Algunas Técnicas de Generación de Casos de Test para Estructuras Complejas Alojadas en Memoria Dinámica

Nazareno Aguirre[1], Valeria Bengolea[1], Juan Pablo Galeotti[2] y Marcelo Frias[3]

[1]Universidad Nacional de Río Cuarto

[2]Universidad de Buenos Aires

[3]Instituto Tecnológico Buenos Aires

JCC 2010, Rosario, Octubre de 2010

Generación de Casos de Test

- Es generalmente una actividad costosa en tiempo, realizada manualmente
- Fácil de automatizar para rutinas parametrizadas con tipos de datos básicos (e.g., generación aleatoria)
- Muy difícil para rutinas parametrizadas con tipos de datos estructuralmente complejos (e.g., AVLs, árboles de búsqueda, árboles rojos y negros, grafos, ...)

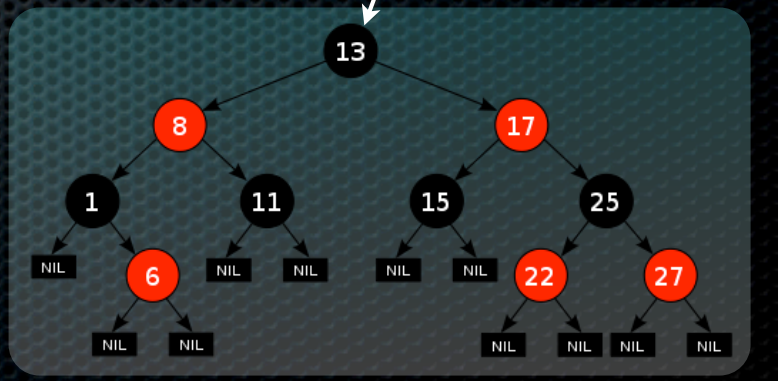
Datos Estructuralmente Complejos en Aplicaciones

- Las estructuras complejas no forman parte solamente de librerías de estructuras de datos. Se encuentran en:
  - Aplicaciones que manipulan archivos XML (o similares), los cuales deben respetar ciertas reglas de buena formación
  - Aplicaciones para el análisis de sitios web, donde los sitios pueden interpretarse como grafos con ciertas características (acíclicos?, fuertemente conexos?)
  - ...

Enumerar estructuras. Cuán difícil resulta?

- Consideremos árboles rojos y negros (n nodos, m claves)
  - Número de árboles binarios: $\frac{(2n)!}{(n+1)! \times n!}$
  - Número de asignaciones de claves a nodos: m^n
  - Número de asignaciones de colores a nodos:2^n

este es uno de los “pocos” que satisface el invariante de RN



10 n, 10 m ->
659758296862017282944639409804963640002358120158936736338165139364428982252962109045307813275645217086840083605484761094973030400000000000000000000000000000

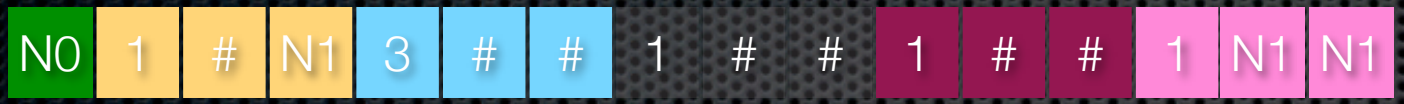
Problemas en la Explosión de Estructuras Posibles

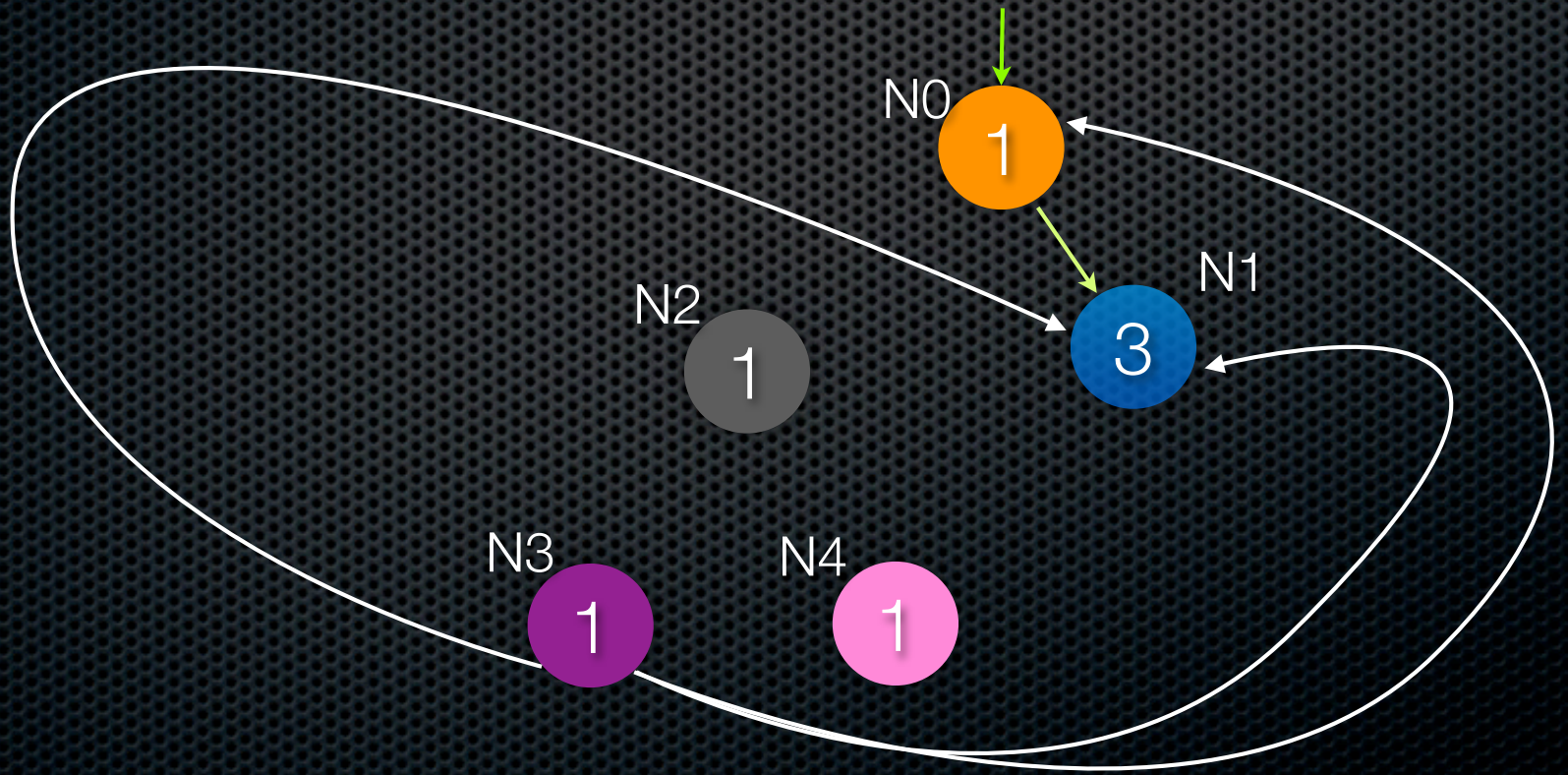
- Algunos problemas en el manejo de la explosión de estructuras posibles son los siguientes:
  - Iteración sobre elementos irrelevantes del espacio de estados de la estructura (e.g., sobre elementos inalcanzables del heap)
  - Estructuras simétricas (i.e., instancias redundantes)

Un Ejemplo: Iteración sobre Elementos no Alcanzables

- Supongamos que queremos testear una rutina que manipula árboles binarios de búsqueda.
  - Nos interesa generar TODOS los árboles binarios de búsqueda de (hasta) 5 elementos (conteniendo 1 a 5).
  - Cada estructura puede representarse por:
    - 1 valor r (nodo raíz)
    - 5 tuplas (v_i, izq_i, der_i), que representan los nodos

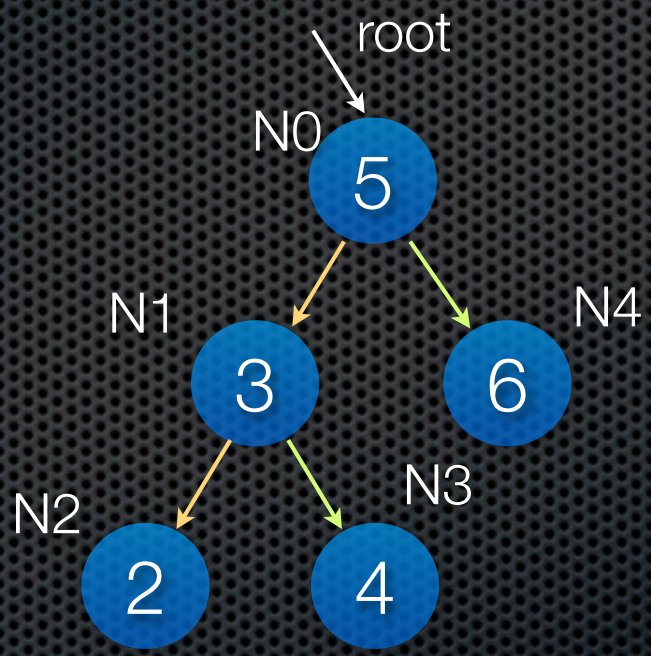
Un Ejemplo: Iteración sobre Elementos no Alcanzables





Estructuras Simétricas

- Consideremos el siguiente árbol:



- Si nos abstraemos de las direcciones específicas de los nodos, un total de 5! estructuras diferentes representan el mismo árbol.

Estado del Arte en la Generación de Casos de Test para Estructuras Complejas

herramientas #1

herramientas #2

- FAJITA (basada en SAT solving)
- Korat (basada en búsqueda)
- Alloy (basada en SAT solving)
- Java PathFinder (basada en model checking)
- Udita (basada en JPF)
- Eclat (basada en la combinación de métodos)
- PKorat (basada en búsqueda paralela)

herramientas #3

Alloy (SAT solving)

- Adecuado para la especificación de propiedades estructurales de sistemas
- Las especificaciones se basan en dominios de datos y operaciones sobre éstos (*alla Z*)
- Las especificaciones son analizables automáticamente, mediante SAT solving
  - PERO, la lógica subyacente a Alloy (extensión de FOL) no es decidible: se requieren cotas en los dominios

Alloy: Ejemplo

```
sig Data { }
one sig Null { }
sig Node {
      val: Data,
      next: Node+Null
}
sig List {
      head: Node+Null
}

fact AcyclicLists {
      all l: List, n: Node | n in l.head.(*next) => n !in n.(^next)
}

assert testGeneration {
      all l: List | l != l
}
```

Alloy (SAT solving)

- Rotura de simetrías: parcial
- Iteración sobre porciones no alcanzables del heap: eliminable via predicados (alcanzabilidad es expresable en el lenguaje)
- Performance general: pobre en comparación con otras técnicas
- Otras desventajas: pobre soporte de aritmética

Korat (Búsqueda)

- Realiza generación exhaustiva acotada (bounded exhaustive) de casos de test para código Java
- Requiere cotas para dominios
- Requiere la especificación imperativa del invariante de representación de la estructura (rutina repOK)
- Basado en búsqueda *depth first search* (backtracking) con potentes mecanismos de poda

Ejemplo de Invariante de Representación

```
public class SinglyLinkedList {

    public Entry header;

    private int size = 0;

    ...
}
```

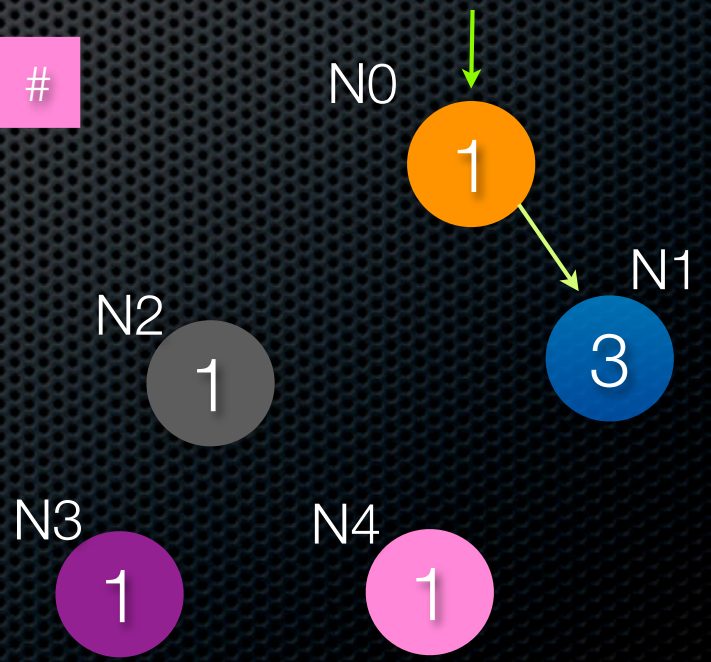
```
public class Entry {

    Object element;

    Entry next;
}
```

```
public boolean repOK() {
    if (header == null)
        return false;
    if (header.element != null)
        return false;
    Set<Entry> visited = new java.util.HashSet<Entry>();
    visited.add(header);
    Entry current = header;
    while (true) {
        Entry next = current.next;
        if (next == null)
            break;
        if (next.element == null)
            return false;
        if (!visited.add(next))
            return false;
        current = next;
    }
    if (visited.size() - 1 != size)
        return false;
    return true;
}
```

Korat: Estrategia de Búsqueda

- Korat realiza una búsqueda de instancias válidas sobre el espacio de instancias posibles
- Representa las instancias con vectores candidatos

- El backtracking se realiza de acuerdo al orden de visita de los elementos de la estructura, según repOK()



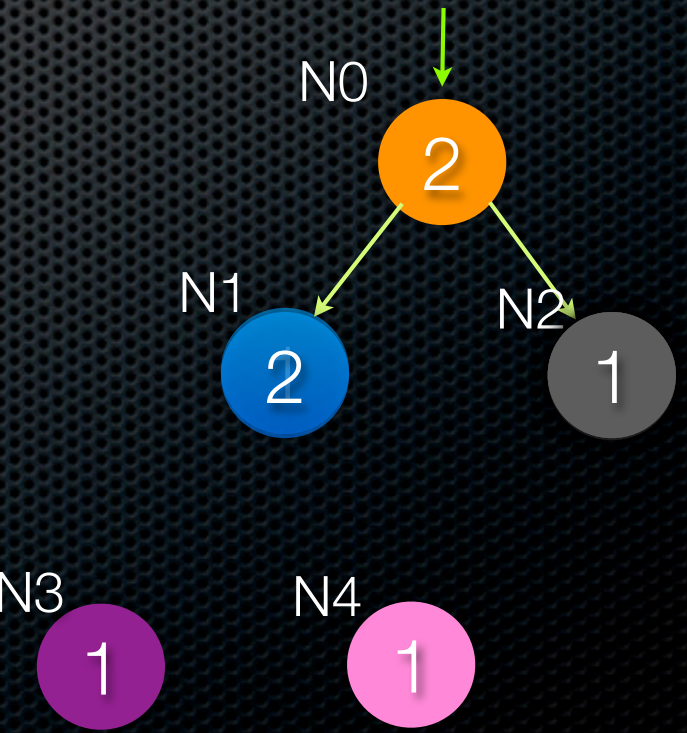
Backtracking en Vectores Candidatos

- Consideremos el siguiente ejemplo (ABB):

N0	1	N1	N2	2	#	#	1	#	#	1	#	#	1	#	#

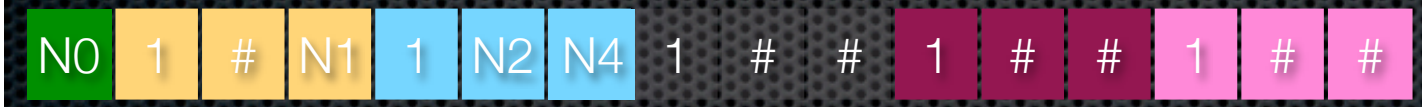
- repOK(): isBinaryTree(); isSorted()

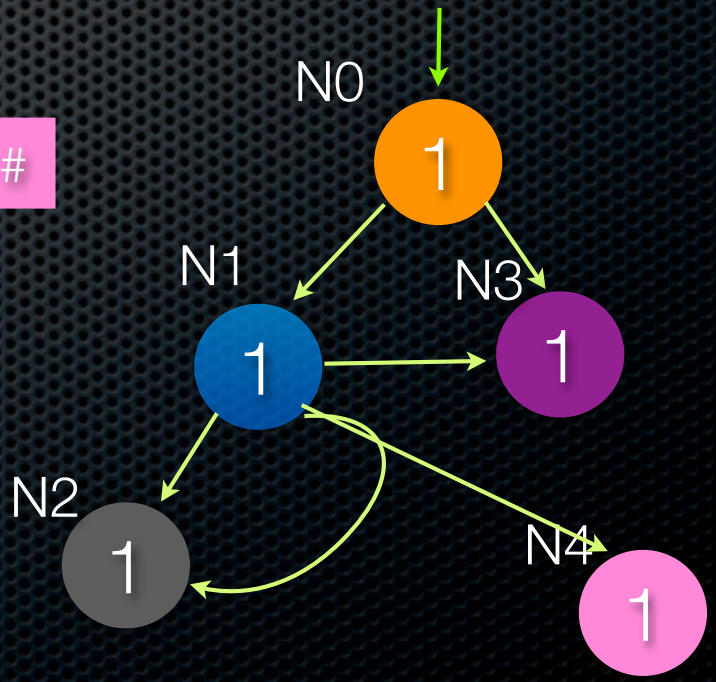
- El backtracking evita iterar sobre porciones no alcanzables de la estructura



Korat: Estrategia de Poda

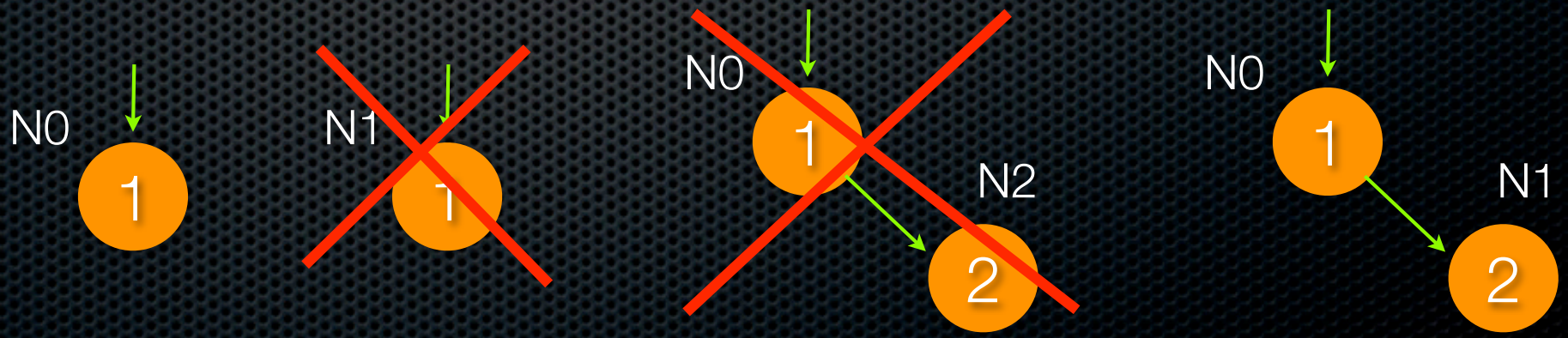
- Korat realiza podas del espacio de estados en casos en los cuales repOK() falla
- Evita en muchos casos visitar espacios grandes de vectores candidatos inválidos





Rotura de Simetrías

- Korat realiza rotura de simetrías mediante el uso de una regla muy simple:
  - “Durante la visita, se puede observar a lo sumo un objeto ‘no tocado’ previamente”
    - El índice de un nodo no puede ser mayor a k+1, con k el mayor índice de los objetos del mismo tipo ya visitados



Aún asi...

- En muchos casos el espacio de búsqueda es extremadamente amplio, incluso para cotas de tamaño pequeño.
- Korat funciona mejor cuando repOK() “falla mucho”
- No funciona bien cuando repOK() triunfa con frecuencia
  - Ejemplo: Testing de Merge para Binomial Heaps

Qué Hacer? Poda guiada por Cobertura (Korat+)

- Se puede llevar el mecanismo de búsqueda y poda de Korat más alla: podar espacios de estados válidos, cuando éstos cubren clases de equivalencias de casos de test ya cubiertas.

Scope	Korat / Korat+	CC
2	348(36) 147(15)	6
3	5,389(784) 1,315(56)	10
4	150,448(14,400) 46,786(435)	10
5	3,125,314(876,096) 647,410(1,872)	10
6	274,808,123(57,790,404) 55,745,855(43,134)	10

Otras Alternativas/Mejoras

- Cotas “ajustadas” para mejorar el análisis basado en SAT (TACO/FAJITA)
- Rotura de simetrías “perfecta” para SAT
- Técnicas basadas en ejecución simbólica para testing de caja blanca (Java PathFinder)
- Paralelización de la búsqueda (PKorat)
- Generación aleatoria de casos de test (Eiffel’s AutoTest)
- ...