Automatic Data Structure Repair using Separation Logic

Guolong Zheng, ThanhVu Nguyen
University of Nebraska-Lincoln

Quang Loc Le
Teesside University

Quoc-Sang Phan
Fujitsu Labs. of America

MVD'18

Introduction

Offline Repair

- Stop program
- Repair source code
- Recompile
- Rerun
- GenProg; Angelix; InferFix

On-the-fly Repair

- Suspend program
- Repair program states
- No need to recompile
- Resume running
- Tarmeem;

StarFix

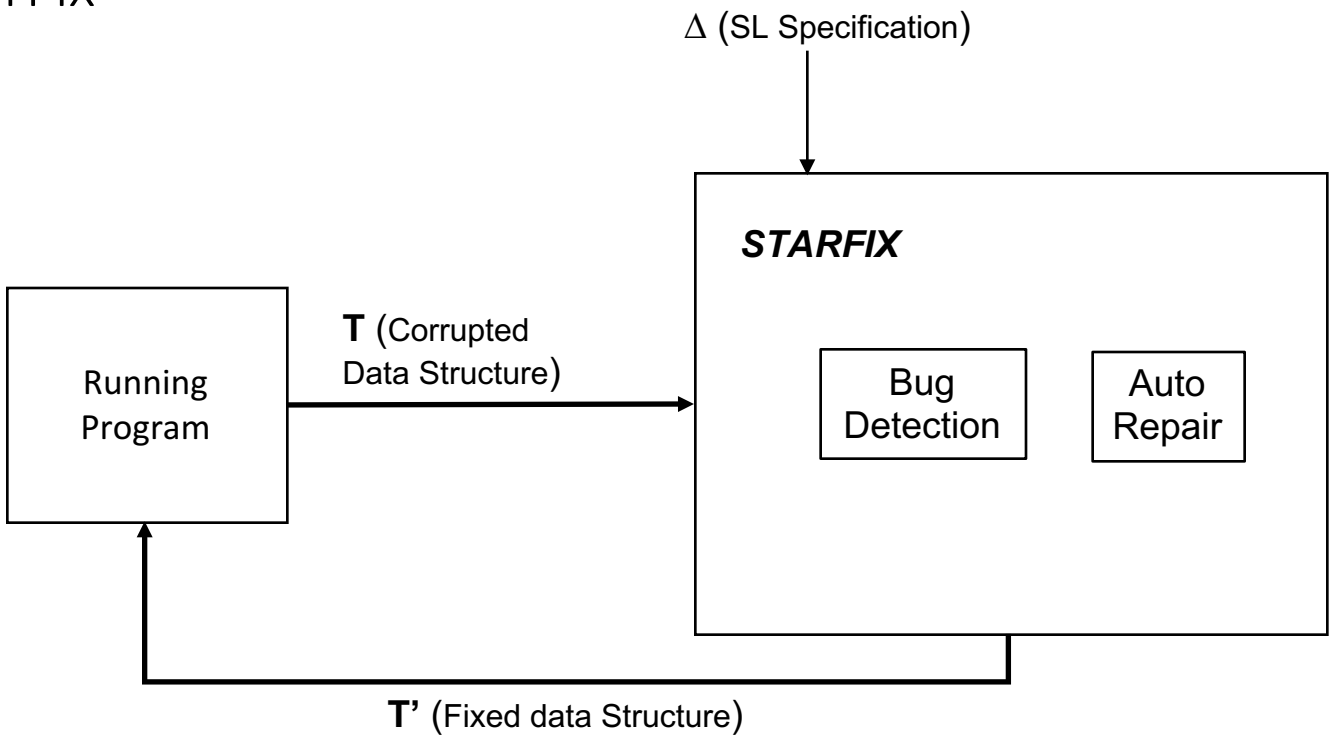
- On-the-fly Repair
  - Repairs corrupted data structure
- Specification based repair
  - Uses Separation Logic
  - Uses STARLIB

Separation Logic

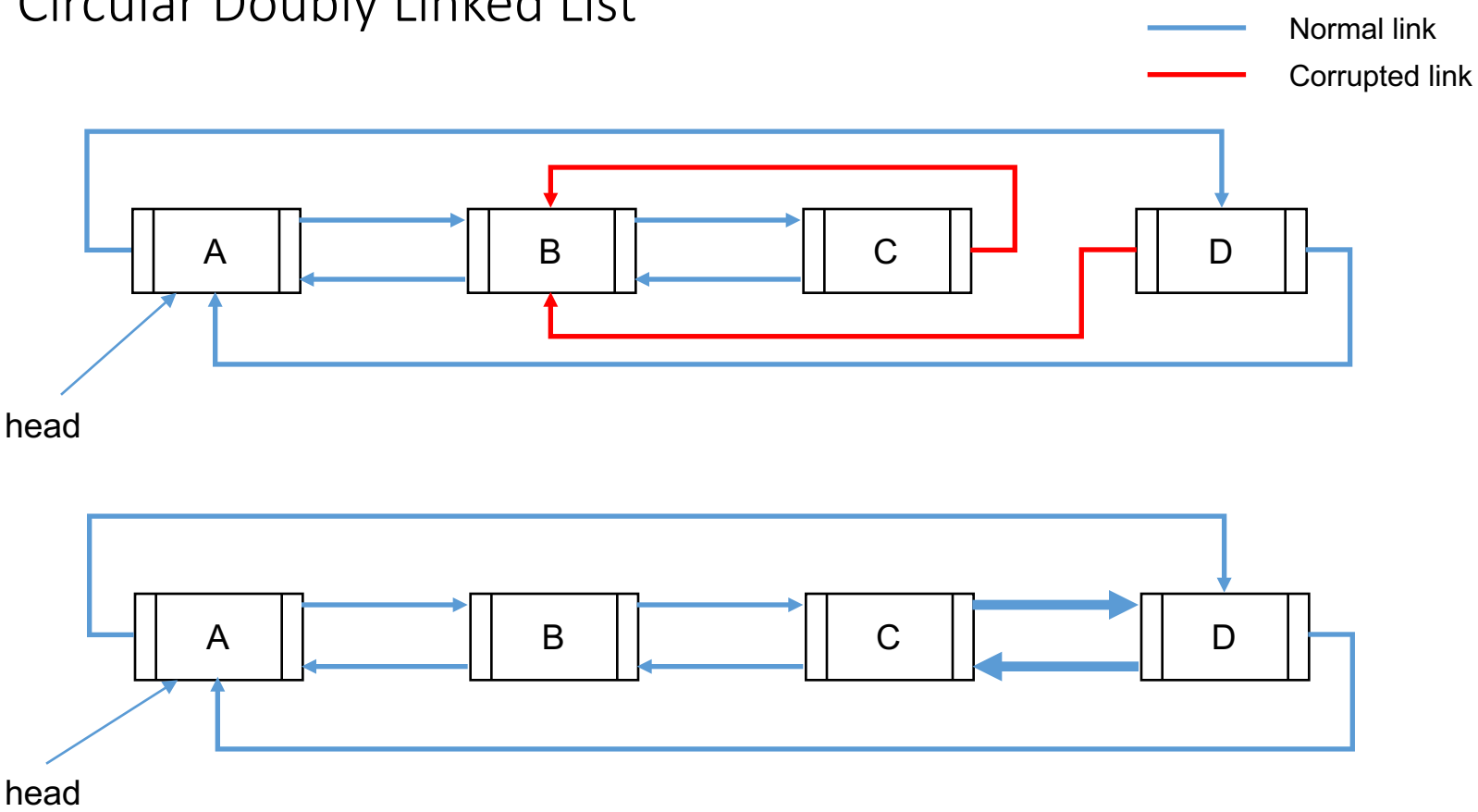
- Extends Hoare Logic
  - Deal with pointers, dynamically allocated memory
  - tree(root) := emp $\wedge$ root = null **OR**

    root|->(left, right) $*$ tree(left) $*$ tree(right)
- Benefits
  - v.s. **repOk**
  - Describes memory shape property
  - Naturally encodes recursively defined data structure
  - Utilizes existing model checking algorithms

StarFix


Δ (SL Specification)
STARFIX
Running Program
T (Corrupted Data Structure)
Bug Detection
Auto Repair
T' (Fixed data Structure)

Circular Doubly Linked List



SL definition for DLL

dll(head)

(**emp** $\wedge$ head=null) empty list

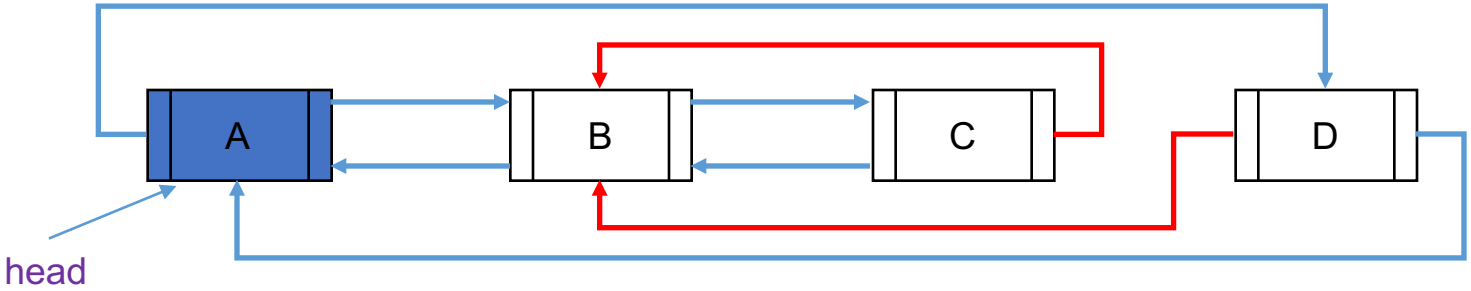
($\exists$ p, n. head $\mapsto$Node(p, n) $*$ lst(head, p, head, n)) inductive list

lst(h, $prev_h$, cur, next)

(**emp** $\wedge$ $prev_h$=cur & next=h) cur is the tail

($\exists$ n. next $\mapsto$Node(cur, n) $*$ lst(h, $prev_h$, next, n)) inductive sublist

Unfolding dll(head)



Symbolic Heap

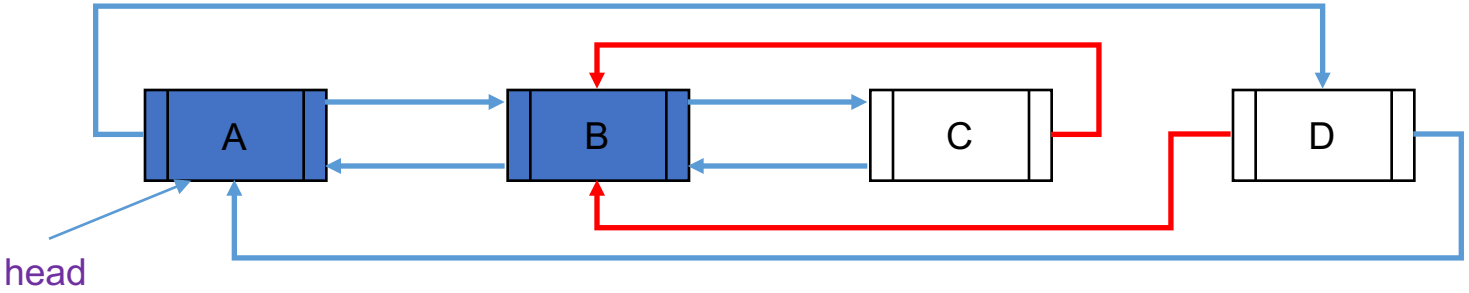
$\Delta 1 = \mathbf{emp} \wedge \text{head=null}$ ✗

$\Delta 2 = \exists p_1, n_1 .\ \text{head} \mapsto \text{Node}(p_1, n_1) * \text{lst}(\text{head}, p_1, \text{head}, n_1))$ ✓

Concrete Model

$M_0 \equiv \{\text{head=A};\ p_1\text{=D};\ n_1\text{=B}\}$

Unfolding Δ2



Symbolic Heap

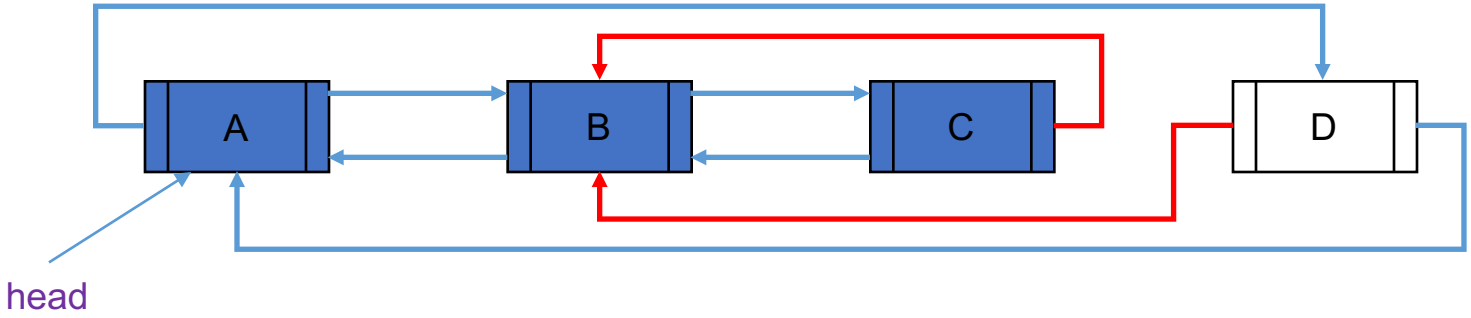
$\Delta 3 \equiv \exists p_1, n_1.\ head \mid\!\rightarrow Node(p_1, n_1) \wedge p_1 = head \wedge head = n_1$ ✗

$\Delta 4 \equiv \exists p_1, n_1, n_2.\ head \mid\!\rightarrow Node(p_1, n_1) * n_1 \mid\!\rightarrow Node(head, n_2) * lst(head, p1, n_1, n2)$ ✓

Concrete Model

$M_1 \equiv \{head=A;\ p_1=D;\ n_1=B;\ n_2=C\}$

Unfolding Δ4



Symbolic Heap

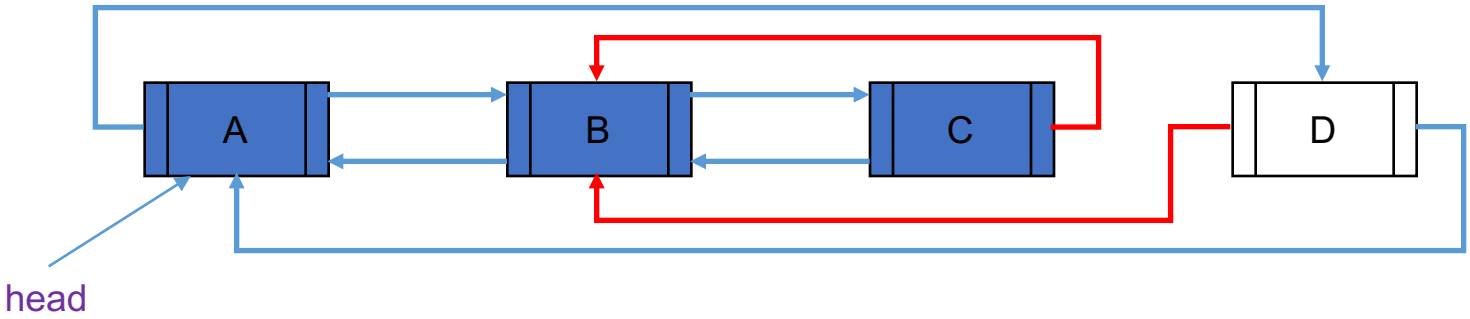
$\Delta 5 \equiv \exists p_1,n_1,n_2.\ head \mid\!\rightarrow Node(p_1, n_1) * n_1 \mid\!\rightarrow Node(head,n_2)$ ✗
$\wedge\ p_1 = n_1 \wedge n_2 = head$

$\Delta 6 \equiv \exists p_1,n_1,n_2,n_3.\ head \mid\!\rightarrow Node(p_1, n_1) * n_1 \mid\!\rightarrow Node(head,n_2)$ ✓
$* n_2 \mid\!\rightarrow Node(n_1, n_3) * lst(head, p_1, n_2, n_3)$

Concrete Model

$M_2 \equiv \{head=A;\ p_1=D;\ n_1=B;\ n_2=C; n_3=B\}$

Unfolding Δ6



Symbolic Heap

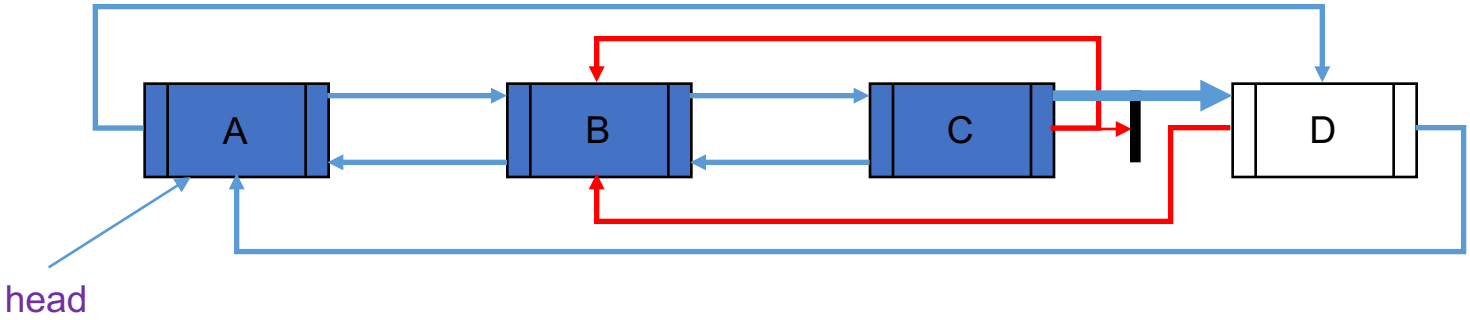
$\Delta 7 \equiv \exists\, p_1,B,n_2,n_3.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1,n_3) \wedge p_1=n_2 \wedge n_3=head$ ✗

$\Delta 8 \equiv \exists\, p1,B,n2,n3,n_4.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1, n_3) * n_3|\rightarrow Node(n_2, n_4) * lst(head, p_1, n_3, n_4)$ ✗

Concrete Model

$M_2 \equiv \{head=A;\ p_1=D;\ n_1=B;\ n_2=C;n_3=B\}$

Roll back to Δ6 and Repair n3



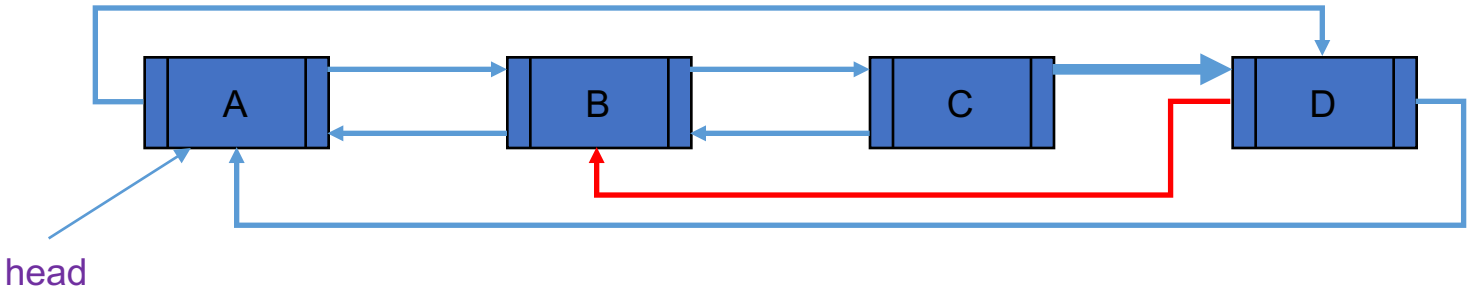
Symbolic Heap

$\Delta 6 \equiv \exists\, p_1,B,n_2,n_3.\mathrm{head}\mid\!\rightarrow \mathrm{Node}(p_1, n_1) * n_1\mid\!\rightarrow \mathrm{Node}(\mathrm{head},n_2) * n_2\mid\!\rightarrow \mathrm{Node}(n_1, n_3) * \mathrm{lst}(\mathrm{head}, p_1, n_2, n_3)$

Concrete Model

$M_2' \equiv \{\mathrm{head}=A;\ p_1=D;\ B=B;\ n_2=C;\ n_3=(\ \mathrm{null} \mid D \mid A \mid C)\}$

Unfolding Δ6



Symbolic Heap

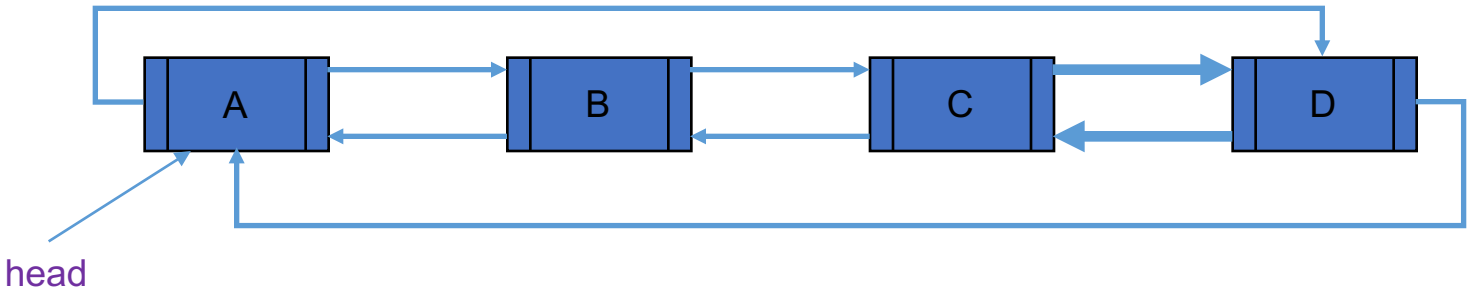
$\Delta 7 \equiv \exists\, p_1,B,n_2,n_3.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1,n_3) \wedge p_1=n_2 \wedge n_3=head$ ✗

$\Delta 8 \equiv \exists\, p1,B,n2,n3,n_4.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1, n_3) * n_3|\rightarrow Node(n_2, n_4) * lst(head, p_1, n_3, n_4)$ ✗

Concrete Model

$M_3 \equiv \{head=A; p_1=D; n_1=B; n_2=C; n_3=D; n_2=B; n_4=A\}$

Repairing n_2



Symbolic Heap

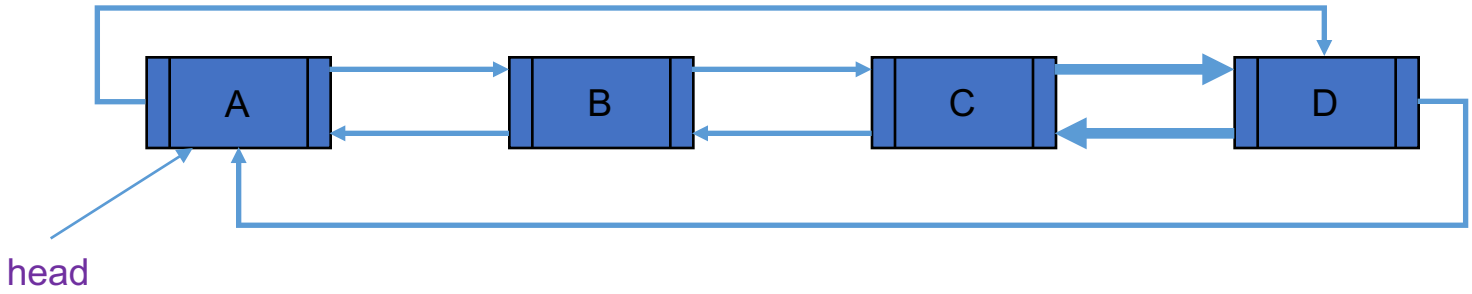
$\Delta 7 \equiv \exists\, p_1,B,n_2,n_3.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1,n_3) \wedge p_1=n_2 \wedge n_3=head$ ✗

$\Delta 8 \equiv \exists\, p1,B,n2,n3,n_4.\ head|\rightarrow Node(p_1, n_1) * n_1|\rightarrow Node(head,n_2) * n_2|\rightarrow Node(n_1, n_3) * n_3|\rightarrow Node(n_2, n_4) * lst(head, p_1, n_3, n_4)$ ✓

Concrete Model

$M_3 \equiv \{head=A;\ p_1=D;\ n_1=B;\ n_2=C;n_3=D;n_4=A\}$

Unfolding Δ8



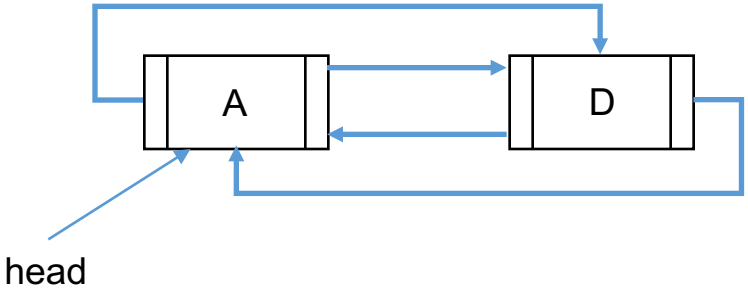
Symbolic Heap

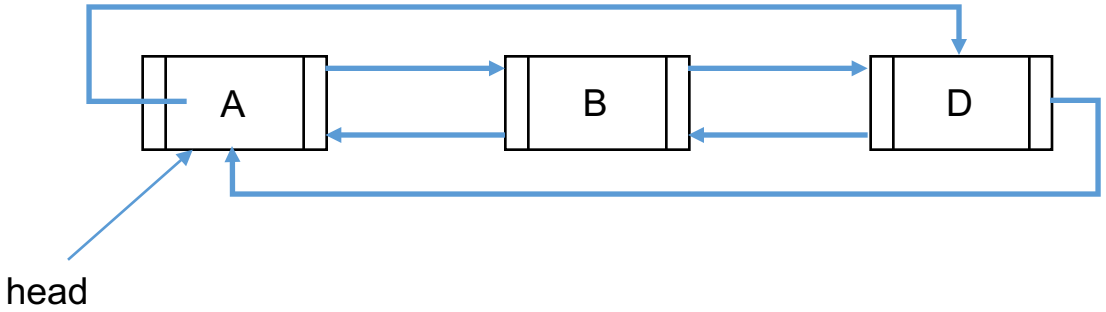
$\Delta 9 \equiv \exists$ p1,B,n2,n3,n_4. head$|\rightarrow$Node(p_1, n_1) $*$ $n_1|\rightarrow$Node(head,n_2)
$*$ $n_2|\rightarrow$Node(n_1, n_3) $*$ $n_3|\rightarrow$Node(n_2, n_4)
$\wedge p_1 = n_3 \wedge$ head=n_4

Concrete Model

$M_3 \equiv$ {head=A; p_1=D; n_1=B; n_2=C;n_3=D;n_4=A}

Multiple Valid Fixes



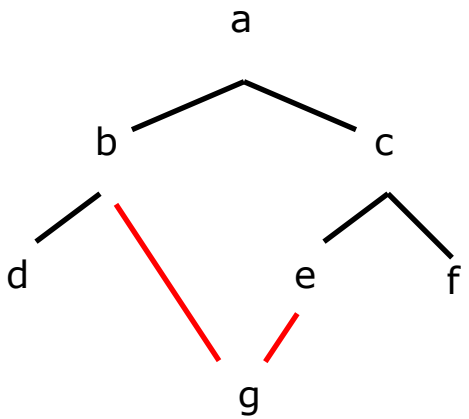


Algorithm

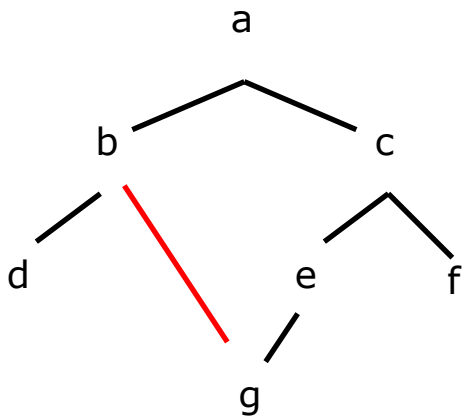
StarFix (SL spec Δ, Data StructureT)

- Check T $\Longrightarrow$ Δ
  - Unfold Δ one depth to find a match
    - True: return T
    - False: repair T
    - Neither: keep unfolding
- Repair T
  - Modify T to T’
  - Check T'

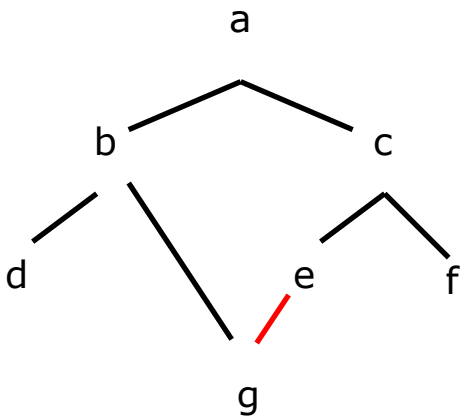
Repairing a Binary Tree



Corrupted Tree

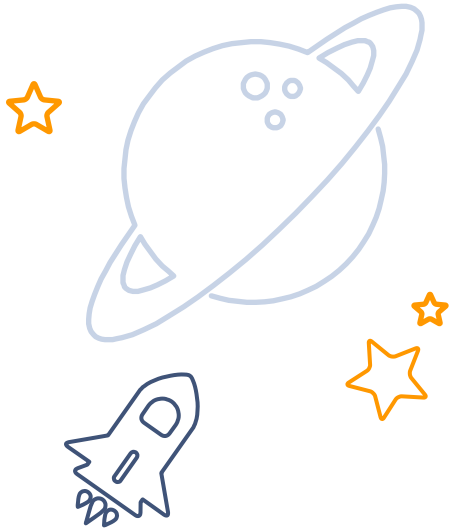


One fix



Another Fix

Future Work



- Use more powerful SL model checker
- Ranking fixes
- Heuristics to optimize repairing performance
- Automatically inferring specifications from good program runs

THANKS!

Any questions?