BLISS: Bounded Lazy Initialization with SAT-Support

Marcelo Frias
Buenos Aires Institute of Technology

Joint work with Nicolás Rosner, Nazareno Aguirre, Jaco Geldenhuys and Willem Visser

Funded by NPRP grant NPRP-4-1109-1-174 from the Qatar National Research Fund

IFIP WG 2.2, Munich, September 2014

Contents

- Symbolic Execution
- Lazy Initialization
- TACO Bounds
- (Refined) Bounded Lazy Initialization
- BLISS
- BLISS-DB

Goal

BLISS is a technique that improves the way in which SPF analyzes code over heap-allocated data structures.

checker that uses symbolic states.

Symbolic Execution

```
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)

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

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

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

(i=i0, j=j0, k=k0, output=k0,

(i0>j0 || i0>k0) && (i0<=j0 || k0<=j0))

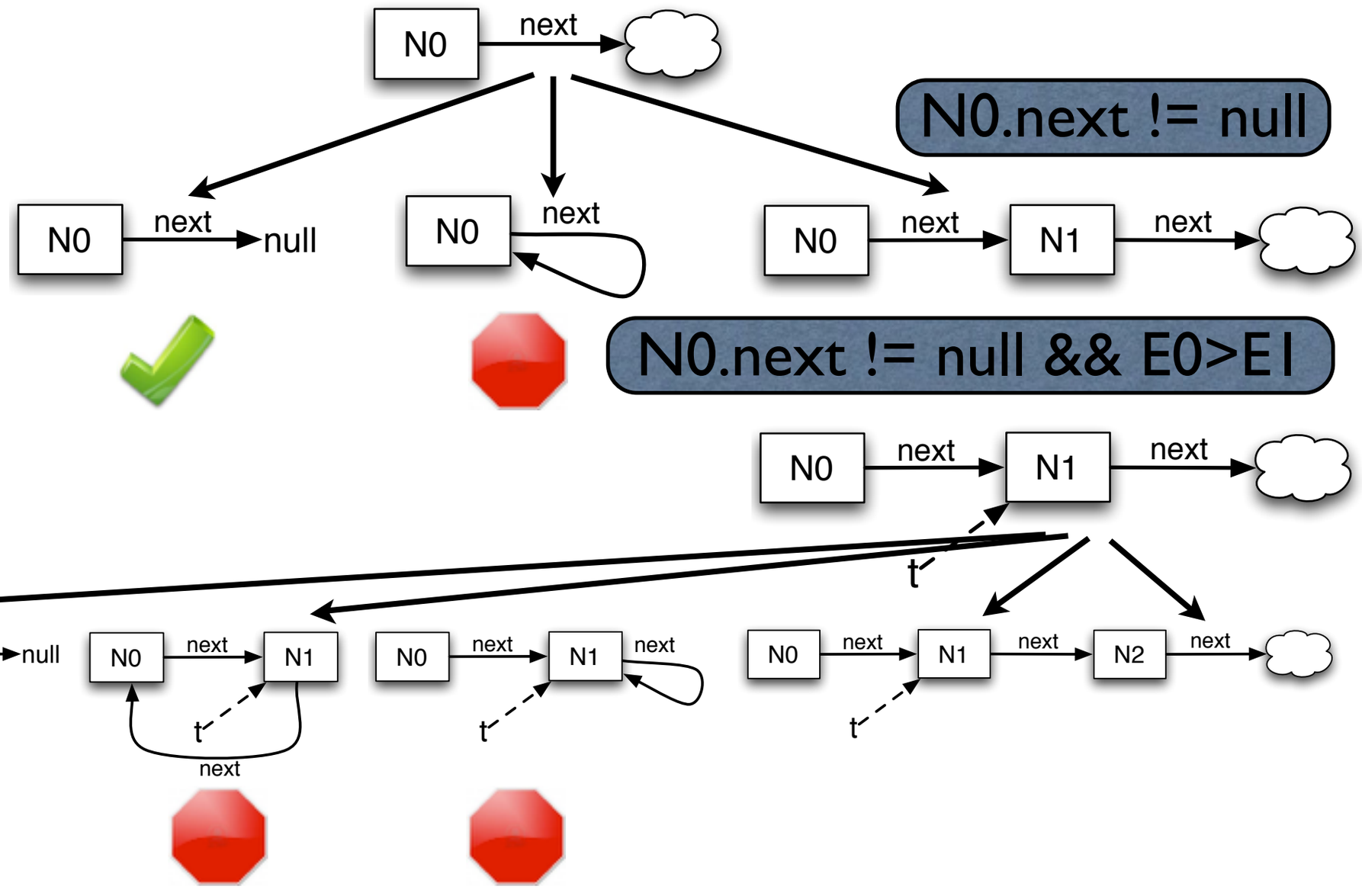
(i0>j0 || i0>k0) && (i0<=j0 || k0<=j0))

Symbolic Execution and Dynamically Allocated Structures

- Khurshid, Pasareanu and Visser proposed Lazy Initialization [TACAS03].
- An object attribute is initialized just when its value is accessed. Up to that moment, the attribute value is kept symbolic.

Lazy Initialization (LI)

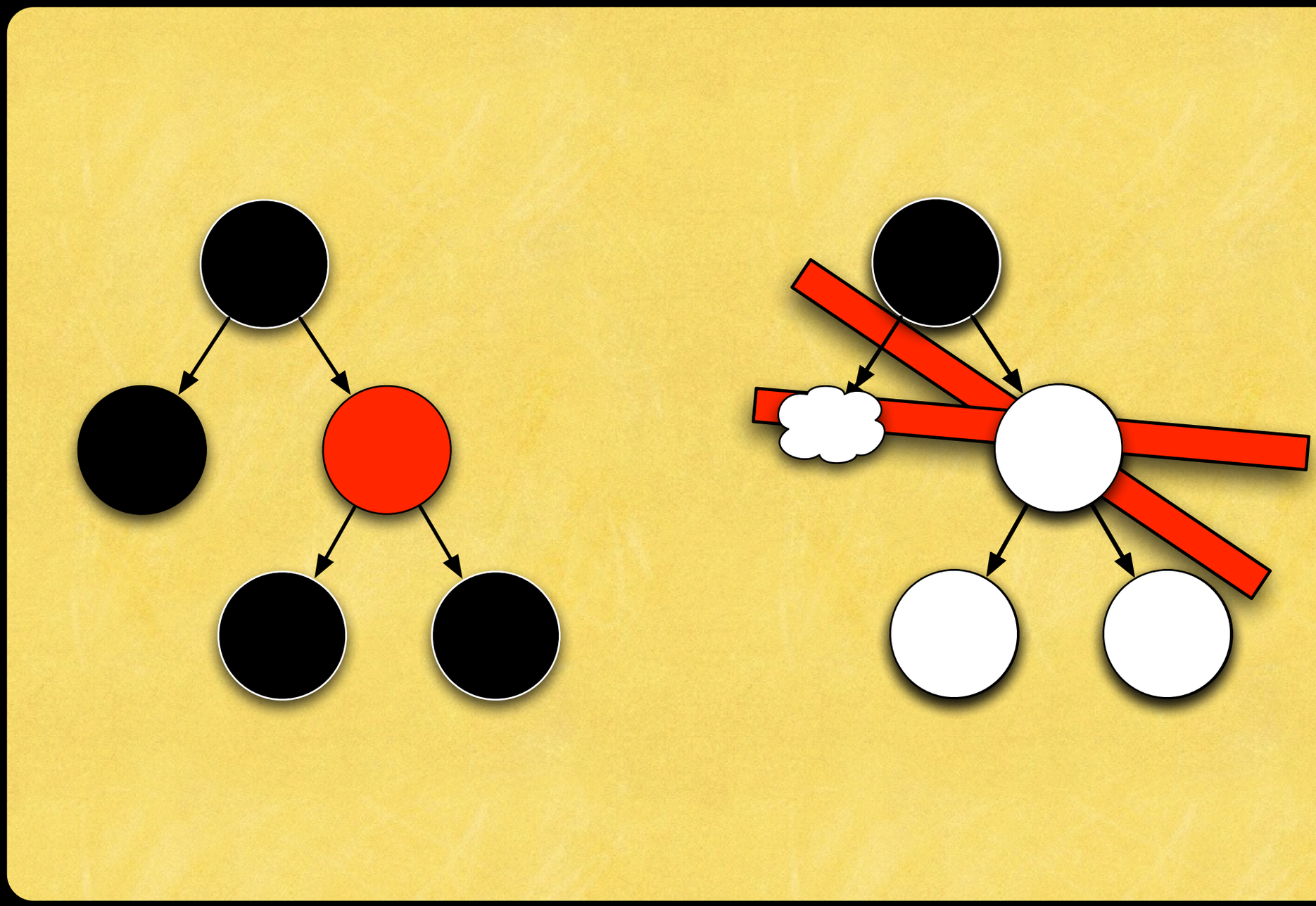
```
class Node {
   int elem;
   Node next;

   \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

- Does not generate isomorphic heaps.
- Constraints must be adapted so that they can be evaluated on partially symbolic structures.

Consider a Red-Black Tree with up to 5 nodes



Bounded Lazy Initialization (BLI)

- TACO: Translation of Annotated COde (ISSTA2010, TSE2013)
- Use TACO bounds to reduce the number of options whenever a symbolic node is concretized.

TACO bounds

- Given a java field *f*, a *TACO bound* is a minimal relation U*f* such that in each valid structure, *f* is contained in U*f*.
- Automatically computed from a class invariant.
- Reusable across methods in a class, and across tools.

TACO bounds (RBTree, 4 nodes)

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

U*left*

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

U*right*

Reduces the options from 40 (4x5 + 4x5) to just 15.

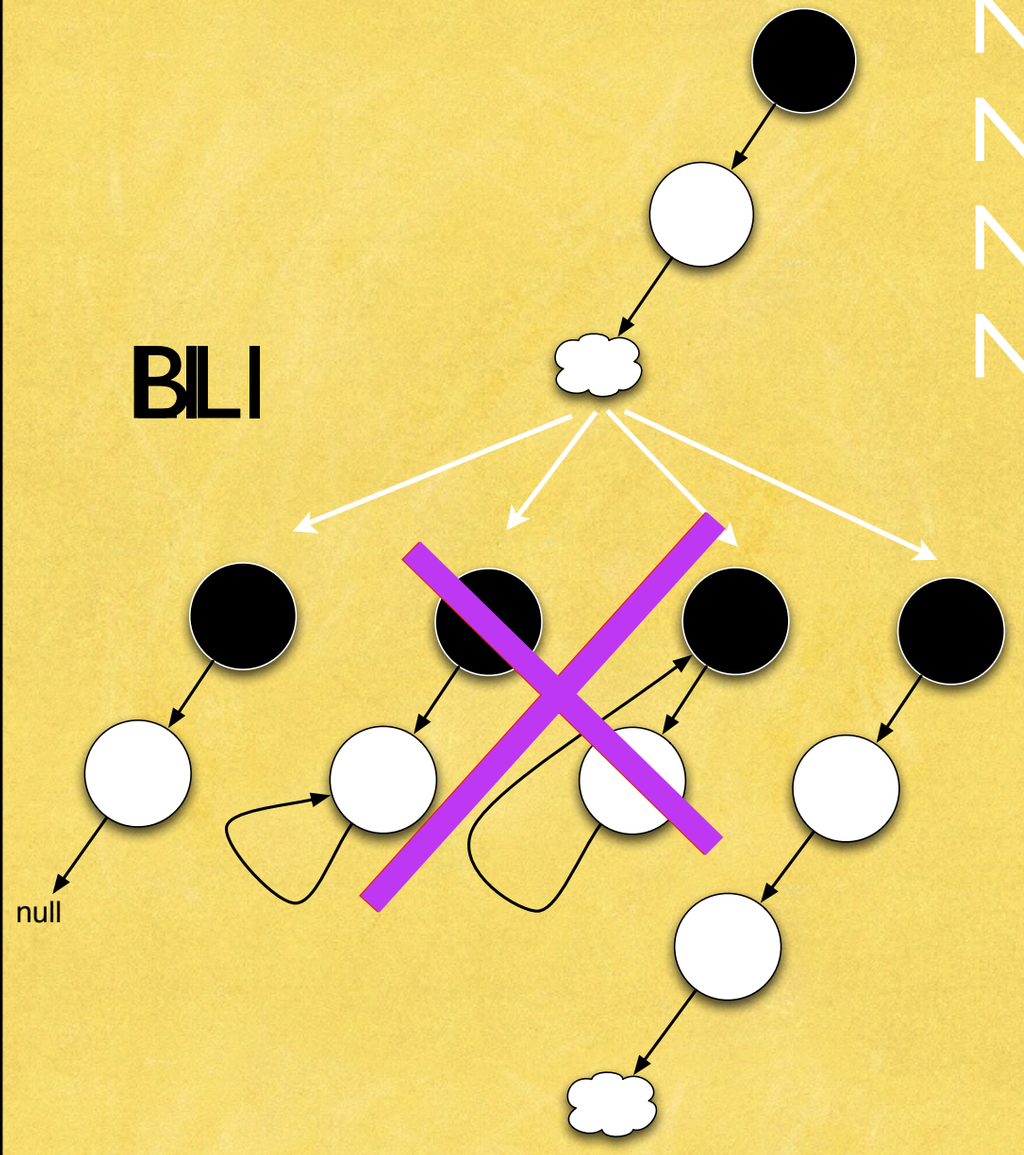
Idea

- When a reference is concretized, consider only the target objects feasible as per the bounds.

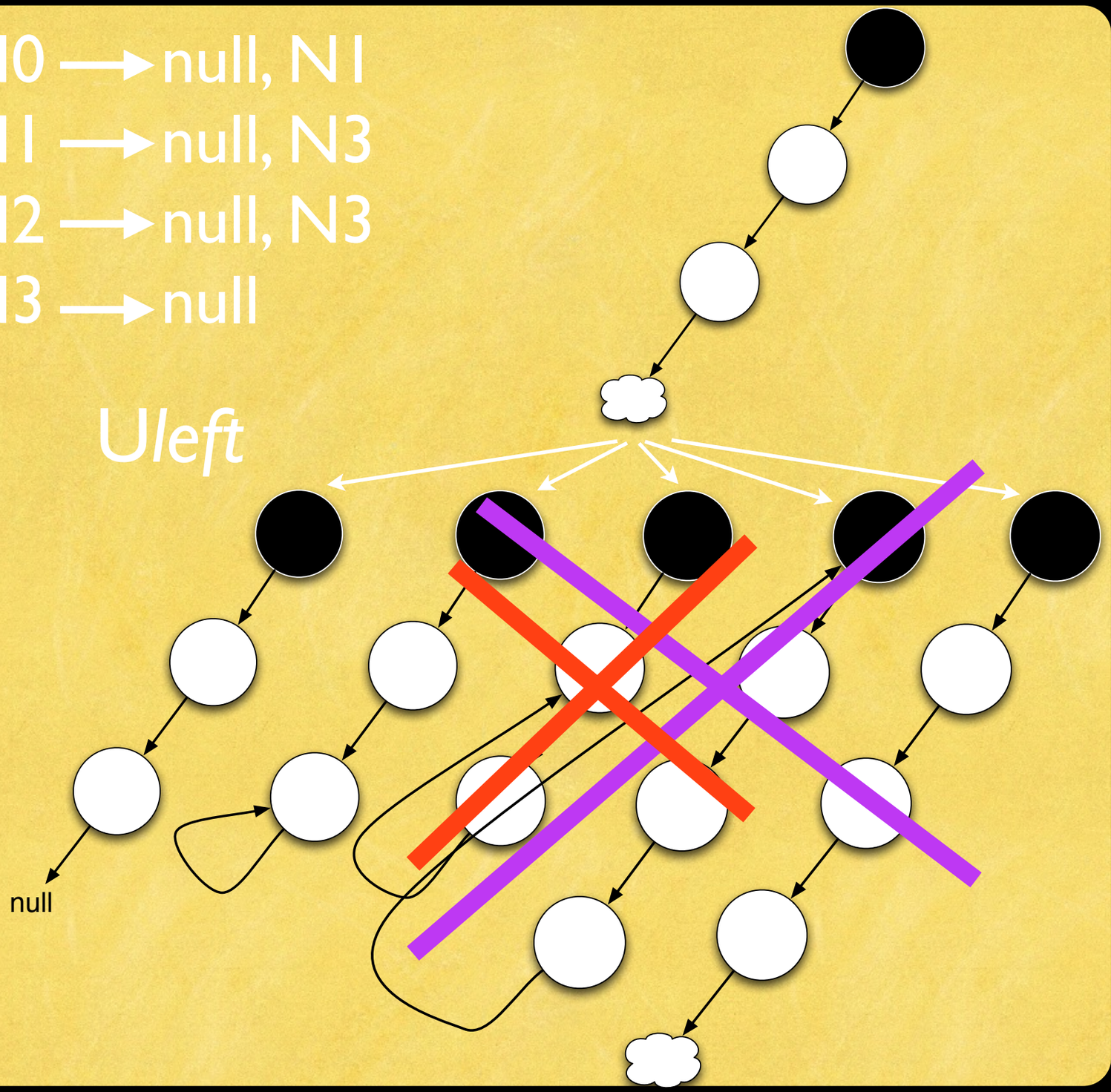
Improvement

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

BILI



Uleft



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

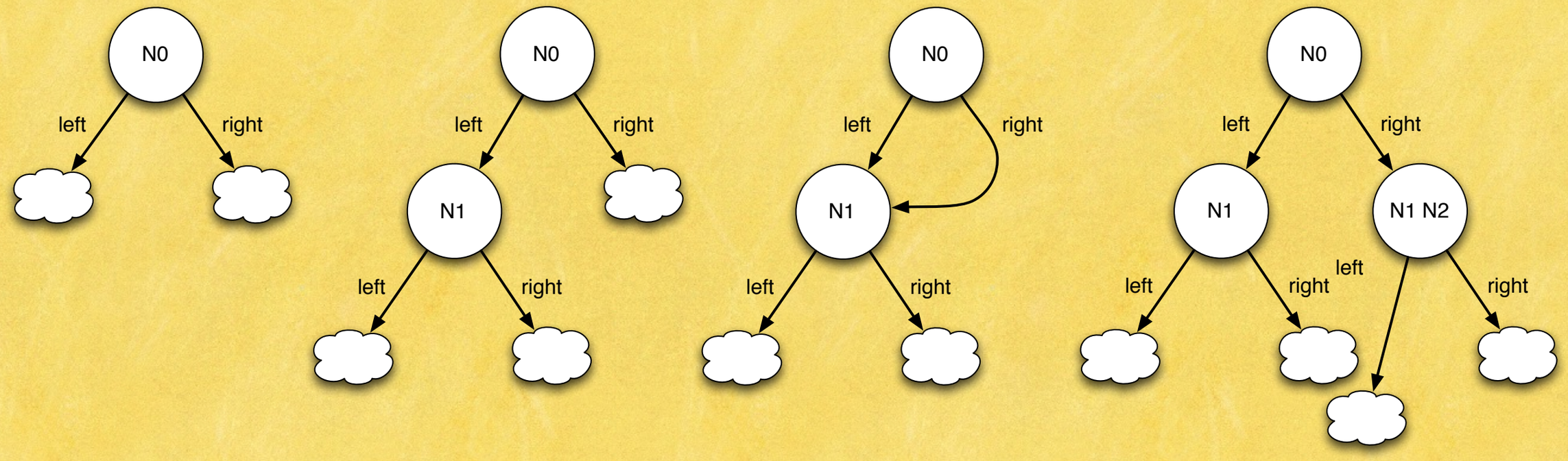
Uleft

BLI: Illustration

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

Uright

- Root node labeled N0
- Ni.f labeled according to the bound U*f*.



BLI Numbers
Class TreeSet, Method BFS_Traverse

Nodes	10	11	12	13	14	15	16	17	18
Time	00:57 00:36	03:46 01:53	14:22 08:08	61:04 37:15	OofM OofM				
Number of generated structures	23713 16353	82499 64835	290511 248783	1033411 936131					

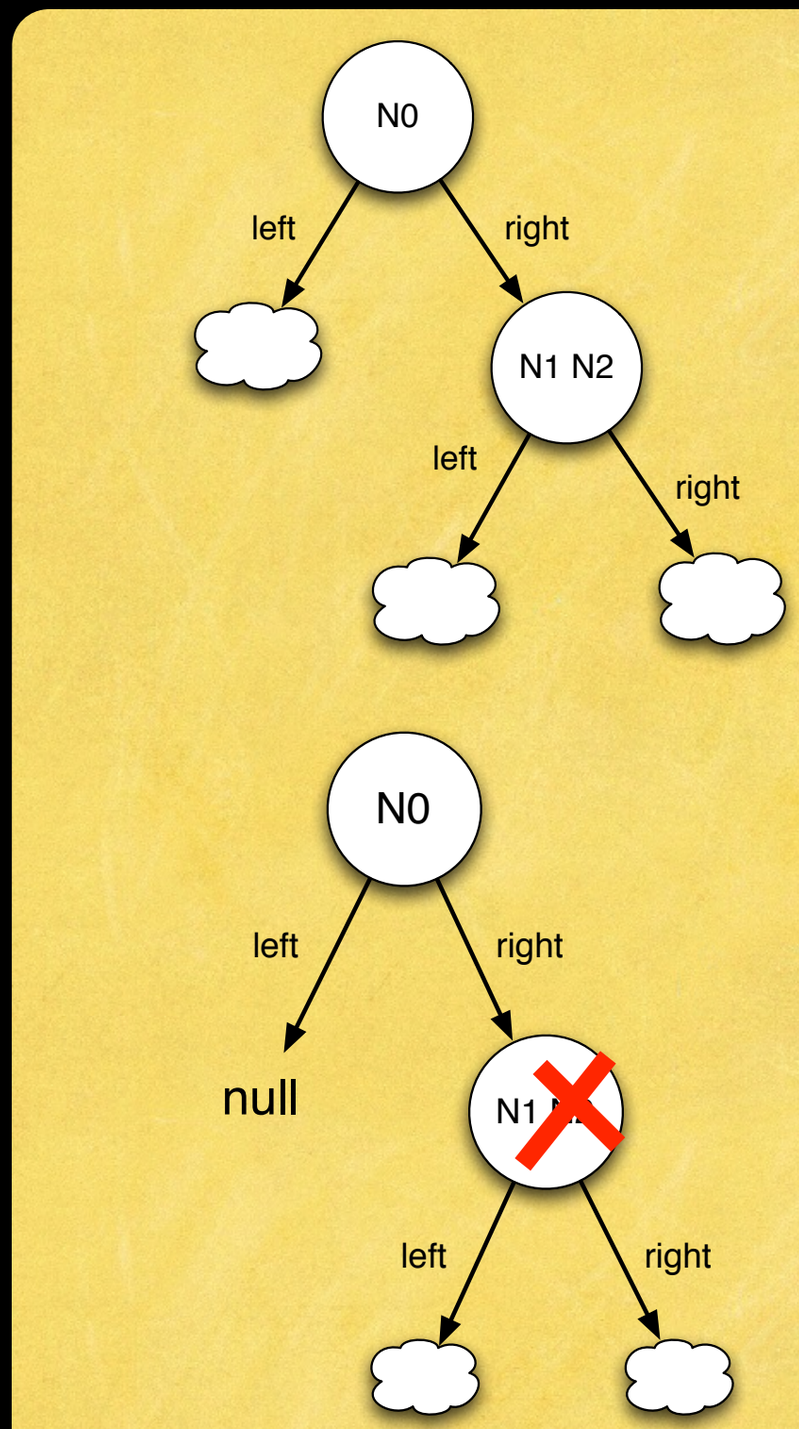
LI BLI

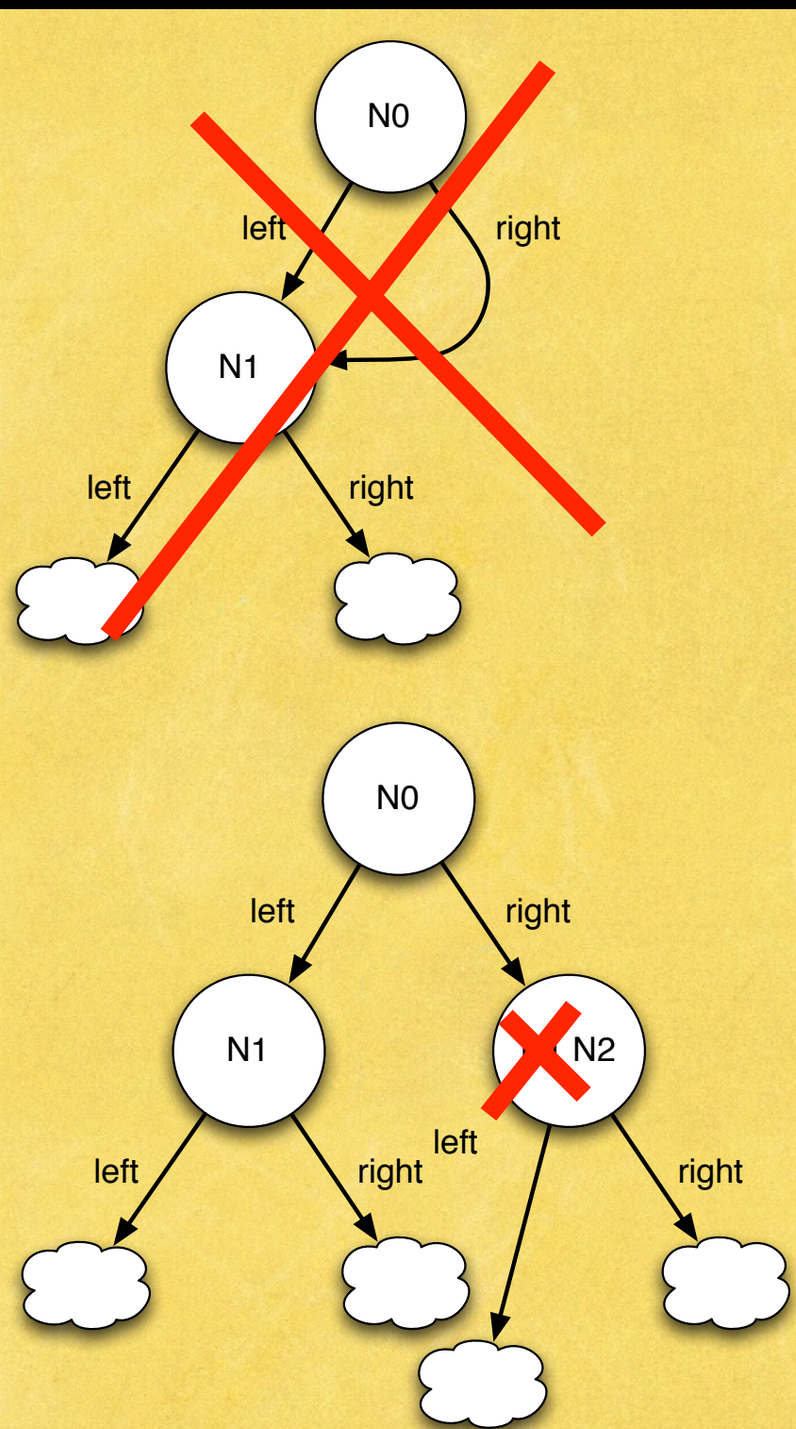
Speed up: about 2X

TACO Symmetry Breaking and Refined BLI

- TACO forces node identifiers to be selected in breadth-first search order.
- Therefore, the labeling of some nodes may be refined.
- This refinement prevents the generation of many structures.

Refined BLI Illustrated





RBLI Numbers
Class TreeSet, Method BFS_Traverse

Nodes	10	11	12	13	14	15	16	17	18
Time	00:57 00:36 00:06	03:46 01:53 00:20	14:22 08:08 01:13	61:04 37:15 06:18	OofM OofM 19:24	 OofM			
Number of generated structures	23713 16353 4557	82499 64835 18375	290511 248783 72399	1033411 936131 278763	1508943				

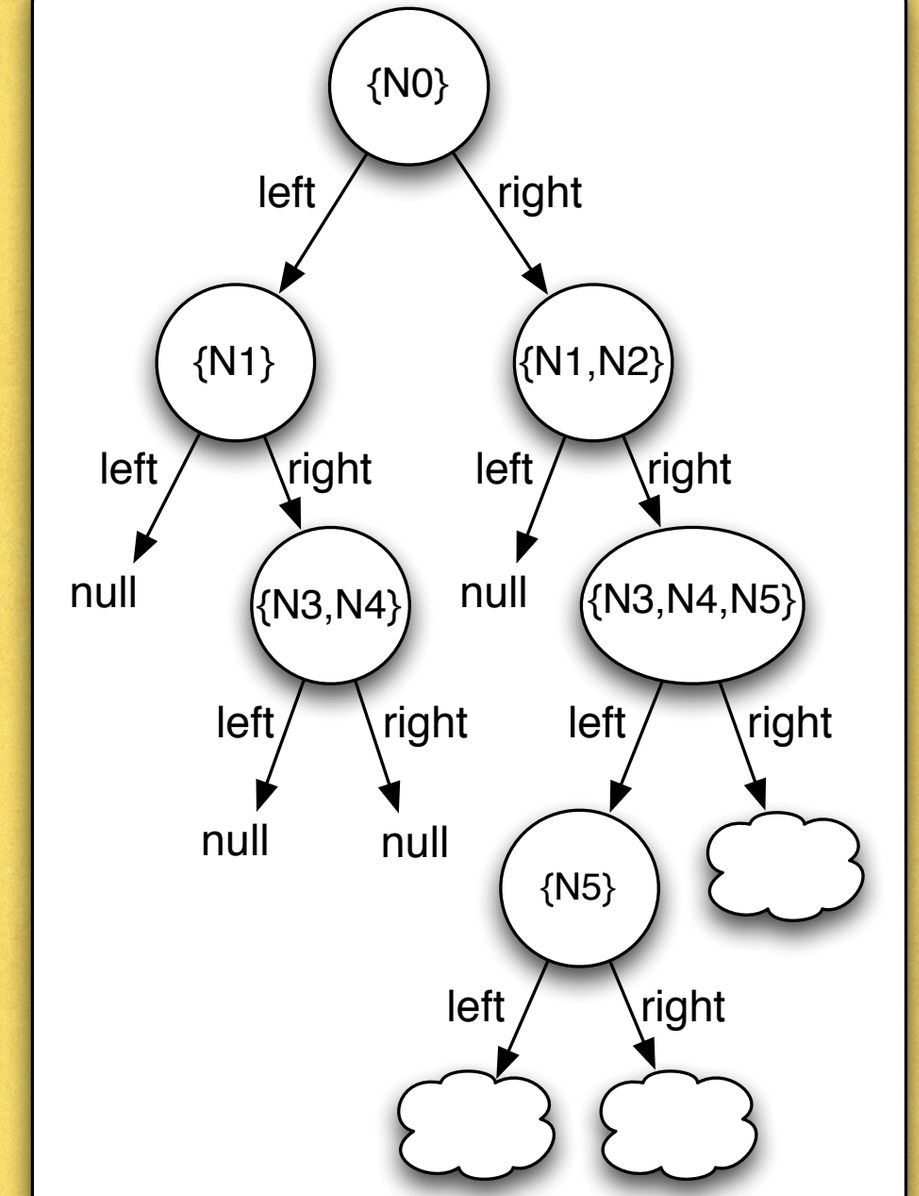
LI BLI RBLI

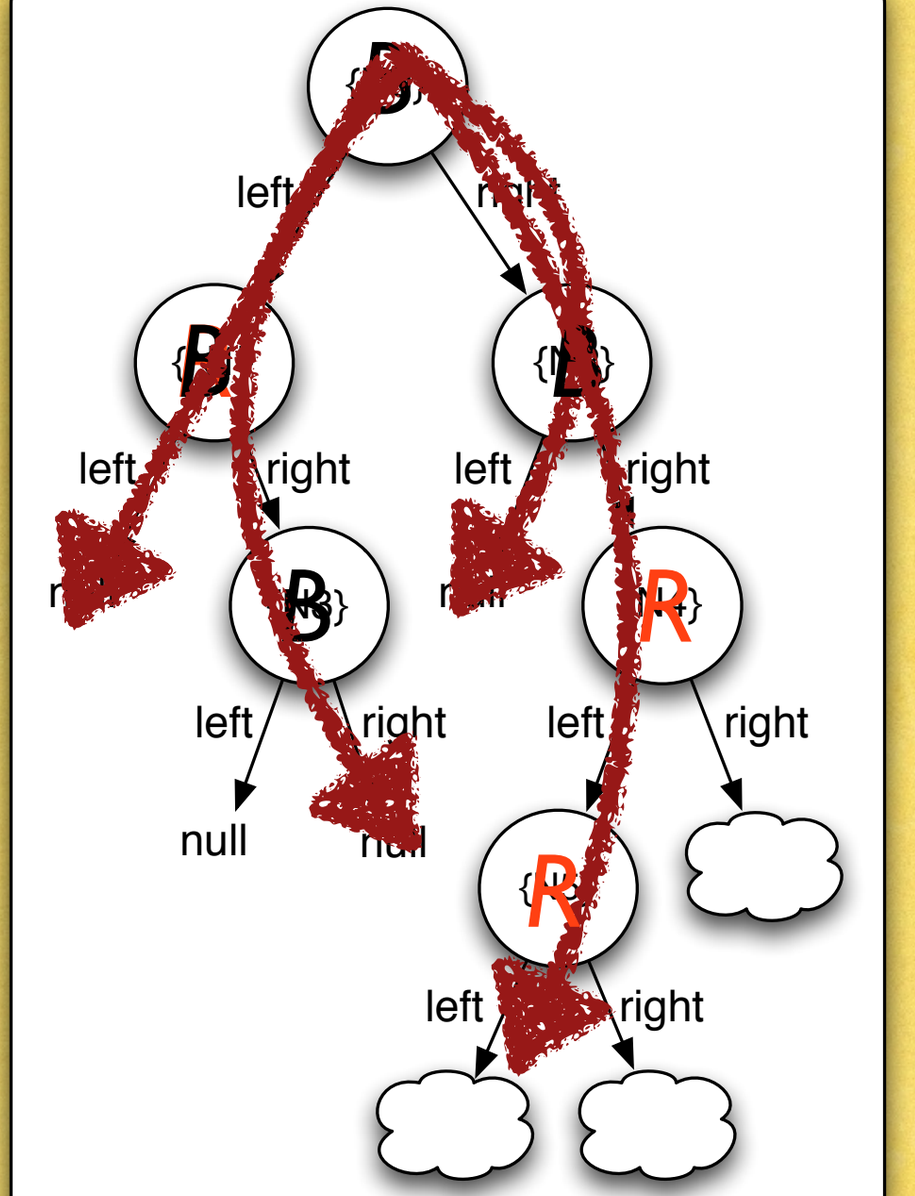
Speed up: about 10X (S13)
Infinite for S14

BLISS (Bounded Lazy Initialization with Sat Support)

- Our goal was to promote pruning induced by invariants as early as possible.
- Before extending a partially symbolic structure we check the invariant in order to determine if it can ever be made into a concrete valid structure.
- SAT-checks are expensive, so only useful if many structures are pruned.

An Example


{N0}
left
right
{N1}
{N1,N2}
left
right
left
right
null
{N3,N4}
null
{N3,N4,N5}
left
right
left
right
left
right
null
null
{N5}
left
right
left
right


left
right
left
right
left
right
left
right
null
left
right
left
right
left
right

From Partially Symbolic Structures to a SAT Problem

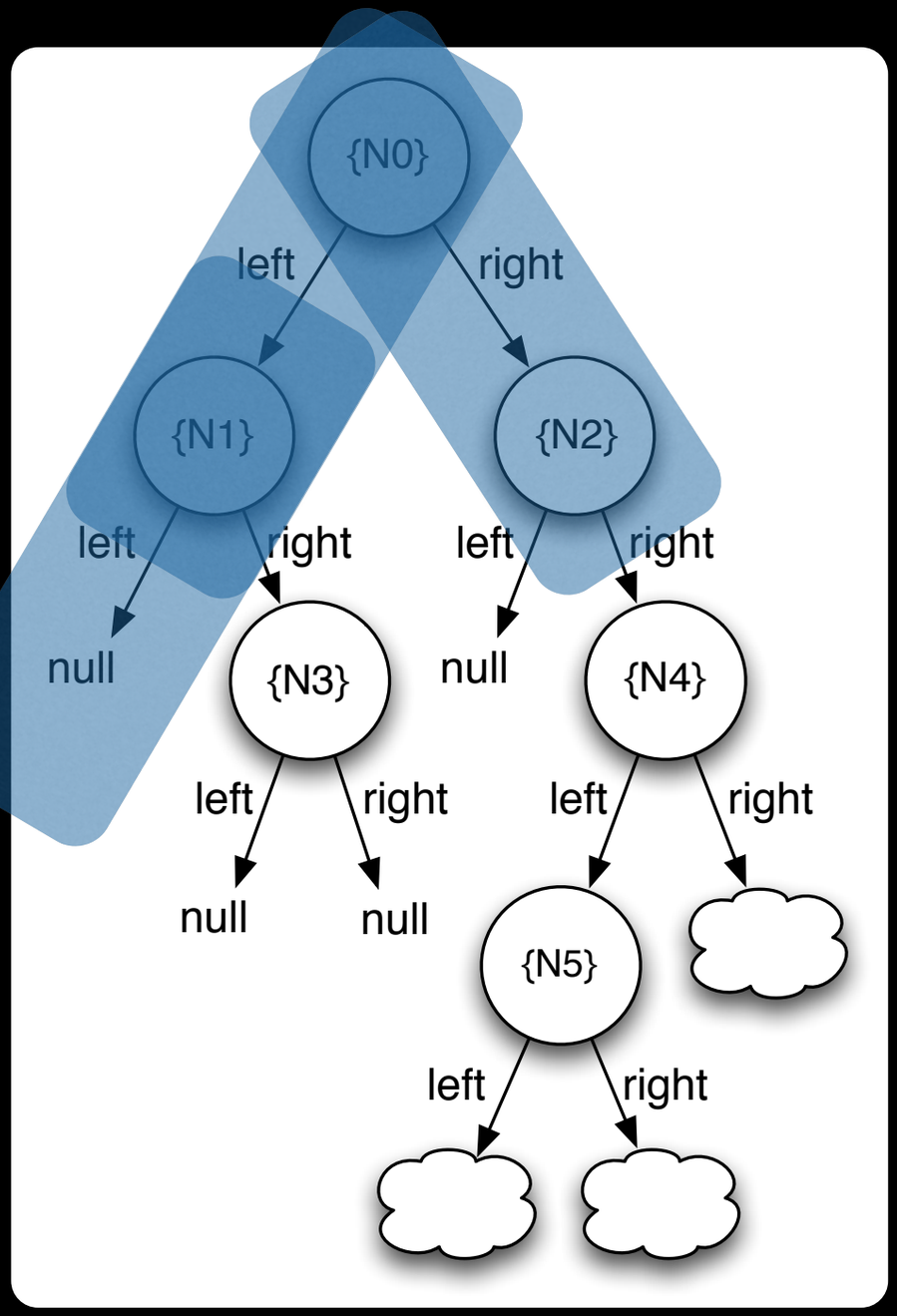
We store a map M such that

M(sourceNode,field,targetNode) = var

iff var captures the fact

sourceNode.field = targetNode

From Partially Symbolic Structures to a SAT Problem



SAT(M(N0,left,N1), M(N0,right,N2), M(N1,left,null),… TreeSet Invariant)

BLISS Numbers
Class TreeSet, Method BFS_Traverse

Nodes	10	11	12	13	14	15	16	17	18
Time	00:57 00:36 00:06 00:10	03:46 01:53 00:20 00:23	14:22 08:08 01:13 01:10	61:04 37:15 06:18 03:10	OofM OofM 19:24 08:10	 OofM 20:36	 48:53	 136:12	 331:12
Number of generated structures	23713 16353 4557 217	82499 64835 18375 407	290511 248783 72399 863	1033411 936131 278763 1767	 1508943 3463	 6804	 13572	 27400	 56080

LI BLI RBLI BLISS

Speed up: about 20X (S13)
Infinite for S14-S18

An Optimization

- The kind of analyses provided by SPF are executed starting with some limit *k* on the number of nodes, and this bound is increased until:
  - we are happy with the reached value,
  - the analysis takes too long and it is stopped.

Optimization (Continued)

- If a SAT check in bound *k* yields SAT, it will also be SAT in larger bounds.
- We store partial instances and the analysis outcome in a database, and look up previous verdicts before launching the SAT solver.

BLISS-DB Numbers
Class TreeSet, Method BFS_Traverse

Nodes	10	11	12	13	14	15	16	17	18
Time	00:57 00:36 00:06 00:10 00:07	03:46 01:53 00:20 00:23 00:16	14:22 08:08 01:13 01:10 00:45	61:04 37:15 06:18 03:10 01:59	OofM OofM 19:24 08:10 05:02	OofM 20:36 12:33	48:53 30:12	136:12 89:27	331:12 218:17
Number of generated structures	23713 16353 4557 217	82499 64835 18375 407	290511 248783 72399 863	1033411 936131 278763 1767	1508943 3463	6804	13572	27400	56080

LI BLI RBLI BLISS BLISS-DB

Speed up: about 30X (S13)
Infinite for S14-S18

Discussion

- BLISS offers important speed ups in the analyzed case studies (reaching 20,000X).
- BLISS is sound and complete.
- Two invariants are required:
  - a procedural (hybrid) one that handles partially symbolic structures (required by LI).
  - a declarative one required by BLISS.

Other applications of tight bounds

- TACO (ISSTA 2010, IEEE TSE 2013).
- FAJITA, test generation (ICST 2013)
- MUCHO-TACO (SAT-based distributed analysis of code, ISSTA 2013).
- BLI (Bounded Lazy Initialization, NFM 2013).
- BLISS (Bounded Lazy Initialization with SAT-Support, to appear in IEEE TSE).
- HyTek (Exhaustive input generation from hybrid specs, OOPSLA 2014).

Conclusion

- If you have developed a tool for the analysis of Java-like programs, it may be worth thinking if TACO bounds can improve your analyses.

Thanks!