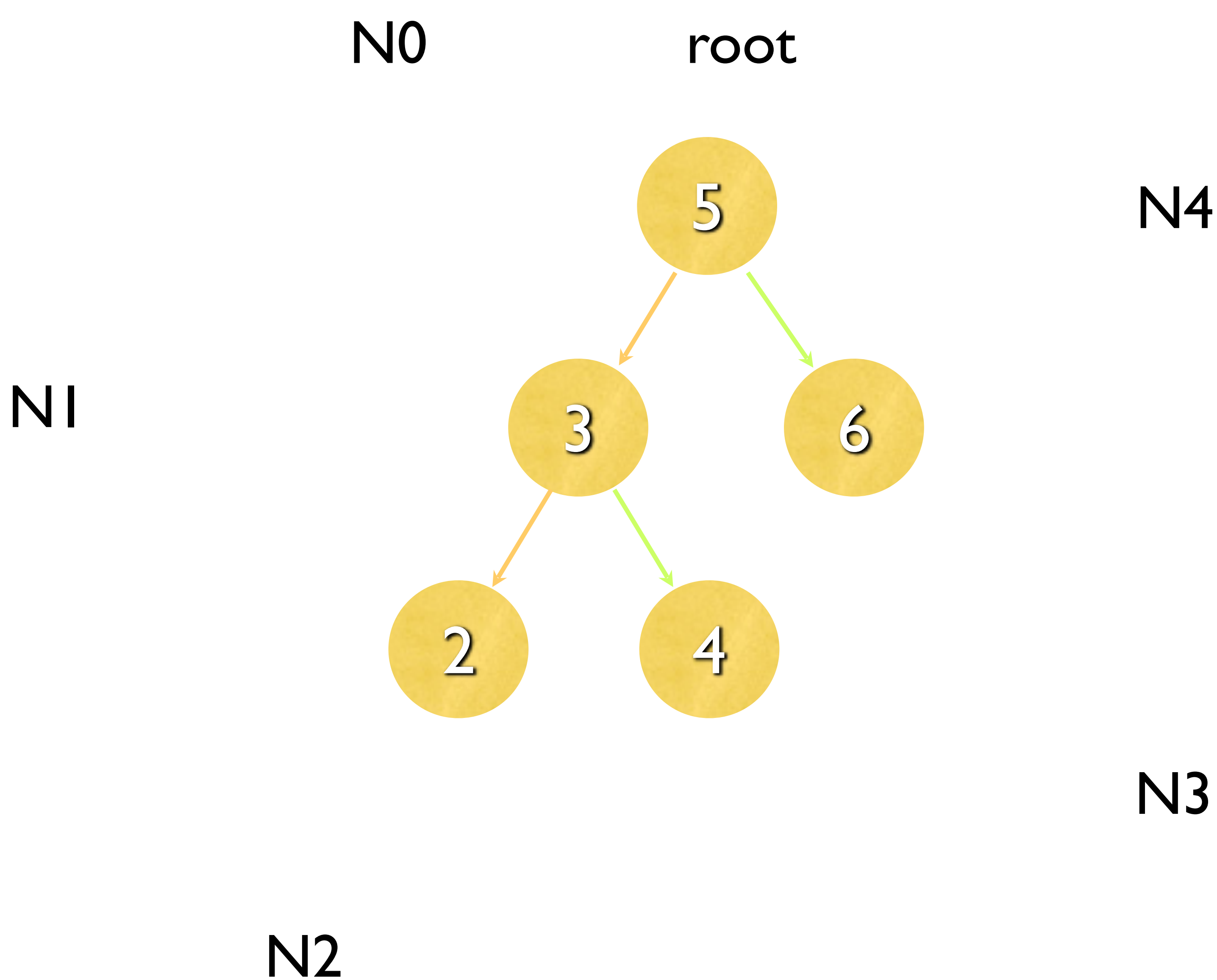
Object identifiers in the heap follow a canonical ordering: BFS-traversal.



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

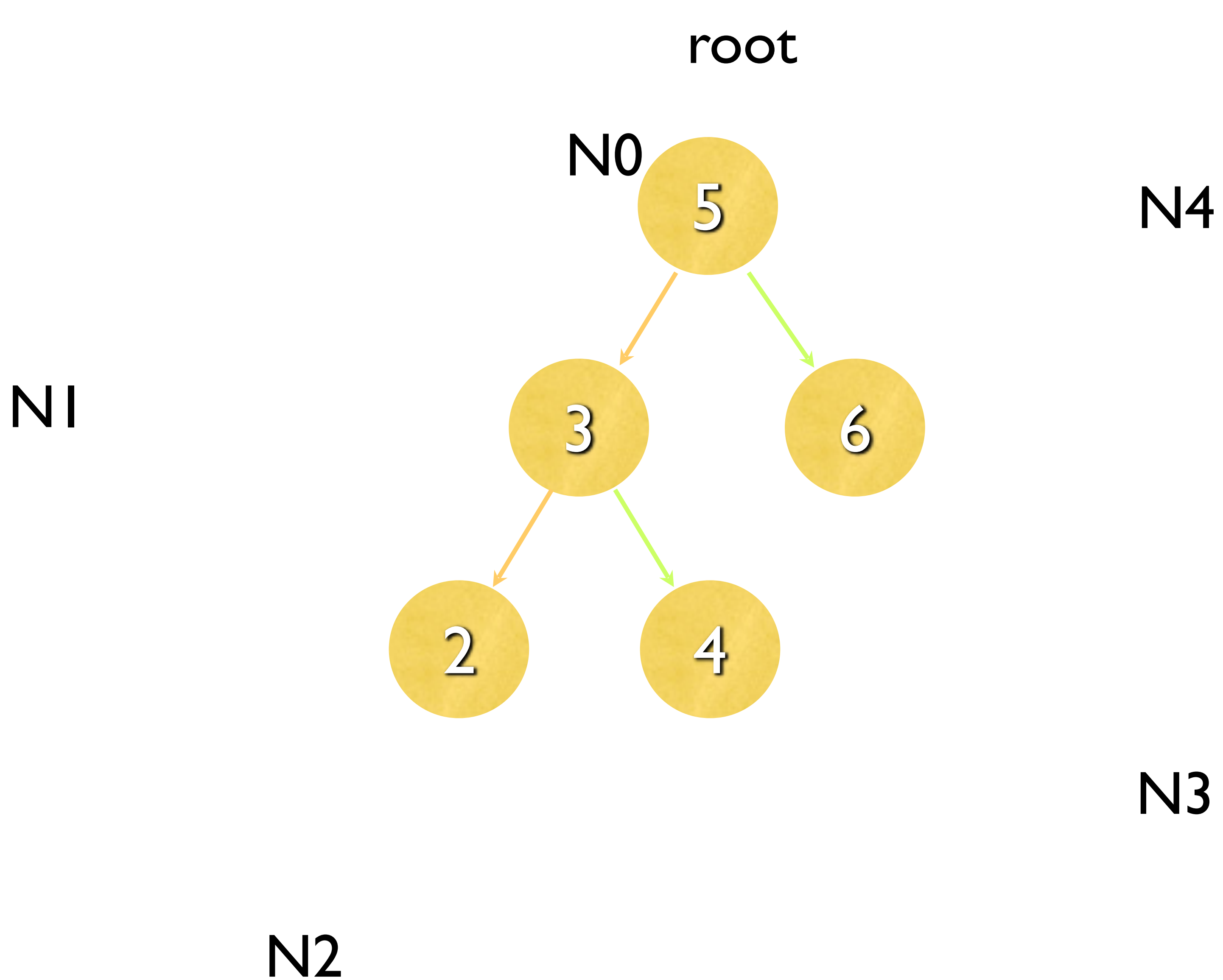
Rule1: The graph's root is labeled N0.



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

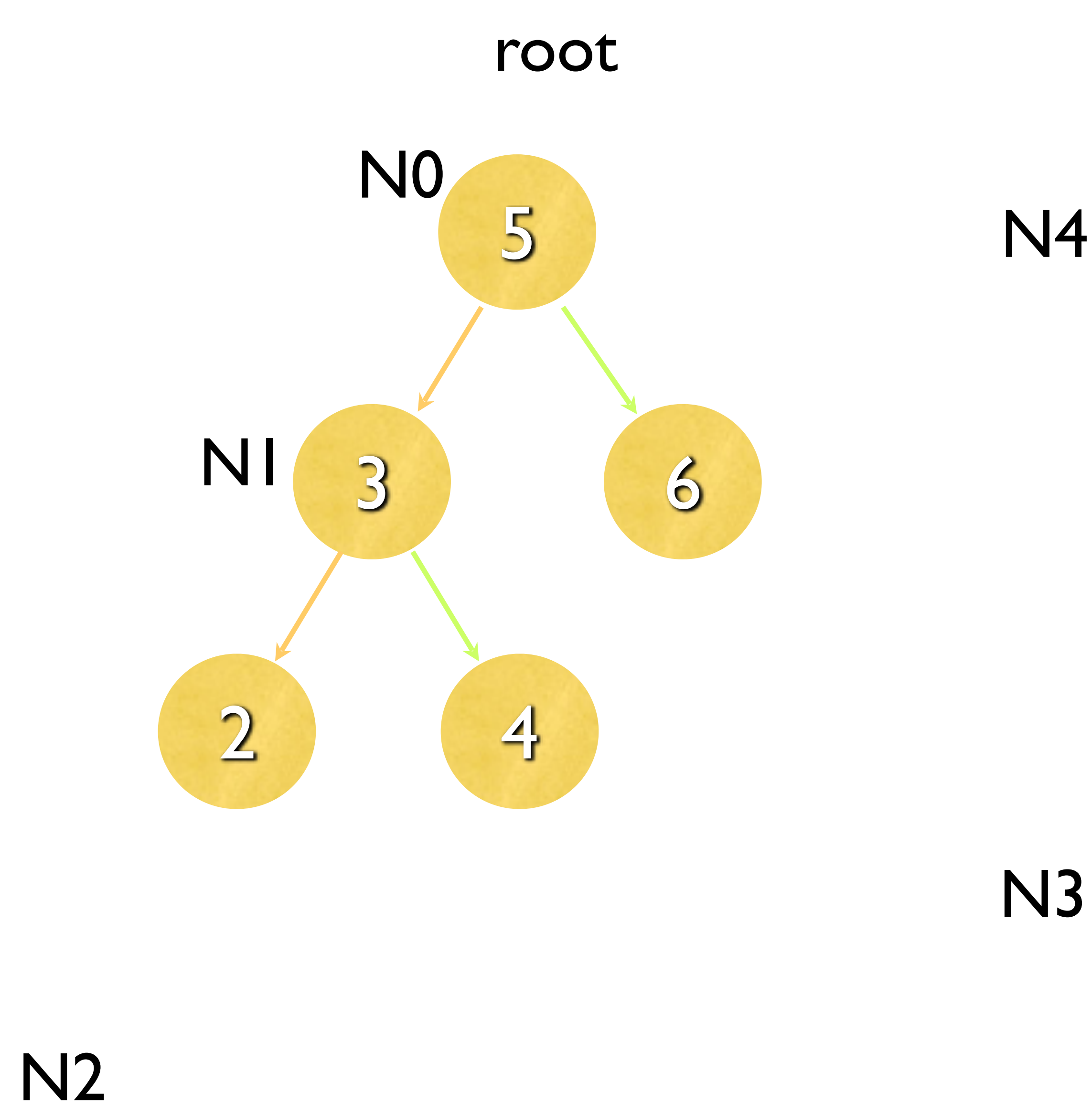
Rule2: Two nodes with the same parent are labeled from left to right.



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

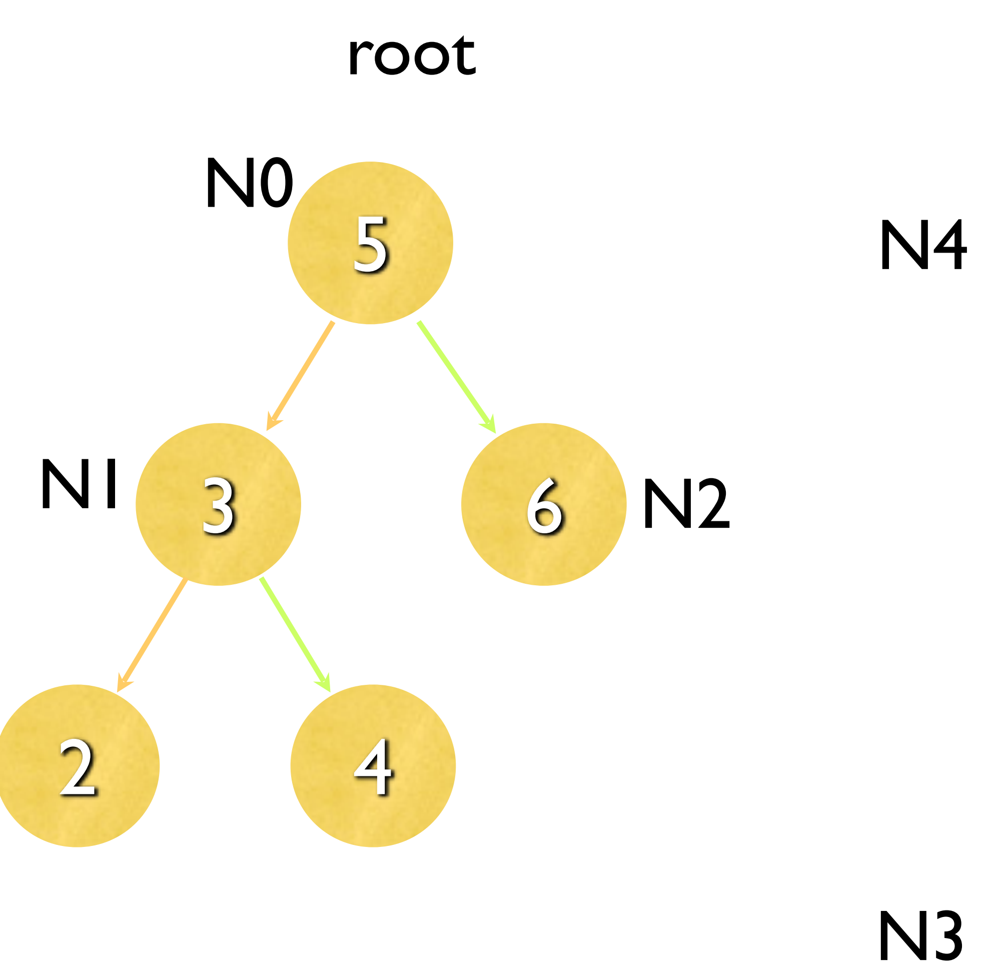
Rule2: Two nodes with the same parent are labeled from left to right.



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

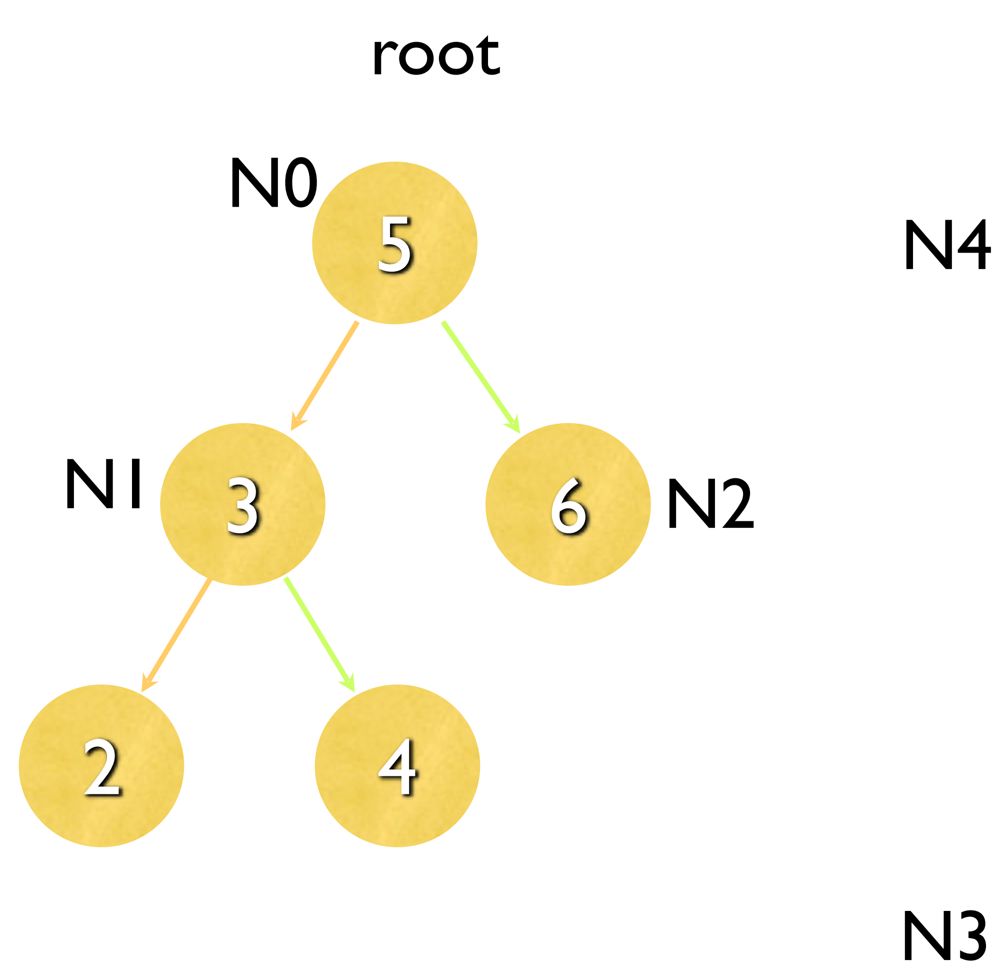
Rule2: Two nodes with the same parent are labeled from left to right.



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

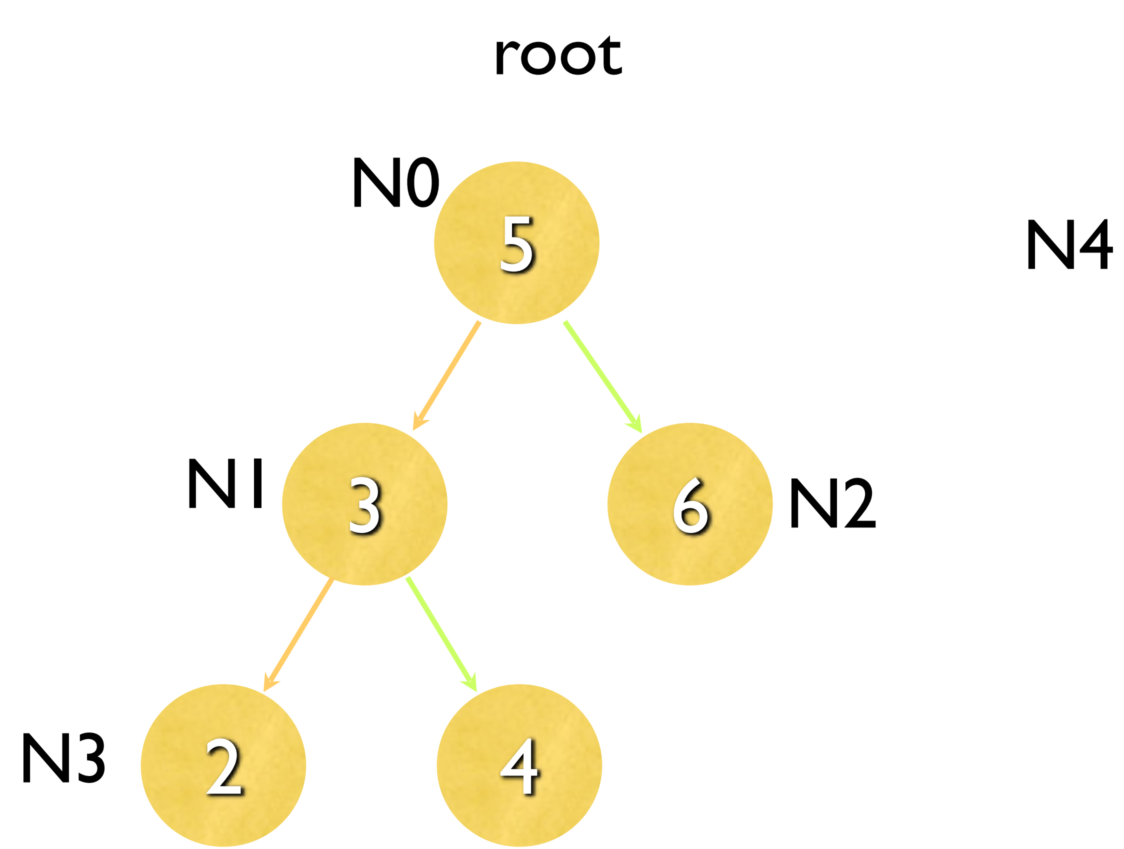
Applying once again Rule 2,



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

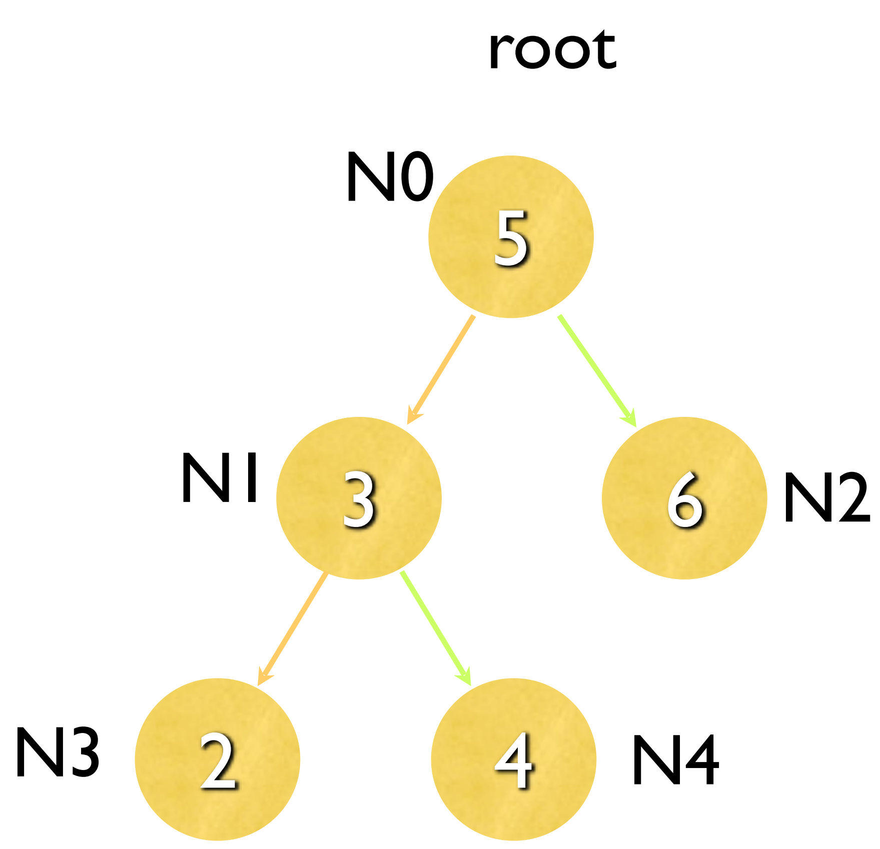
Applying once again Rule 2,



Tight Field Bounds (7)

Satisfying Symmetry-Breaking

Applying once again Rule 2,



Tight Field Bounds (8)

Computing Tight Field Bounds

There are two algorithms, a distributed and a sequential one. Will focus on the sequential.

Intuition: Incrementally ask the SAT-solver for valid heaps, each one contributing new pairs to the bound, until the solver says “UNSAT”.

left = {}

Tight Field Bounds (9)

Computing Tight Field Bounds: Example

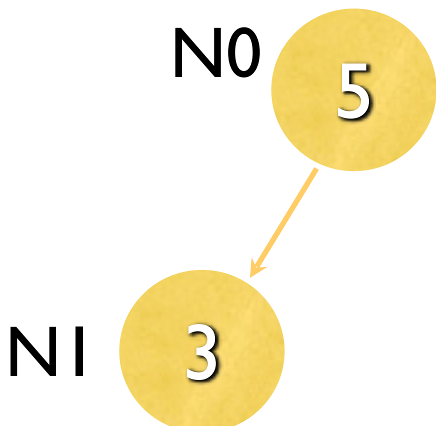
Consider AVL trees with up to 4 nodes, and fields left and right.

left = {}

Tight Field Bounds (9)

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

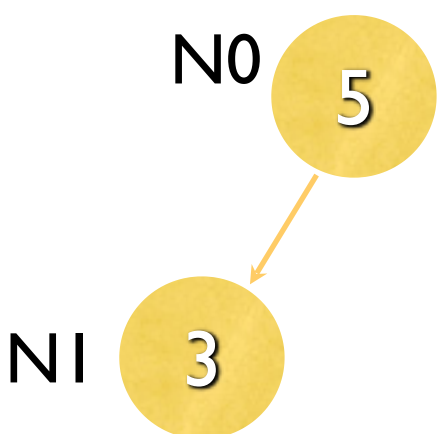


left = {(N0,N1)}

Tight Field Bounds (9)

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

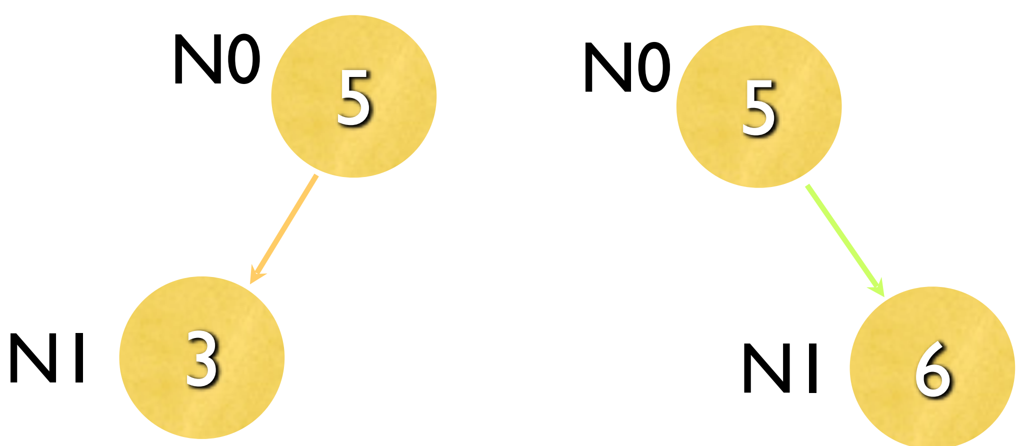


left = {(N0,N1)}

Tight Field Bounds (9)

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

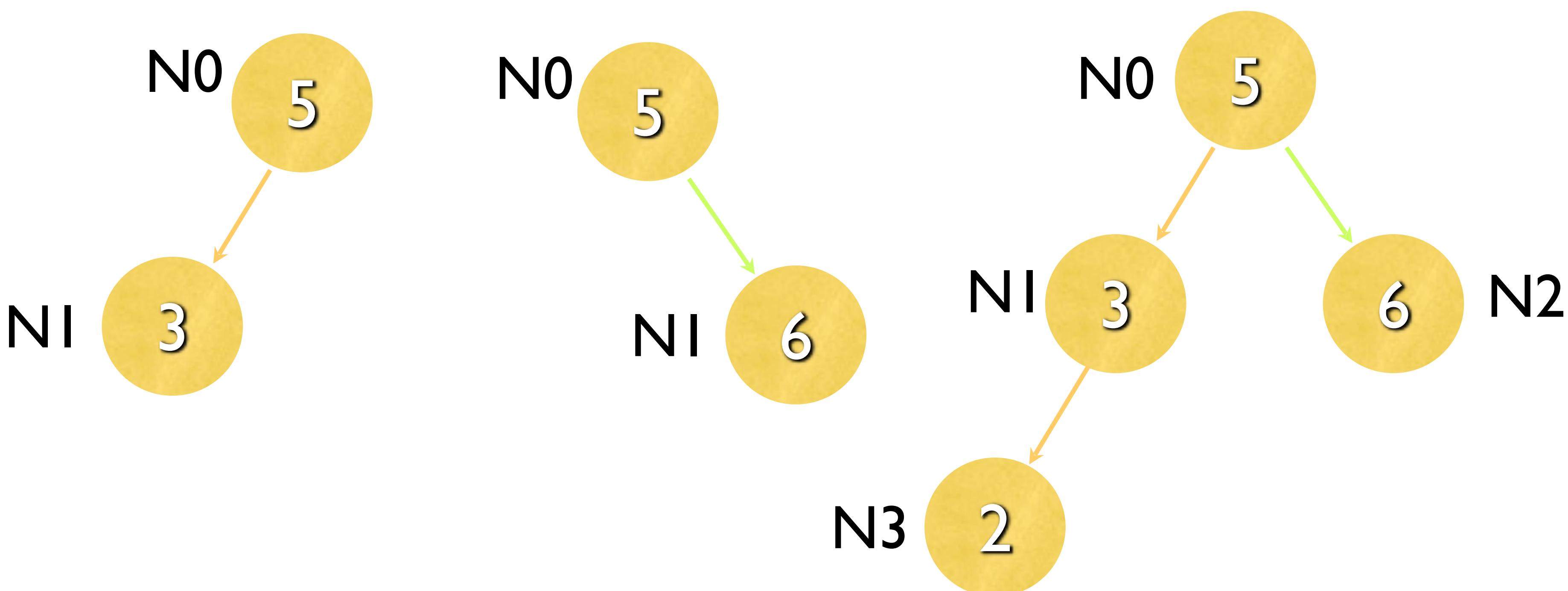


left = {(N0,N1)}

Tight Field Bounds (9)

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

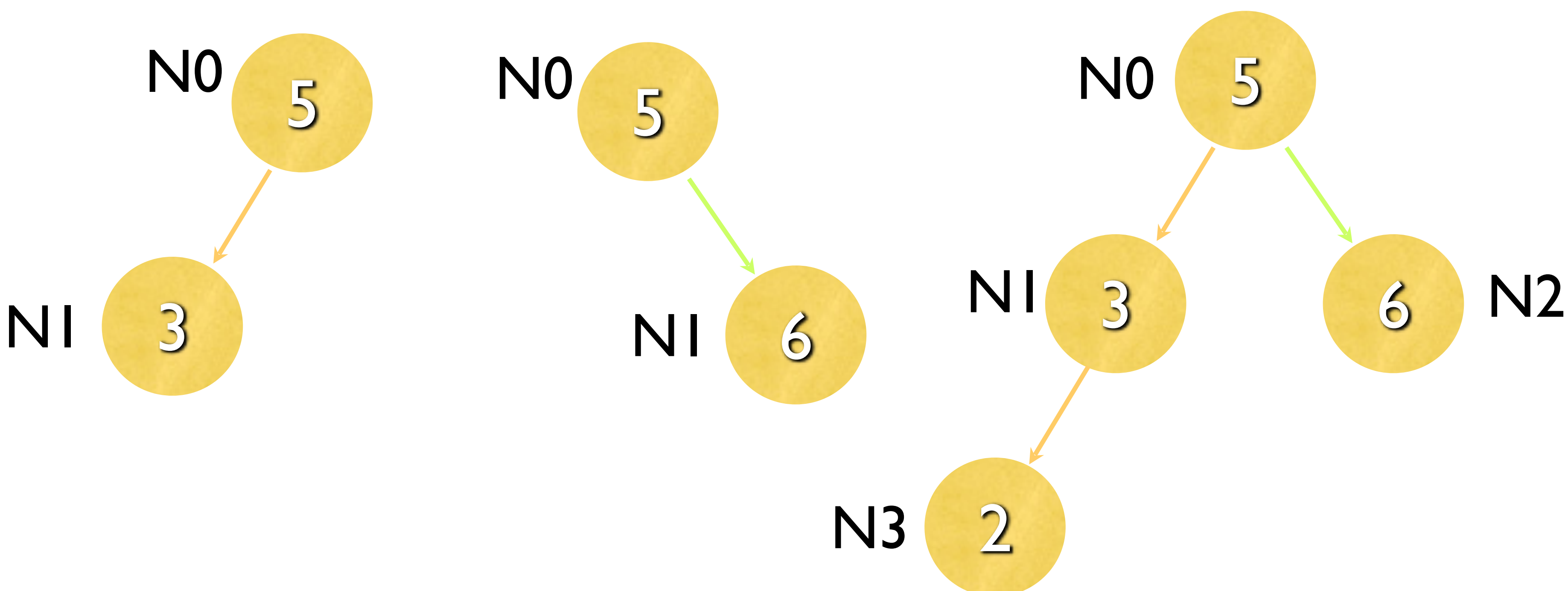


Tight Field Bounds (9)

left = {(N0,N1),
(N1,N3)}

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

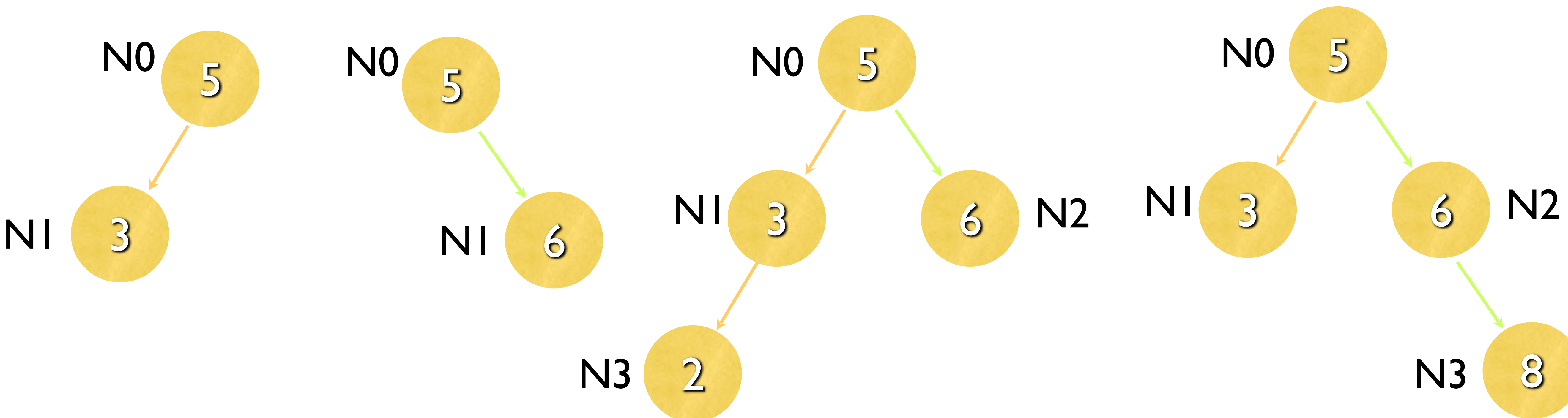


Tight Field Bounds (9)

left = {(N0,N1),
(N1,N3)}

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

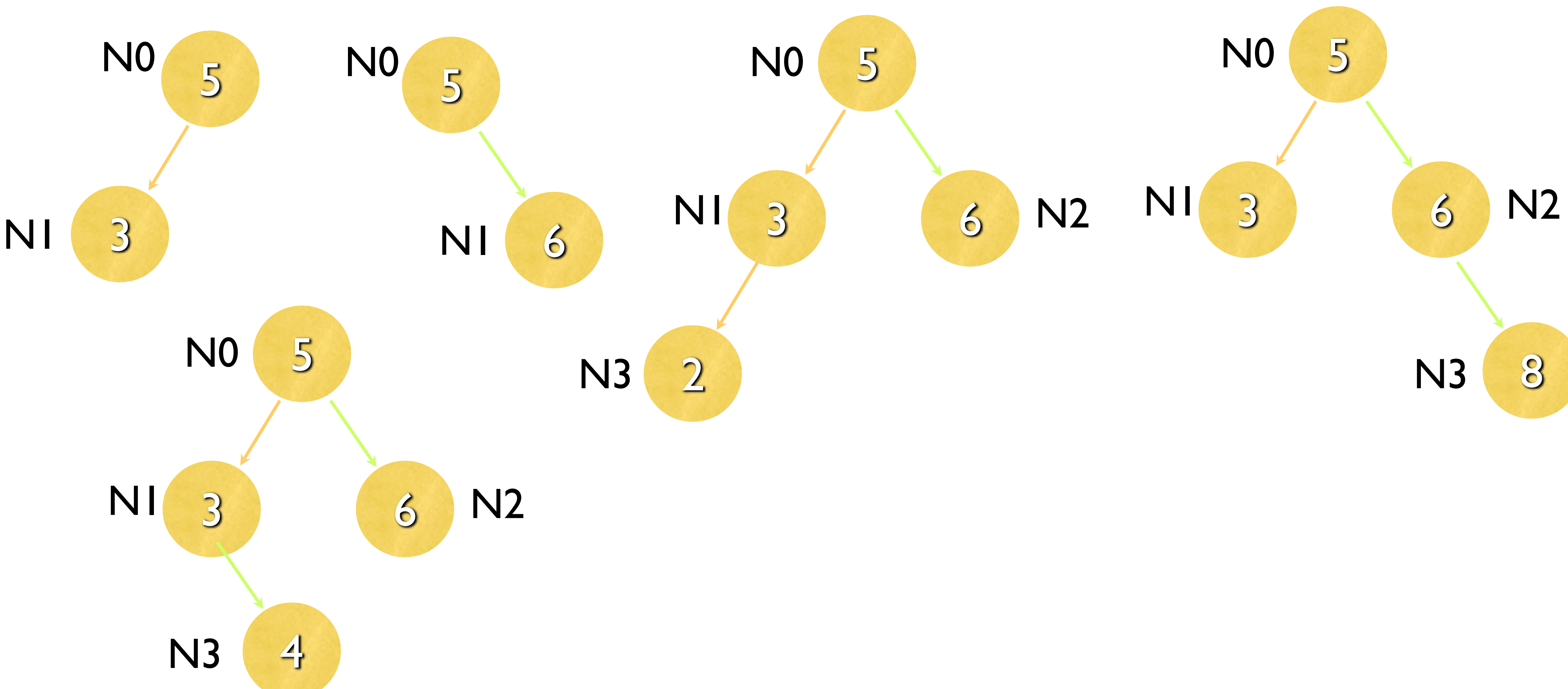


Tight Field Bounds (9)

left = {(N0,N1),
(N1,N3)}

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.

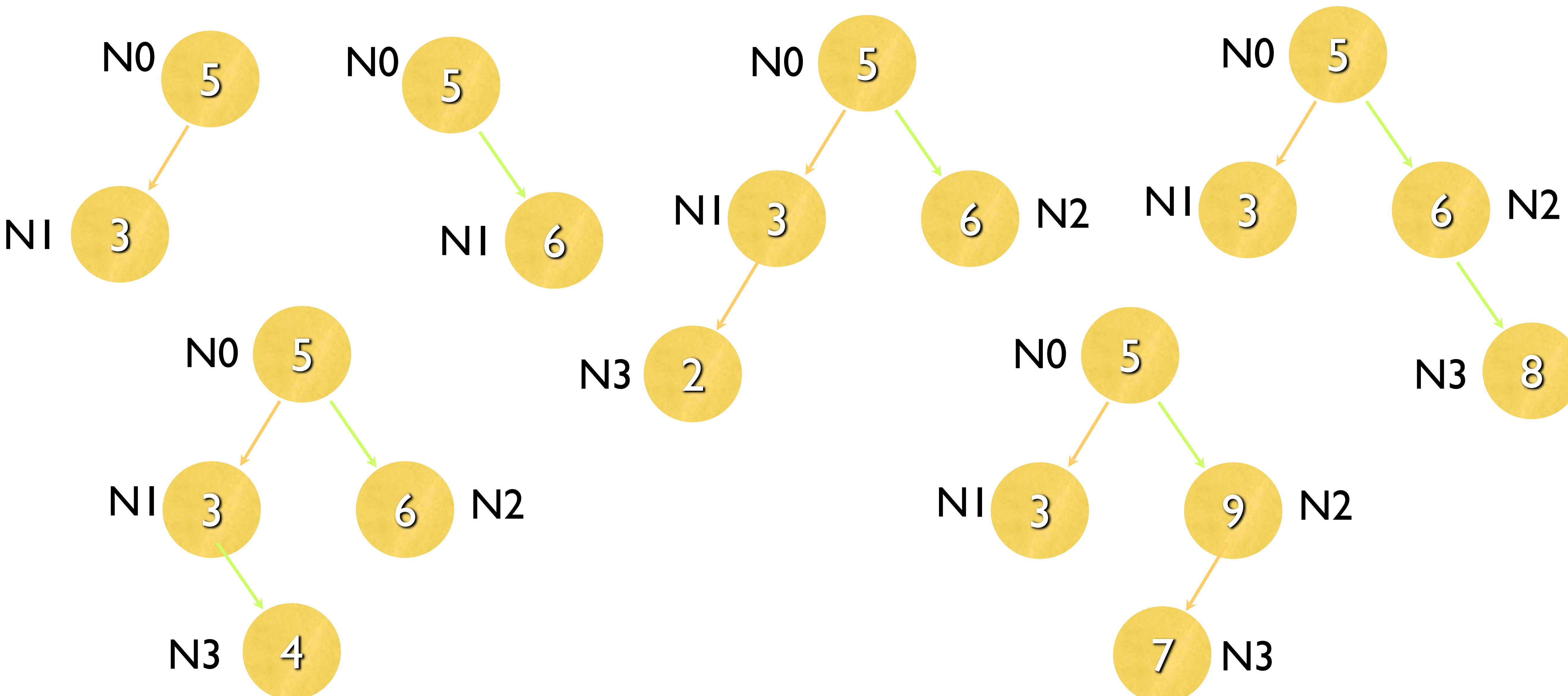


Tight Field Bounds (9)

left = {(N0,N1),
(N1,N3)}
(N2,N3)}

Computing Tight Field Bounds: Example

Consider AVL trees with up to 4 nodes, and fields left and right.



Tight Field Bounds in Bug Finding

Demo

- All variables that represent pairs of nodes that are not in the bound, are set to false on translation.
- In this way, the SAT-problem has fewer propositional variables.

Tight Field Bounds in Symbolic Execution

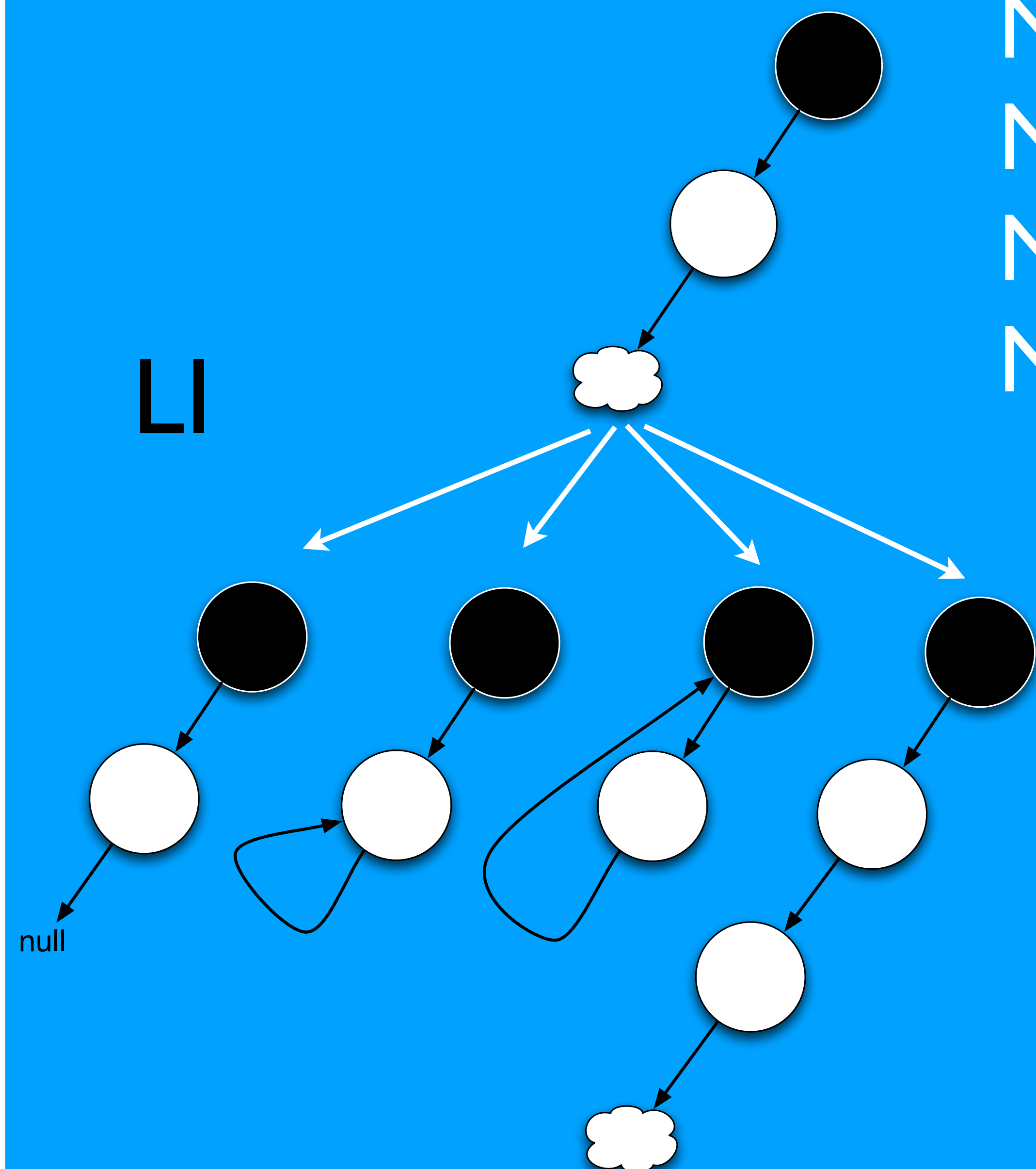
Intuition

- Pairs that are outside the tight field bounds are not used as options during lazy initialization.
- In this way, there are fewer cases to consider.

Nicolás Rosner, Jaco Geldenhuys, Nazareno Aguirre, Willem Visser, Marcelo F. Frias:
BLISS: Improved Symbolic Execution by Bounded Lazy Initialization with SAT Support. IEEE Trans. Software Eng. 41(7): 639-660 (2015)

Tight Field Bounds in Symbolic Execution

Intuition



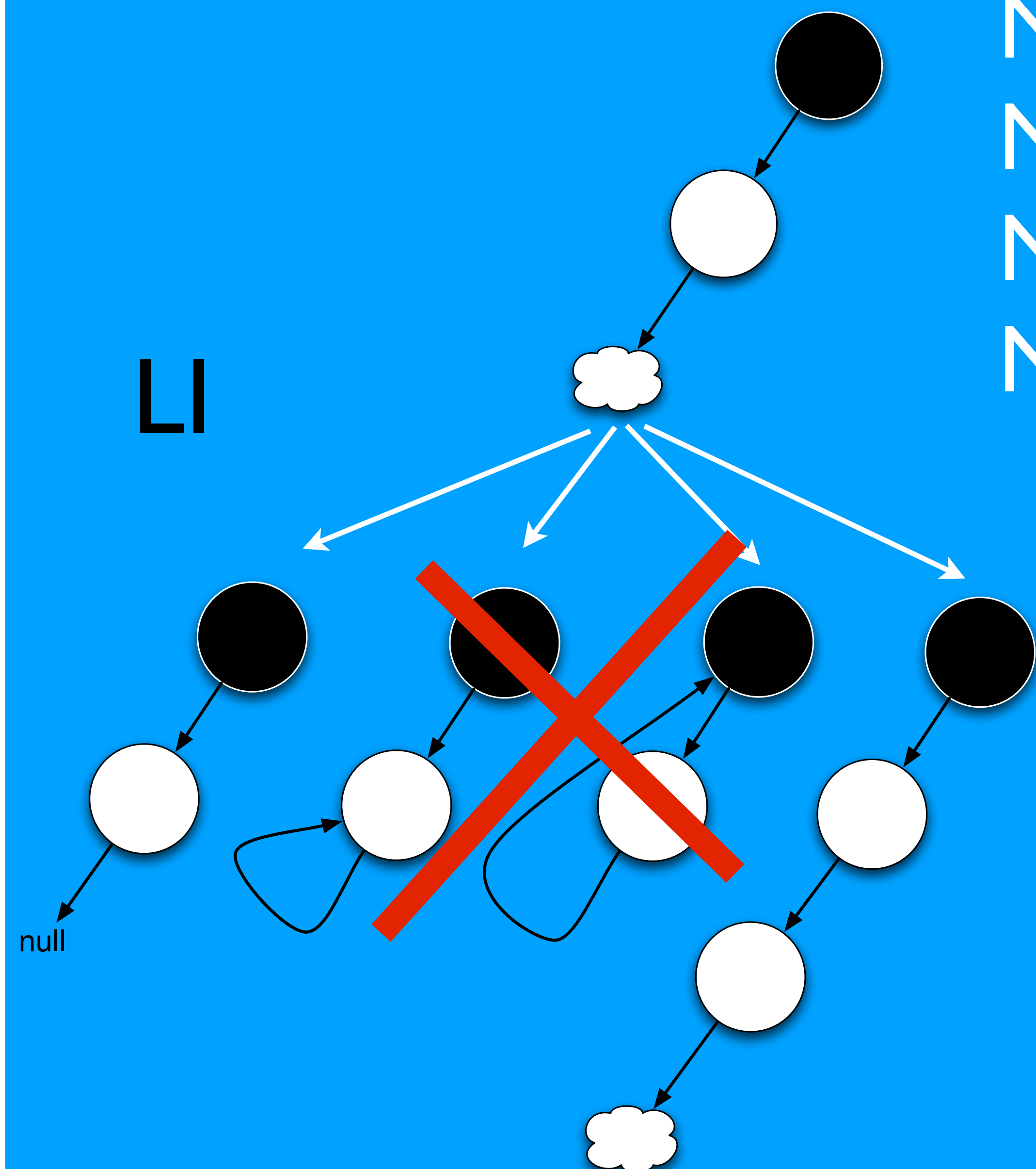
N0 ⟶ null, N1
N1 ⟶ null, N3
N2 ⟶ null, N3
N3 ⟶ null

Uleft

Tight Field Bounds in Symbolic Execution

Intuition

LI

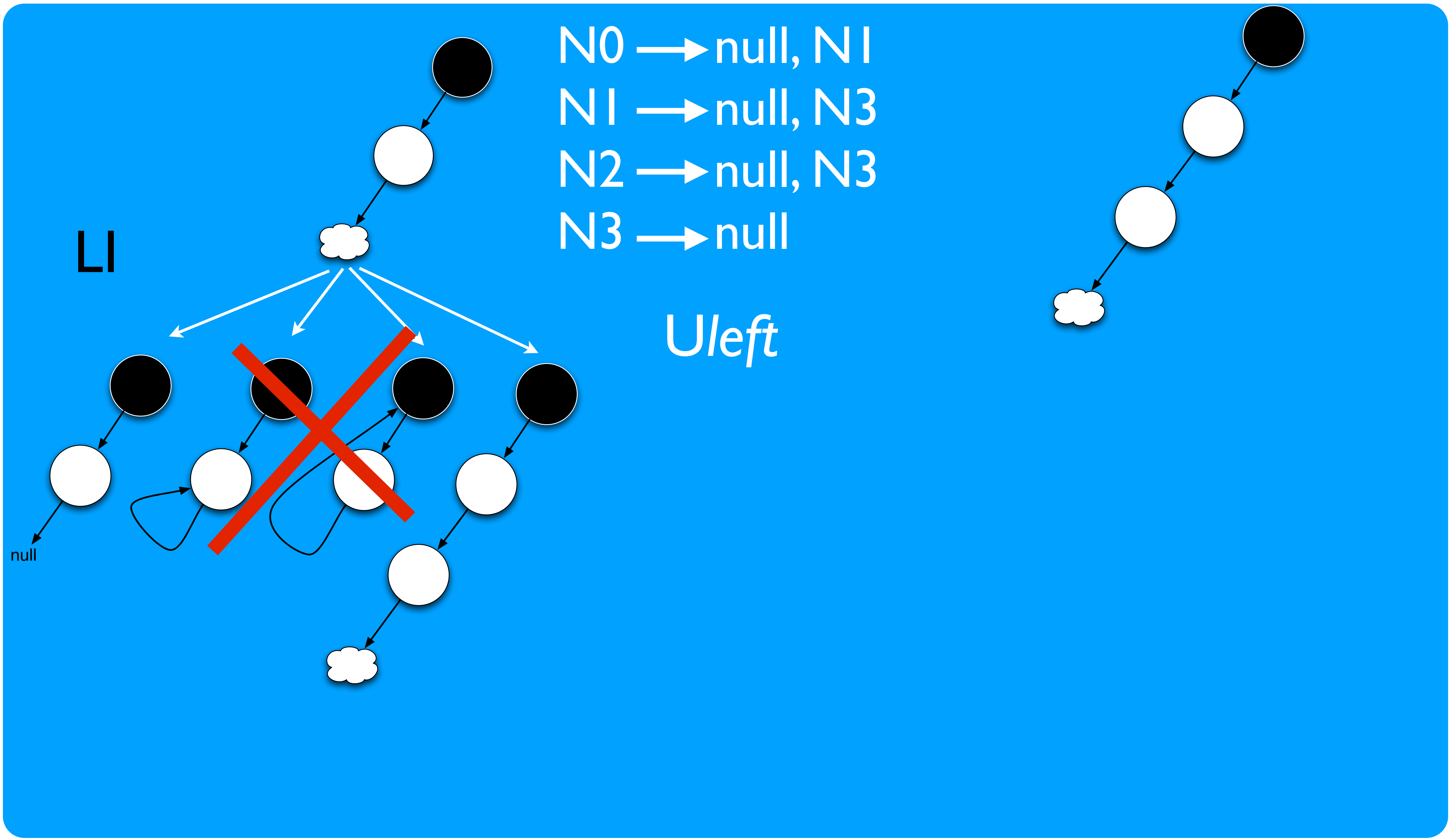


N0 → null, N1
N1 → null, N3
N2 → null, N3
N3 → null

Uleft

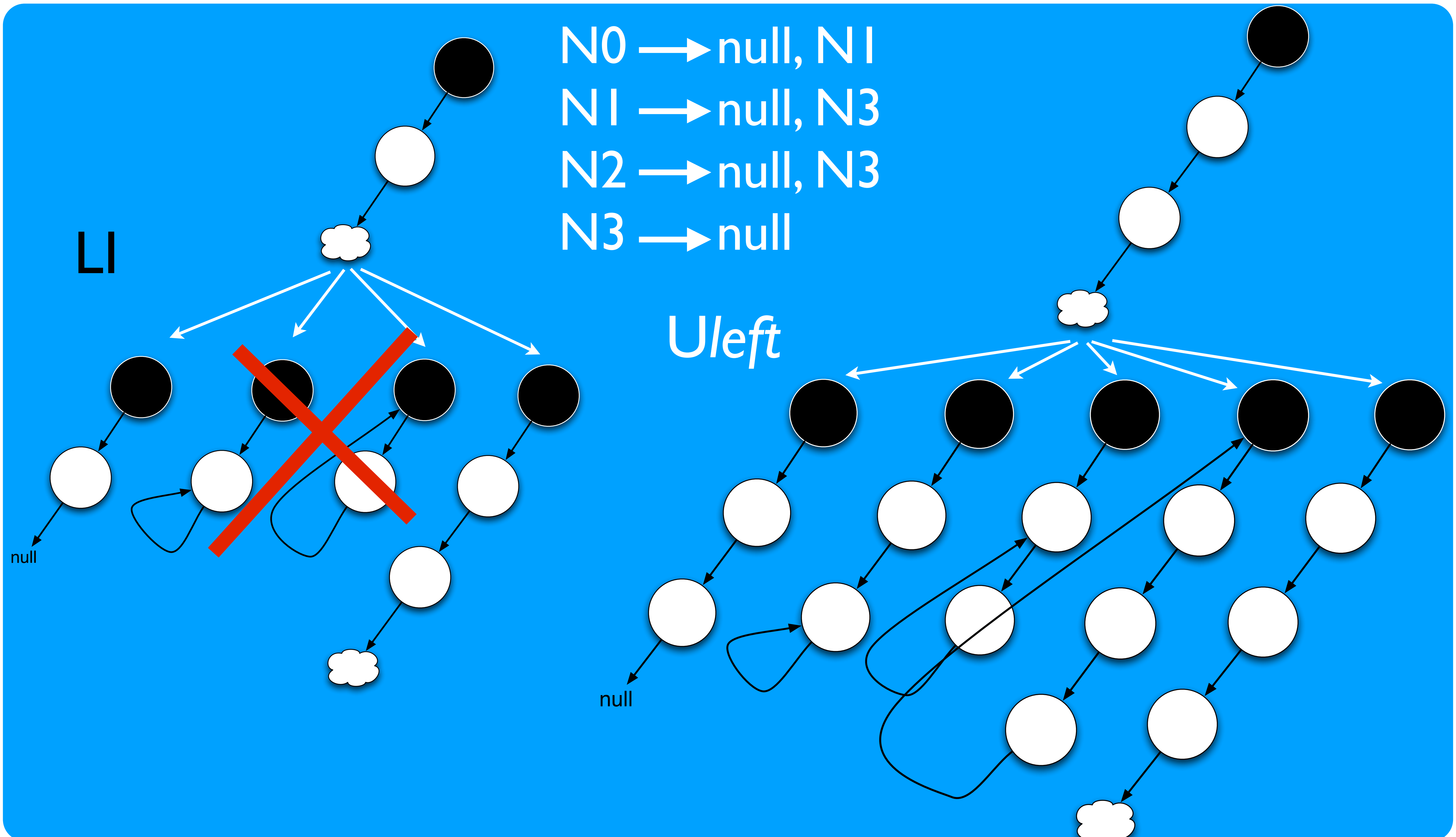
Tight Field Bounds in Symbolic Execution

Intuition



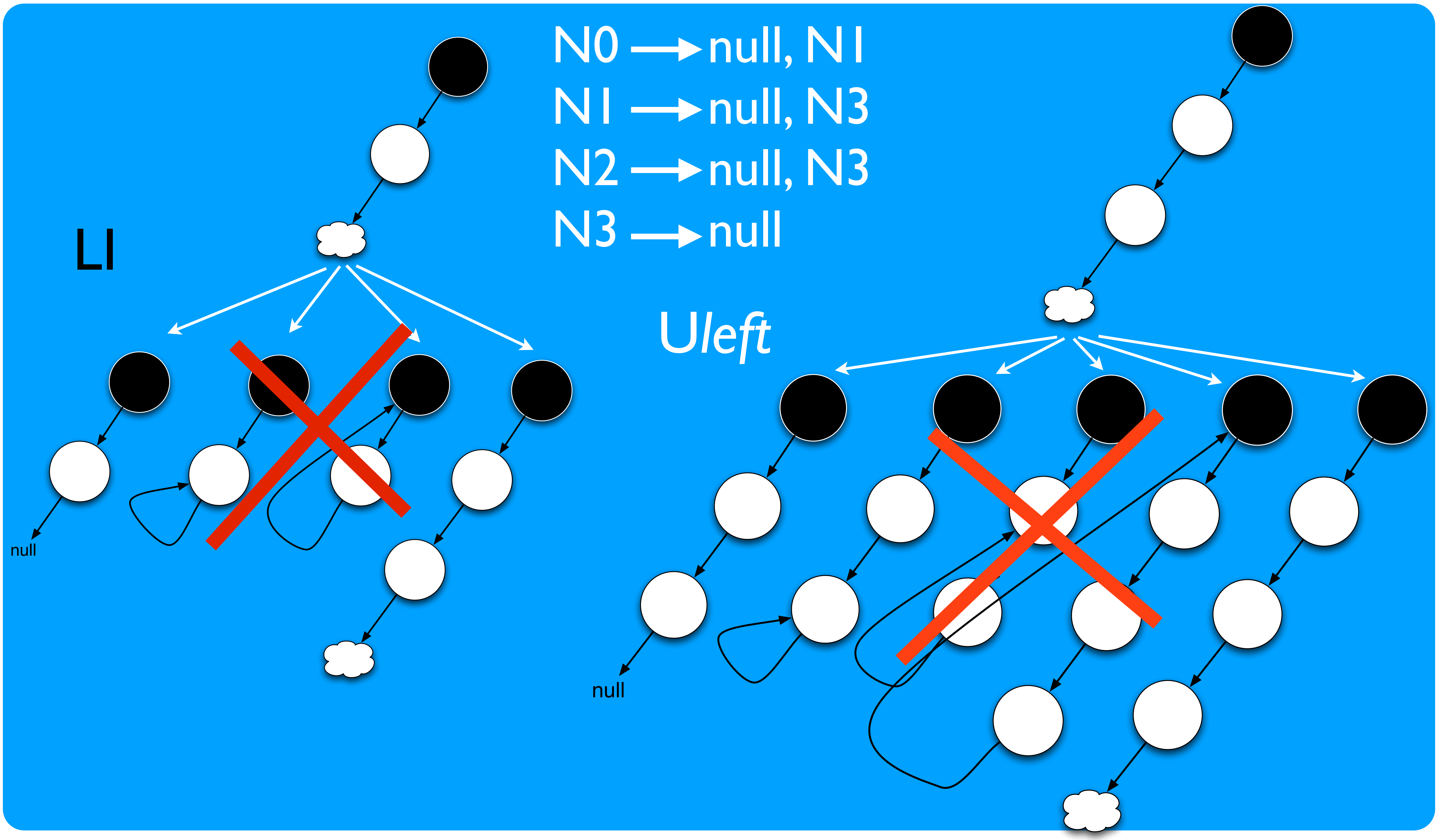
Tight Field Bounds in Symbolic Execution

Intuition



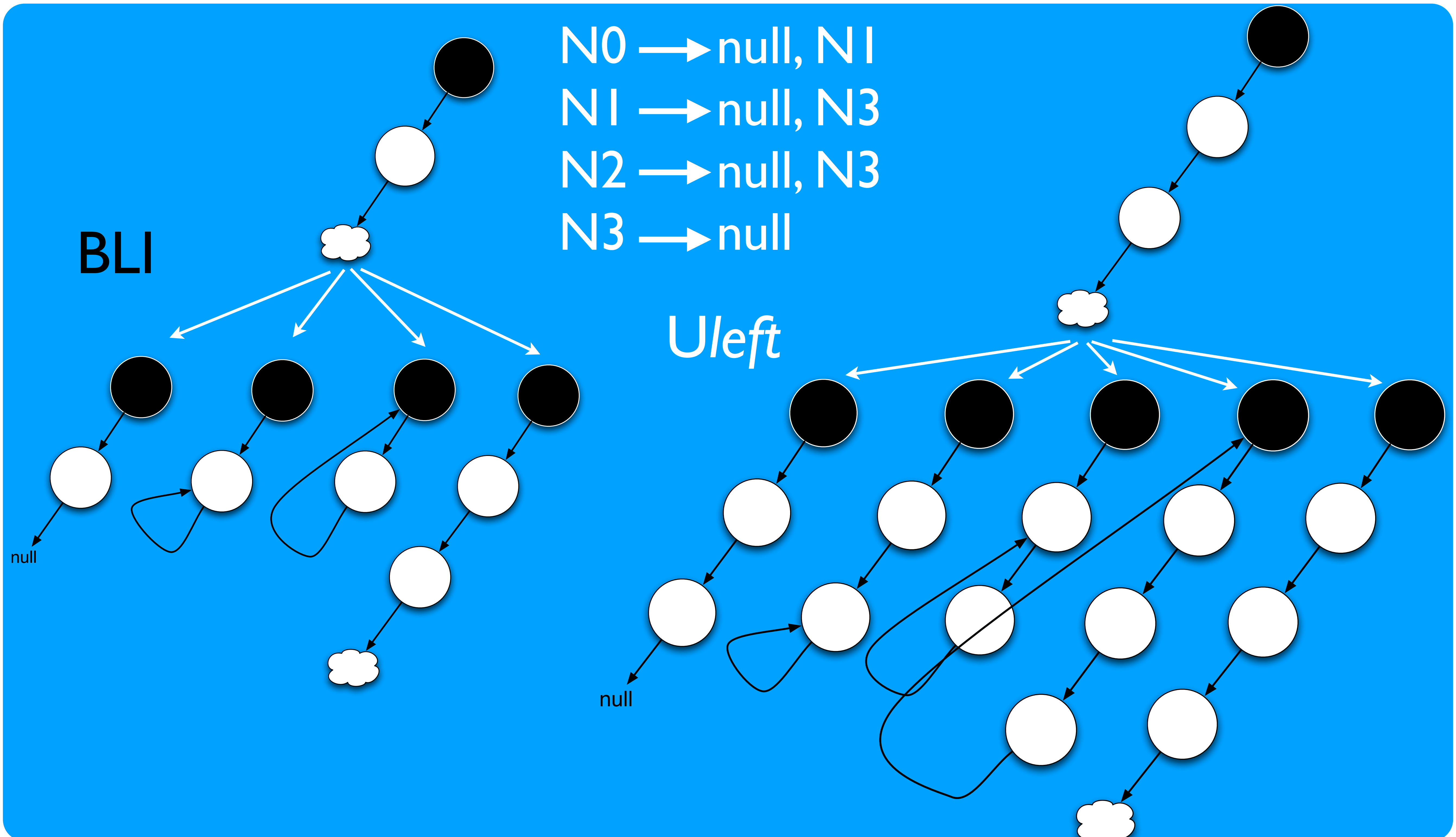
Tight Field Bounds in Symbolic Execution

Intuition



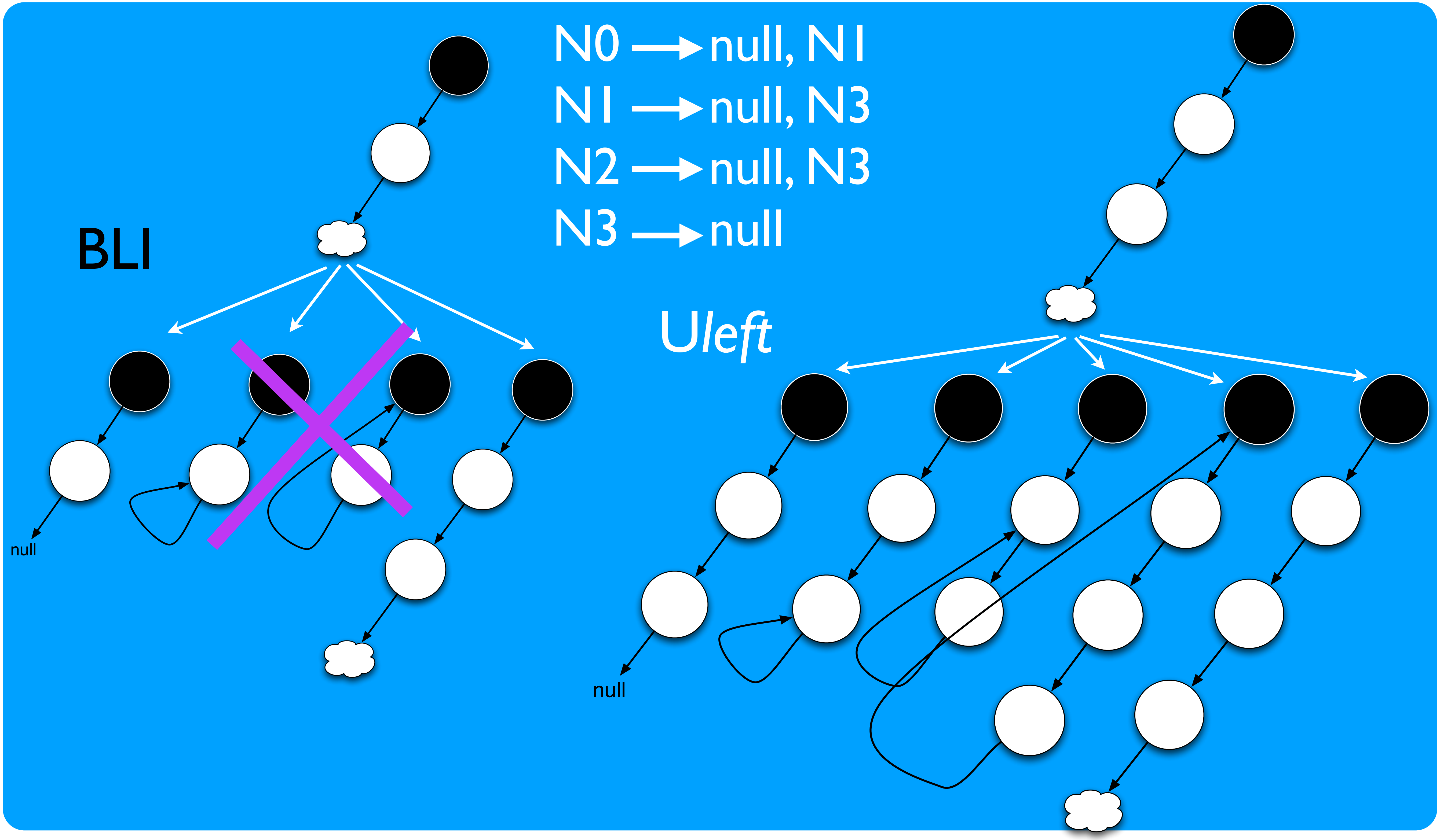
Tight Field Bounds in Symbolic Execution

Intuition



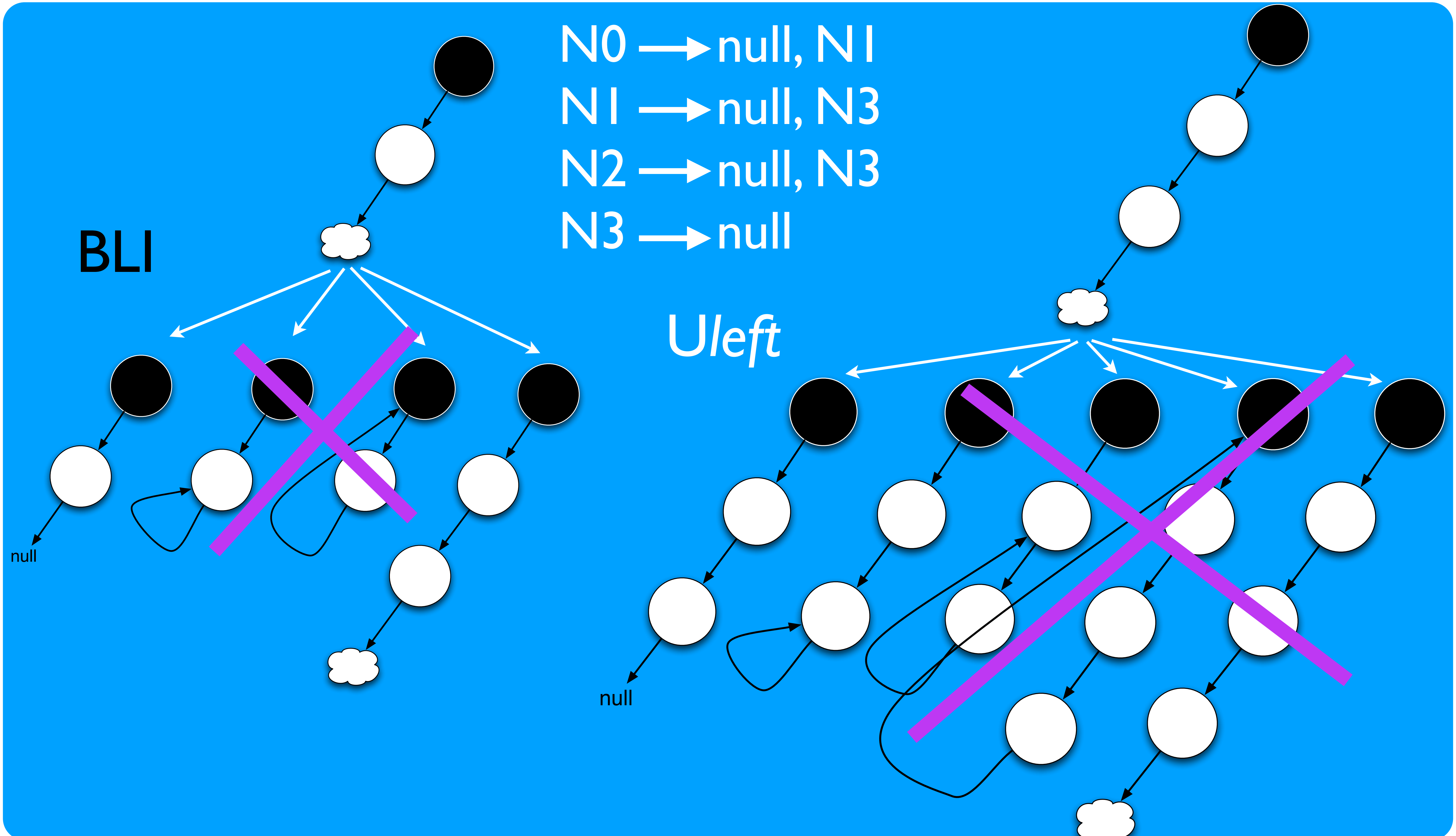
Tight Field Bounds in Symbolic Execution

Intuition



Tight Field Bounds in Symbolic Execution

Intuition



Distributed Analysis by Refinement of Tight Field Bounds

Bounds: A Running Example

`left in N0->N1 + N0->null` `+ N1->N3 + N1->null` `+ N2->N3 + N2->N4 + N2->null` `+ N3->null` `+ N4->null`	`right in N0->N1 + N0->N2 + N0->null` `+ N1->N3 + N1->N4 + N1->null` `+ N2->N3 + N2->N4 + N2->null` `+ N3->null` `+ N4->null`

Fields *left* and *right* from Red-Black tree, up to 5 nodes.

The Distribution Idea

`left in N0->N1 + N0->null` `  + N1->N3 + N1->null` `  + N2->N3 + N2->N4 + N2->null` `  + N3->null` `  + N4->null`	`right in N0->N1 + N0->N2 + N0->null` `  + N1->N3 + N1->N4 + N1->null` `  + N2->N3 + N2->N4 + N2->null` `  + N3->null` `  + N4->null`

- Fields are functions. Therefore, there is still a degree of nondeterminism that can be eliminated from the bounds.

How to split problems

```
left in N0->N1 + N0->null
    + N1->N3 + N1->null
    + N2->N3 + N2->N4 + N2->null
    + N3->null
    + N4->null
```

N0->N1	N0->null	N0->N1	N0->null
+N1->N3	+N1->N3	+N1->null	+N1->null
+N2->N3	+N2->N3	+N2->N3	+N2->N3
+N2->N4	+N2->N4	+N2->N4	+N2->N4
+N2->null	+N2->null	+N2->null	+N2->null
+N3->null	+N3->null	+N3->null	+N3->null
+N4->null	+N4->null	+N4->null	+N4->null

Problem Splitting

```
left in N0->N1 + N0->null
   + N1->N3 + N1->null
   + N2->N3 + N2->N4 + N2->null
   + N3->null
   + N4->null
```

```
right in N0->N1 + N0->N2 + N0->null
   + N1->N3 + N1->N4 + N1->null
   + N2->N3 + N2->N4 + N2->null
   + N3->null
   + N4->null
```

Removing all nondeterminism from nodes N0 and N1 yields 36 subproblems.

Problem Explosion

n	#subprob.
2	36
3	720
4	14,400
5	604,800
6	33,868,800

- Red-Black tree 15 nodes, number of problems removing nondeterminism from nodes N0-N(n-1).

Taming problem explosion

- 1. Do not generate those subproblems that violate the symmetry breaking axioms.
- 2. Mine aliasing information, and avoid subproblems that contain unwanted aliasing.
- 3. Check first that the class invariant is satisfied.

Taming problem explosion

TreeSet	NO	36	720	14,400	604,800	33,868,800
	OP1	9	56	250	3,028	36,163
	OP2	7	19	52	184	694
	OP3	7	19	34	100	172
AVLTree	NO	36	720	8,640	259,200	14,515,200
	OP1	9	56	105	716	11,670
	OP2	7	19	34	94	334
	OP3	7	19	28	70	142
BinHeap	NO	72	648	10,368	259,200	6,480,000
	OP1	26	108	600	4,204	17,030
	OP2	13	36	111	370	1,034
	OP3	7	11	13	15	17

15 nodes. n = 2,...,6

Experimental Results for MUCHO-TACO: Distributed Bug Finding

		s10	s12	s15	s17	s20
TSet	find	01:39	06:17	93:17	260:56	TO
		00:58	00:40	04:31	16:48	516:28
		1.7x	9.4x	20.6x	15.5x	≫1.1x
	insert	TO	TO	TO	TO	TO
		08:37	56:41	241:52	TO	TO
		>69.6x	⋙10.6x	⋙≫2.5x		
	remove	196:58	TO	TO	TO	TO
		13:03	92:29	TO		
		15.1x	≫6.5x			
BH	min	00:14	00:17	01:31	02:51	07:26
		00:14	00:36	01:13	01:27	03:15
		1.0x	0.4x	1.2x	2.0x	2.3x
	decrKey	30:26	TO	TO	TO	TO
		04:35	13:48	20:51	81:05	236:42
		6.6x	>43.5x	⋙28.8x	⋙≫7.4x	⋙⋙≫2.5x
	insert	37:30	218:13	TO	TO	TO
		08:13	19:02	21:05	114:36	325:28
		4.6x	11.4x	>28.4x	⋙5.2x	⋙≫1.8x
	extrMin	36:52	TO	43:33	176:47	TO
		07:01	26:19	01:46	04:05	01:50
		5.2x	>22.8x	24.6x	43.3x	⋙1058x

Distributed Bug-Finding

Reference

Rosner N., Galeotti J.P., Bermudez S., Marucci Blas G., Perez de Rosso S., Pizzagalli L., Zemin L., Frias M.F., *Parallel bounded analysis in code with rich invariants by refinement of field bounds*, ISSTA 2013: 23-33.

Experimental results for Symbolic Execution

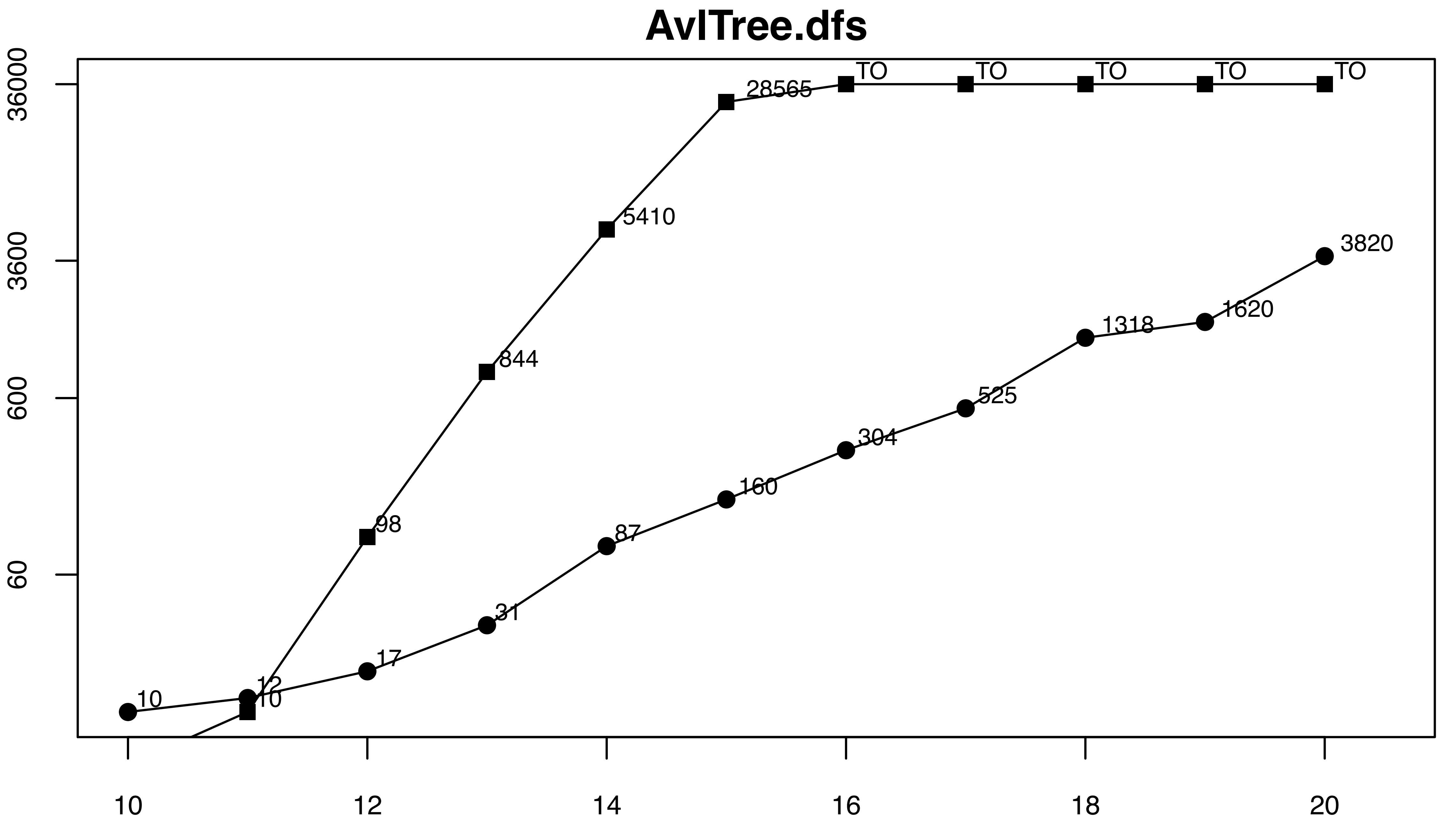


Fig. 14. Analysis time (seconds, logscale) for increasing scope, method AvlTree.dfs().

Experimental results for Symbolic Execution

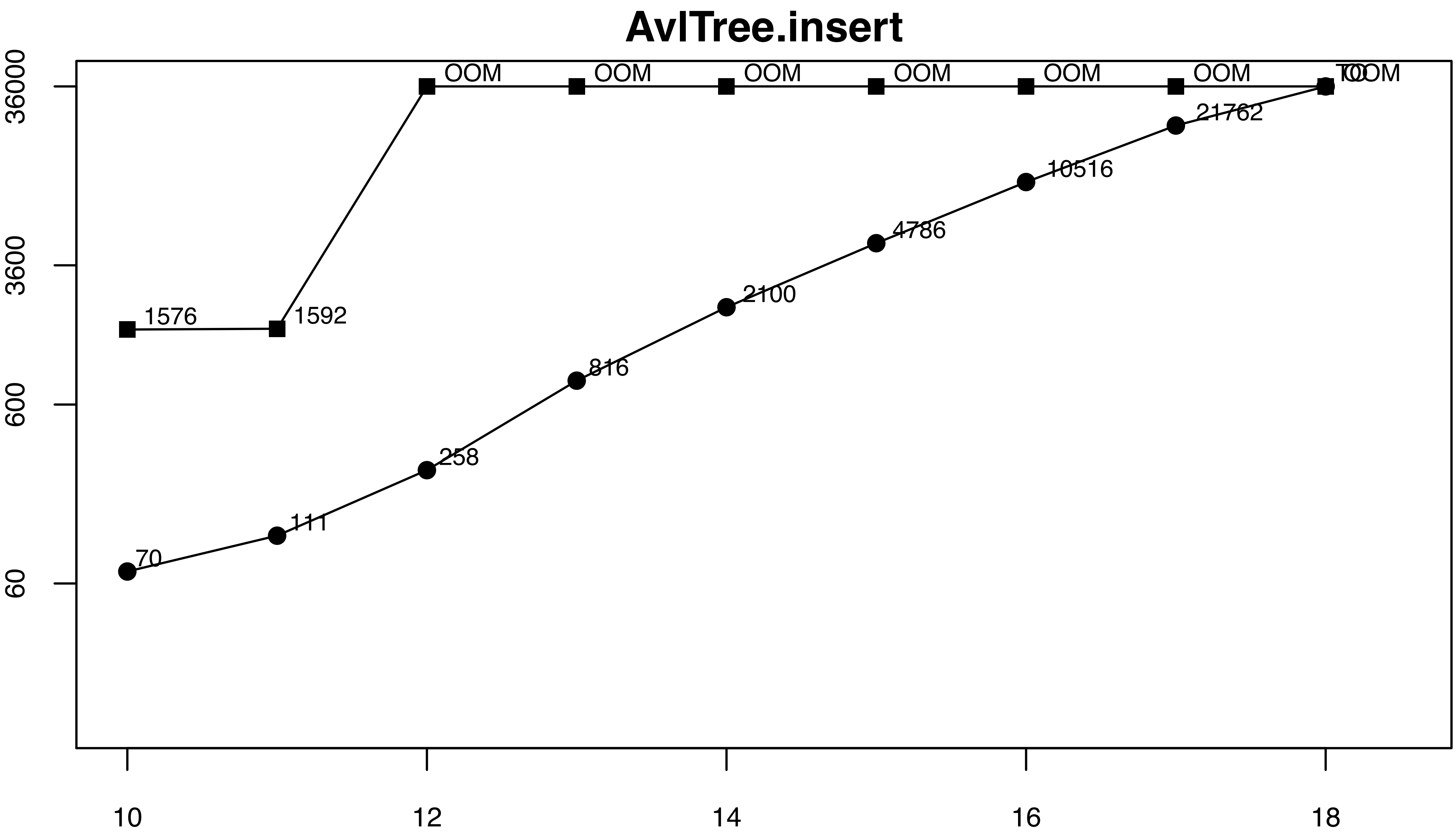


Fig. 15. Analysis time (seconds, logscale) for increasing scope, method AvlTree.insert().

Experimental results for Symbolic Execution

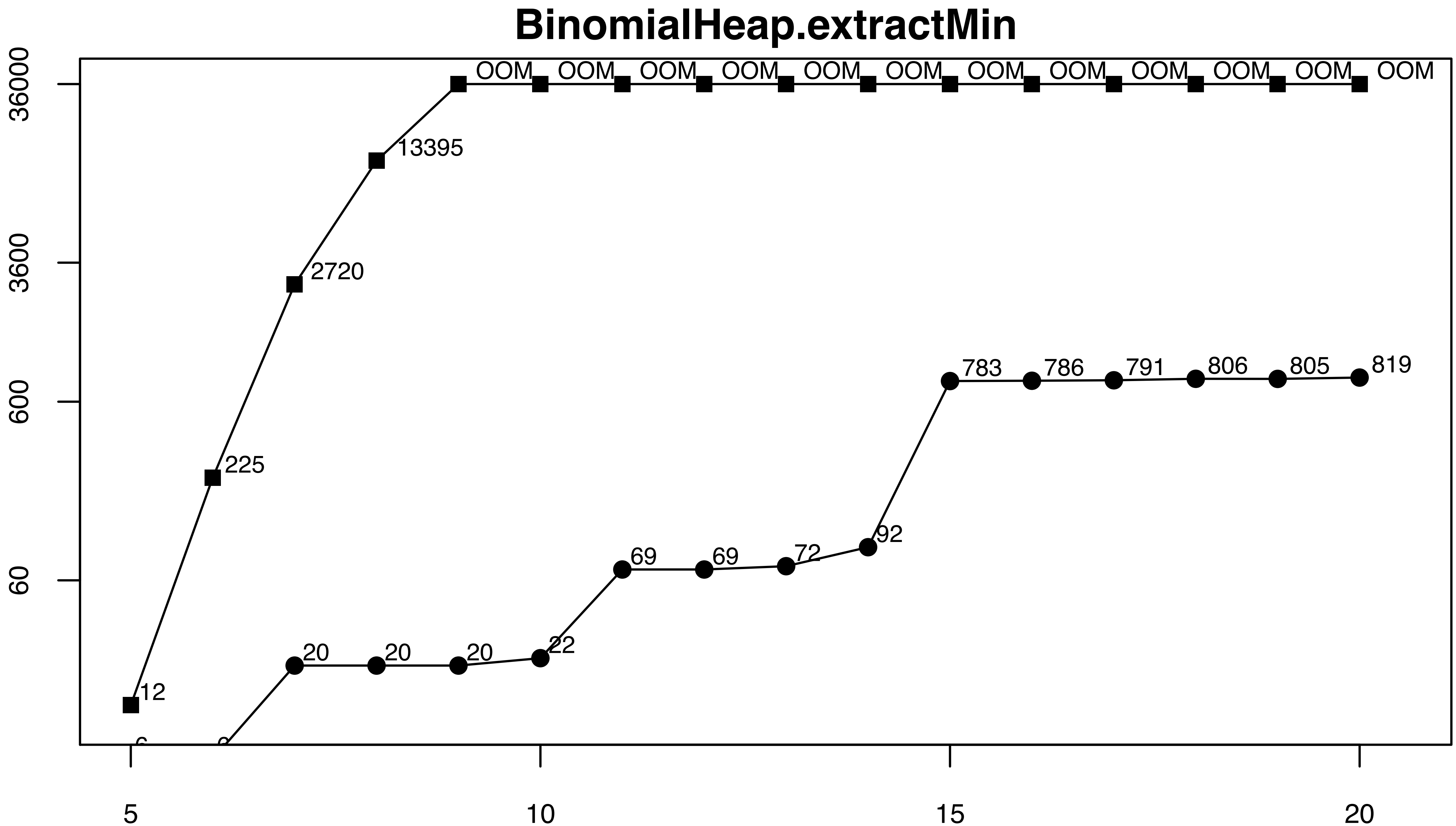
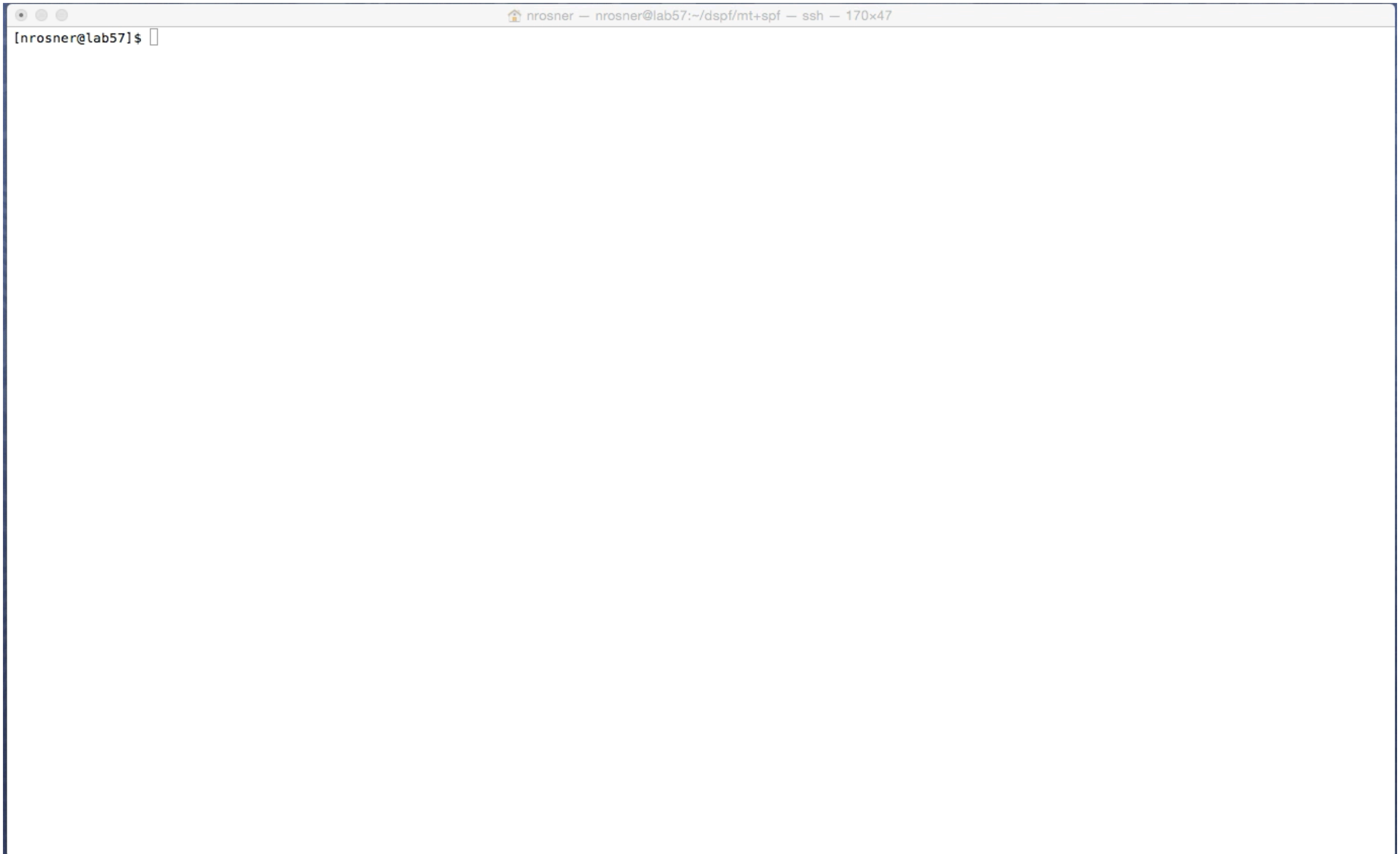


Fig. 27. Analysis time (seconds, logscale) for increasing scope, method BinomialHeap.extractMin().

Distributed Symbolic Execution

Demo



Distributed Symbolic Execution

Demo

```
nrosner — nrosner@lab57:~/dspf/mt+spf — ssh — 170×47
[nrosner@lab57]$ 
```

Conclusions

- We showed that the same partition technique can be used in two seemingly unrelated program analyses.
- In both cases, the results were very positive.
- If you are a user of a program analysis technique, and the technique may profit from the use of tight field bounds, then a distributed version of the analysis can be obtained almost for free by reducing nondeterminism from tight field bounds.

Thanks!