Test Case Generation for Heap Inputs using Separation Logic

Quang Loc Le
A joint work with many collaborators

NII Shonan Meeting Seminar 100, Japan

Oct 2, 2017

Test Case Generation for Heap Inputs

- Input: a Java program and its Precondition
- Output: Valid test cases
  - Goal: high coverage

Test Case Generation for Heap Inputs

Approach: Symbolic Execution

- Path condition
- Branching
- SAT solver

Test Case Generation for Heap Inputs

Symbolic Execution with Lazy Initialization

- JPF - 2003: Assign values to heap inputs on demand
  1. $x \leftarrow$ `null`
  2. $x \leftarrow$ `currentObj`
  3. $x \leftarrow$ `newObj`

- BBE - 2004: with repOK

- JBSE - 2015: with HEX logical precondition

Test Case Generation for Heap Inputs

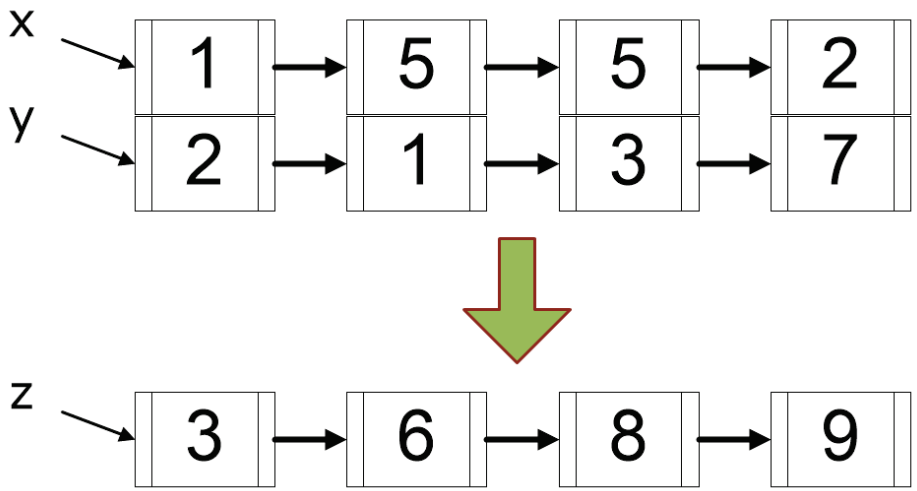
Symbolic Execution with Lazy Initialization

- JPF - 2003
- BBE - 2004
- JBSE - 2015: with logical precondition for validation
  - only regular shape
  - no pure properties
  - bounded - unsound SAT for induction

Test Case Generation for Heap Inputs

- Symbolic Execution
- Lazy Initialization with Least Fixed Point
- SAT solver with induction reasoning

Add two numbers represented by linked lists



$$\texttt{pred}\ \texttt{list\_pair}(a, b) \equiv \texttt{emp} \wedge a = \texttt{null} \wedge b = \texttt{null}$$
$$\vee\ \exists n_1, n_2 . a \mapsto \mathit{Node}(\_, n_1) * b \mapsto \mathit{Node}(\_, n_2) * \texttt{list\_pair}(n_1, n_2)$$

Add two numbers represented by linked lists

Input:
Program

```
Node add(Node x, Node y){
 Node dummyHead = new Node(0, null);
 Node z = dummyHead;
 while(x != null) {
  z.next = new Node(x.next + y.next, null);
  x = x.next;
  y = y.next; z = z.next;}
 return dummyHead.next;}
```

Precondition
$\texttt{list\_pair}(x, y)$

Output: Test Cases
$X{=}\texttt{null} \wedge Y{=}\texttt{null}$
$X \mapsto \mathit{Node}(\_, \texttt{null}) * Y \mapsto \mathit{Node}(\_, \texttt{null})$

Add two numbers represented by linked lists

```
1  Node add(Node x, Node y){
2   Node dummyHead = new Node(0, null);
3   Node z = dummyHead;
4   while(x != null) {
5    z.next = new Node(x.next + y.next, null);
6    x = x.next;
7    y = y.next; z = z.next;}
8   return dummyHead.next;}
```

$pc : \exists D, Z.\texttt{list\_pair}(X, Y) * D \mapsto Node(\_, \texttt{null}) \wedge Z = D$

$pc : \exists D, Z.(X = \texttt{null} \wedge Y = \texttt{null}) * D \mapsto Node(\_, \texttt{null}) \wedge Z = D$

$pc : \exists D, Z, N_1, N_2.X \mapsto Node(\_, N_1) * Y \mapsto Node(\_, N_2) * \texttt{list\_pair}(X, Y) *$
$D \mapsto Node(\_, \texttt{null}) \wedge Z = D$

Experimental Results

- benchmarks: 74 methods - *Singly Linked List*, *Doubly Linked List*, *Stack*, *Binary Search Tree*, and *Red Black Tree* from SIR; *AVL Tree* and *AA Tree* from Sierum/Kiasan, and *Gantt* project from SUSHI (ISSTA 2017).

- Valid Test: BBE (8.14%), JBSE (0.72%), ours (100%)

- Coverage: BBE (38.01%), JBSE (33.23%), ours (99.1%)

1 Program Testing

2 SAT Solver
- Syntax
- Problem
- Decidable Fragment

3 Conclusion

A fragment of Separation Logic

$$
\begin{array}{lll}
\text{Formula} & \Phi & ::= \Delta \mid \Phi_1 \vee \Phi_2 \qquad\qquad \Delta ::= \exists \bar{v}.\ (\kappa \wedge \pi) \\
\text{Spatial formula} & \kappa & ::= \mathtt{emp} \mid x \mapsto c(v_i) \mid \mathtt{P}(\bar{v}) \mid \kappa_1 * \kappa_2 \\
\text{Pure formula} & \pi & ::= \pi_1 \wedge \pi_2 \mid \alpha \mid \phi
\end{array}
$$

- α: Pointer (Dis)Equalities
- ϕ: Presburger arithmetic
- $\mathtt{P}$: inductive predicate. Predicate Definition: $\mathtt{P}(\bar{t}) \equiv \Phi$

Warning: no pointer arithmetic and no magic wand

Satisfiability Problem

- Input: A formula Δ in the fragment
- Question: Is Δ satisfiable?

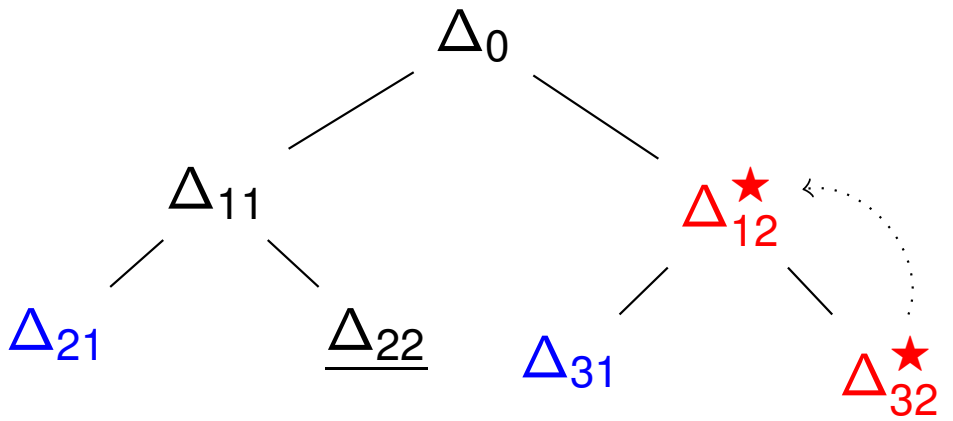
Challenges:

- Unbounded heaps
- Infinite numerical domain

Proof by Induction

- Base case
- Induction case

Cyclic Proof (J. Brotherston - UCL, J. Jaffa et. al. - NUS)



- Weaken Δ_{32} to Δ'_{32}
- Find σ s.t. $\Delta'_{32}\sigma \Rightarrow \Delta_{12}$

Cyclic Proof

- From Entailment Problem ($\Delta_a \vdash \Delta_c$) to Satisfiability Problem ($\Delta_a \vdash \texttt{false}$)

- Shape and Integer domains
  - link back simultaneously (CAV 2016)
  - Shape then Integer (CAV 2017)

Our Approach - CAV 2017

Decision Procedure: Base Computation
Compute for each inductive predicate a finite representation that precisely characterises its satisfiability.

Base of Inductive Predicate: Example 1

- Inductive predicate: Singly-linked list with size property

$$\texttt{pred}\ \mathit{ll\_size}(\texttt{root},n) \equiv \texttt{emp}\wedge\texttt{root}=\texttt{null}\wedge n=0 \vee \exists\, r,n_1 \cdot \texttt{root}\mapsto \mathit{node}(\_,r) * \mathit{ll\_size}(r,n_1) \wedge n=n_1+1$$

- Example:

$$\texttt{base}^{\mathcal{P}}(\texttt{ll\_size}(\texttt{root},\texttt{n}))\equiv \{\texttt{emp}\wedge\texttt{root}=\texttt{null}\wedge n=0, \texttt{root}\mapsto \mathit{node}(\_,\_)\wedge n>0\}$$

Projections

- Inductive predicate: Singly-linked list with size property

$$\texttt{pred}\ \mathit{ll\_size}(\texttt{root},n) \equiv \texttt{emp}\wedge\texttt{root}=\texttt{null}\wedge n=0 \vee \exists\, r,n_1 \cdot \texttt{root}\mapsto \mathit{node}(\_,r) * \mathit{ll\_size}(r,n_1) \wedge n=n_1+1$$

- Spatial projection

$$\mathit{ll\_size}^S(\texttt{root}) \equiv \texttt{emp} \wedge \texttt{root}=\texttt{null} \vee \exists\, r \cdot \texttt{root}\mapsto \mathit{node}^S(r) * \mathit{ll\_size}^S(r)$$

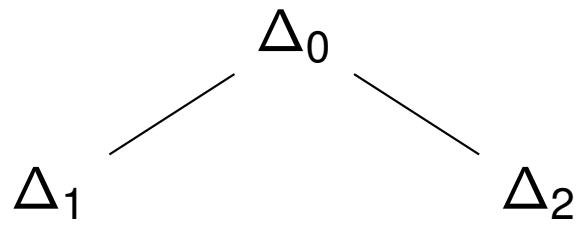
- Numerical projection

$$\mathit{ll\_size}^N(n) \equiv n=0 \vee \exists\, n_1 \cdot \mathit{ll\_size}^N(n_1) \wedge n=n_1+1$$

Phase 1: Cyclic Tree for Spatial projection

$$\mathit{ll\_size}^{S}(\texttt{root}) \equiv \texttt{emp} \wedge \texttt{root=null} \vee \exists\, r \cdot \texttt{root} \mapsto \mathit{node}^{S}(r) * \mathit{ll\_size}^{S}(r)$$

$\Delta_0 \equiv \mathit{ll\_size}^{S}(\texttt{root})$
$\Delta_1 \equiv \texttt{emp} \wedge \texttt{root=null}$
$\Delta_2 \equiv \exists\, r \cdot \texttt{root} \mapsto \mathit{node}^{S}(r) * \mathit{ll\_size}^{S}(r)$



$\{\texttt{emp} \wedge \texttt{root=null},$
$\texttt{root} \mapsto \mathit{node}(\_,\_)\}$

Why not continue unfolding?

Foundation of Base Computation

- For each formula, eliminating existentially quantified pointer-typed variables produces an equi-satisfiable formula.

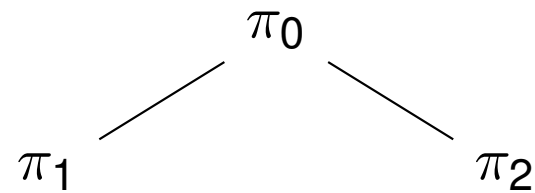
- Example: $\Delta_2 \equiv \exists\, r \cdot \texttt{root} \mapsto \mathit{node}^S(r) * \mathit{ll\_size}^S(r)$
  is equi-satisfiable with
  $\Delta^b{}_2 \equiv \exists\, r \cdot \texttt{root} \mapsto \mathit{node}^S(r)$

Phase 2: Cyclic Tree for Numeric projection

$$ll\_size^N(n) \equiv n{=}0 \vee \exists\, n_1 \cdot ll\_size^N(n_1) \wedge n{=}n_1{+}1$$

Cyclic Tree for Numeric Projection is the same unfolding pattern to the one for Spatial Projection

$$\pi_0 \equiv ll\_size^N(n)$$
$$\pi_1 \equiv n{=}0$$
$$\pi_2 \equiv \exists\, n_1 \cdot ll\_size^N(n_1) \wedge n{=}n_1 + 1$$



$\{n{=}0, n{>}0\}$

find closure form of $ll\_size^N(n_1)$.

Base Computation

Finite Representation: Base Formula (without inductive predicates)

- Combining empty heap ($\mathtt{emp}$), points-to ($\mapsto$), spatial conjunction ($*$) and Presburger Arithmetic

- Example:

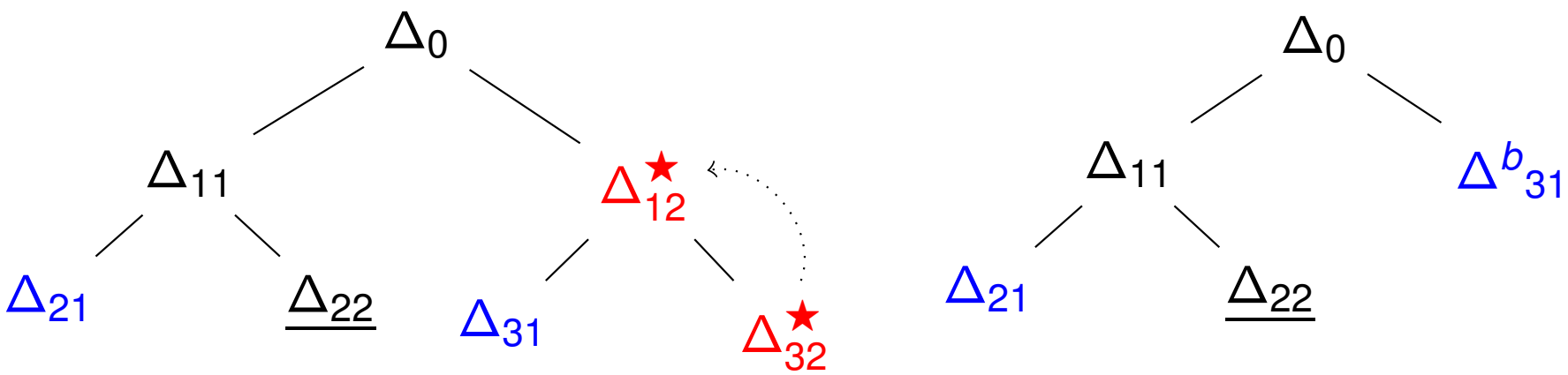
$$\begin{array}{ll} \mathtt{SAT} & \Delta_1 \equiv \mathtt{emp} \wedge x = \mathtt{null} \wedge n = 0 \\ \mathtt{UNSAT} & \Delta_2 \equiv x \mapsto node(n, y) * y \mapsto node(n-1, \mathtt{null}) \wedge x = y \end{array}$$

The fragment of base formulas is decidable
(Piskac, Wies and Zufferey - CAV 2013, Navarro and Rybalchenko - APLAS 2013)

Base Computation

Given an inductive predicate $P(\bar{x}) \equiv \Phi$,

1. Construct a cyclic unfolding tree for $\Delta_0 \equiv P(\bar{x})$
2. Flatten the tree into a disjunctive set of base formulas

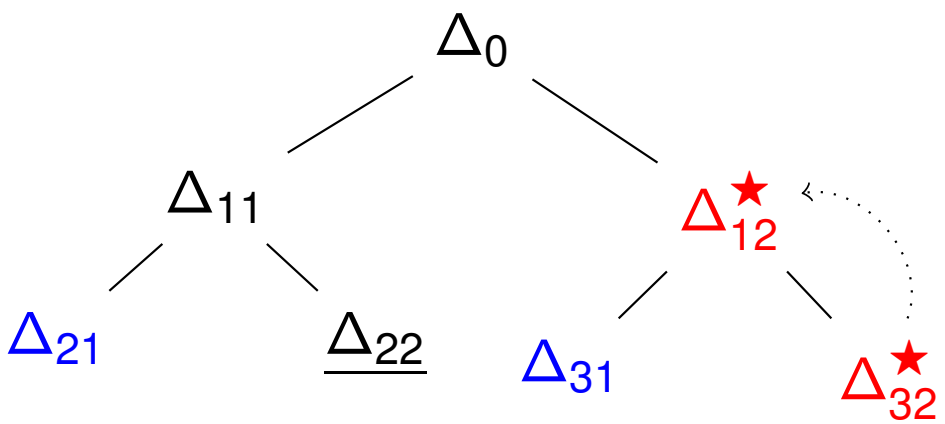


$$\mathtt{base}^{\mathcal{P}}(P(\bar{x})) \equiv \{\Delta_{21}, \Delta^{b}_{31}\}$$

Constructing Cyclic Unfolding Tree

Given an inductive predicate $P(\bar{x}) \equiv \Phi$, construct a unfolding tree for $\Delta_0 \equiv P(\bar{x})$ through iterations of actions:

1. Choose a (open) leaf, close it if
   - it can be reduced into a base formula.
     - a base formula
     - a formula in which pointer-typed parameters of every inductive predicates are existentially quantified.
   - its over-approximation is unsat.
   - can be linked back to form a circular path.
2. Otherwise, unfold it.



Example 2: Constructing Cyclic Unfolding Tree

$$\texttt{pred}\ \mathtt{Q}(x,y,n) \equiv \exists\, y_1.x{\mapsto}node(\texttt{null},y_1){\wedge}y{=}\texttt{null}{\wedge}x{\neq}\texttt{null}{\wedge}n{=}1$$
$$\vee\ \exists\, x_1,y_1,n_1.y{\mapsto}node(x_1,y_1) * \mathtt{Q}(x,y_1,n_1){\wedge}y{\neq}\texttt{null}{\wedge}n{=}n_1{+}2;$$

$$\Delta_0 \equiv \mathtt{Q}(x,y,n)$$

1. Base Detection. None
2. Over-Approximation. $\pi_0 \equiv \texttt{true}$.
   Not UNSAT
3. Cyclic Detection. None

Δ_0

Figure : Unfolding Tree $\mathcal{T}_0$.

Example 2: Constructing Cyclic Unfolding Tree

$$\texttt{pred}\ Q(x,y,n) \equiv \exists\, y_1.x{\mapsto}node(\texttt{null},y_1){\wedge}y{=}\texttt{null}{\wedge}x{\neq}\texttt{null}{\wedge}n{=}1$$
$$\vee\ \exists\, x_1,y_1,n_1.y{\mapsto}node(x_1,y_1) * Q(x,y_1,n_1){\wedge}y{\neq}\texttt{null}{\wedge}n{=}n_1{+}2;$$

$$\Delta_2{\equiv}\exists\, x_1,y_1,n_1.y{\mapsto}node(x_1,y_1) * Q(x,y_1,n_1){\wedge}y{\neq}\texttt{null}{\wedge}n{=}n_1{+}2$$
$$\Delta_3{\equiv}\exists\, x_1,y_1,n_1,y_2.y{\mapsto}node(x_1,y_1) * x{\mapsto}node(\texttt{null},y_2) \wedge$$
$$y_1{=}\texttt{null}{\wedge}x{\neq}\texttt{null}{\wedge}n_1{=}1{\wedge}y{\neq}\texttt{null}{\wedge}n{=}n_1{+}2$$
$$\Delta_4{\equiv}\exists\, x_1,y_1,n_1,x_2,y_2,n_2.y{\mapsto}node(x_1,y_1){*}y_1{\mapsto}node(x_2,y_2){*}$$
$$Q(x,y_2,n_2) {\wedge}y_1{\neq}\texttt{null}{\wedge}n_1{=}n_2{+}2{\wedge}y{\neq}\texttt{null}{\wedge}n{=}n_1{+}2$$

1. Base Detection. Δ_3
2. Over-Approximation. $\pi_4{\equiv}$.....
   Not UNSAT
3. Cyclic Detection. Yes

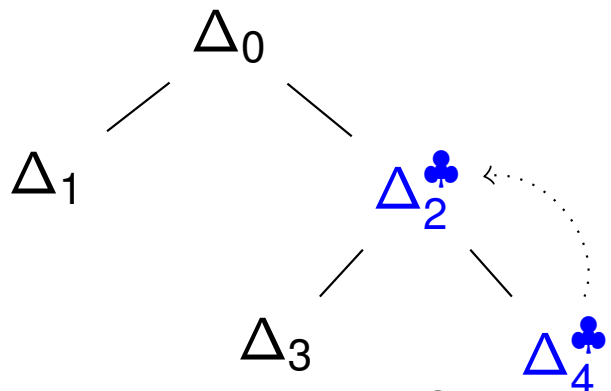


Figure : $\mathcal{T}_2^Q$.

Example 2: Constructing Cyclic Unfolding Tree

Cyclic Detection

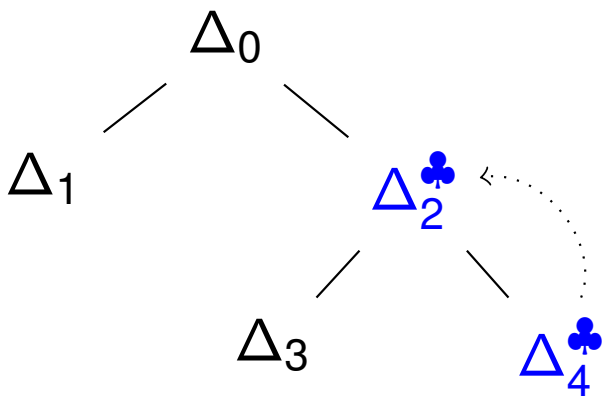
$$\Delta_2 \equiv \exists\, x_1, y_1, n_1 .\, y \mapsto node(x_1, y_1) * \mathcal{Q}(x, y_1, n_1) \wedge y \neq \texttt{null} \wedge n = n_1 + 2$$

$$\Delta_4 \equiv \exists\, x_1, y_1, n_1, x_2, y_2, n_2 .\, y \mapsto node(x_1, y_1) * y_1 \mapsto node(x_2, y_2) * \mathcal{Q}(x, y_2, n_2) \wedge y_1 \neq \texttt{null} \wedge n_1 = n_2 + 2 \wedge y \neq \texttt{null} \wedge n = n_1 + 2$$

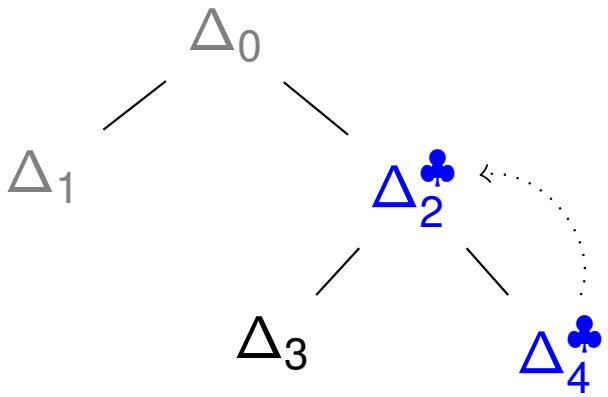
Steps

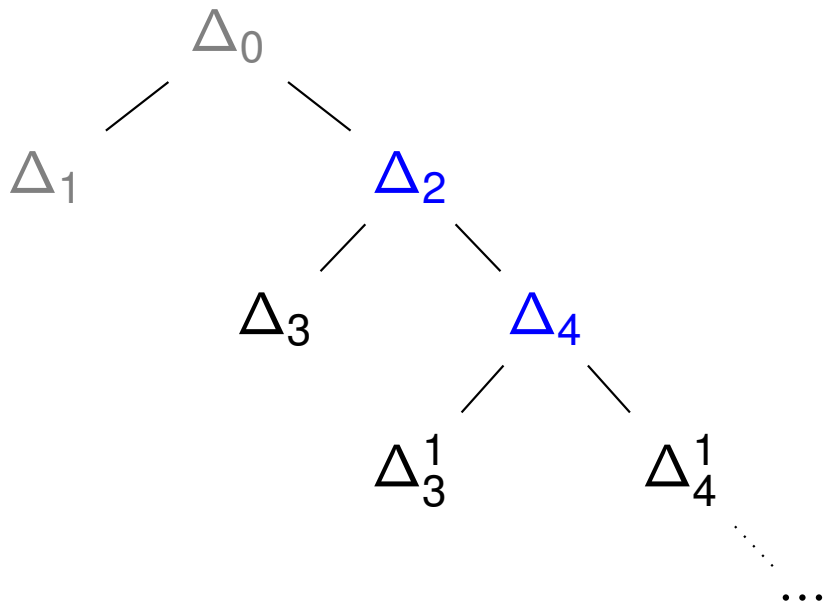
1. matching externally visible points-to predicate: $y \mapsto node(\_, \_)$
2. matching externally visible inductive predicates: $\mathcal{Q}(x, \_, \_)$
   - In general, we may need to group isomorphic inductive predicates beforehand (same predicate name and same sequence of free arguments)
3. matching externally visible (dis)equalities over pointers: $y \neq \texttt{null}$

Example 2: Flattening Cyclic Unfolding Tree



Example 2: Flattening Cyclic Unfolding Tree



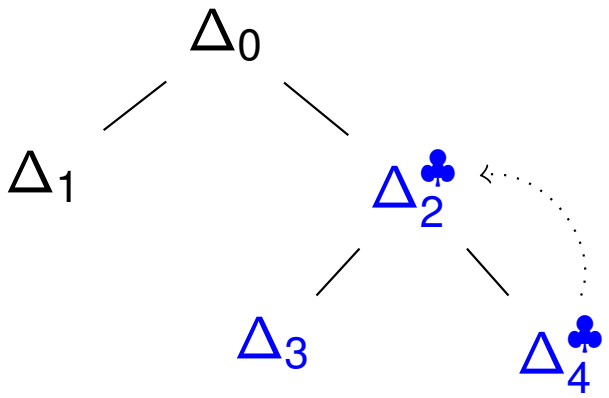


$\Delta_3^{flat} \equiv \Delta_3 \vee \Delta_3^1 \vee ...$

$\Delta_3 \equiv \exists\, x_1, y_1, n_1, y_2 . (y \mapsto node(x_1, y_1) * x \mapsto node(\texttt{null}, y_2) \wedge x \neq \texttt{null} \wedge y \neq \texttt{null} \wedge n = n_1 + 1) \wedge (y_1 = \texttt{null} \wedge n_1 = 1)$

$\Delta_3^1 \equiv \exists\, x_1, y_1, n_1, x_2, y_2, n_2, y_3 . (y \mapsto node(x_1, y_1) * x \mapsto node(\texttt{null}, y_3) \wedge x \neq \texttt{null} \wedge y \neq \texttt{null} \wedge n = n_1 + 1) * (y_1 \mapsto node(x_2, y_2) \wedge y_2 = \texttt{null} \wedge \underline{n_1 = n_2 + 2} \wedge n_2 = 1)$

Example 2: Flattening Cyclic Unfolding Tree



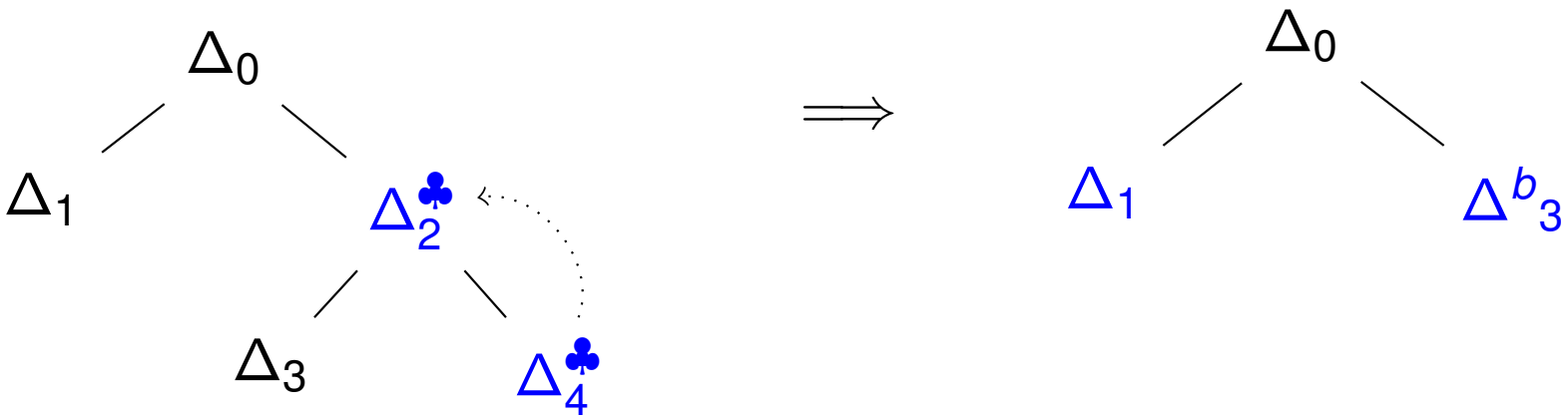
$$\mathtt{P}_{\mathtt{cyc}}(n_1) \equiv n_1 = 1 \vee \exists n_2 . n_1 = n_2 + 2 \wedge \mathtt{P}_{\mathtt{cyc}}(n_2)$$

$$\mathtt{P}_{\mathtt{cyc}}(n_1) \equiv \exists k . n_1 = 2k + 1 \wedge k \geq 0$$

$\Delta^b{}_3$ is equi-satisfiable to Δ_3^{flat}:

$$\Delta^b{}_3 \equiv \exists\, x_1, y_1, x_2, y_2, n_1 . (y \mapsto node(x_1, y_1) * x \mapsto node(\mathtt{null}, y_2) \wedge x \neq \mathtt{null} \wedge y \neq \mathtt{null} \wedge n = n_1 + 1) \wedge (\exists k . n_1 = 2k + 1 \wedge k \geq 0)$$

Flattening Cyclic Unfolding Tree



$\mathtt{base}^{\mathcal{P}}(\mathtt{Q}(\mathtt{x},\mathtt{y},\mathtt{n}))\equiv\{\Delta_1, \Delta^b{}_3\}$

Proposed Decidable Fragment

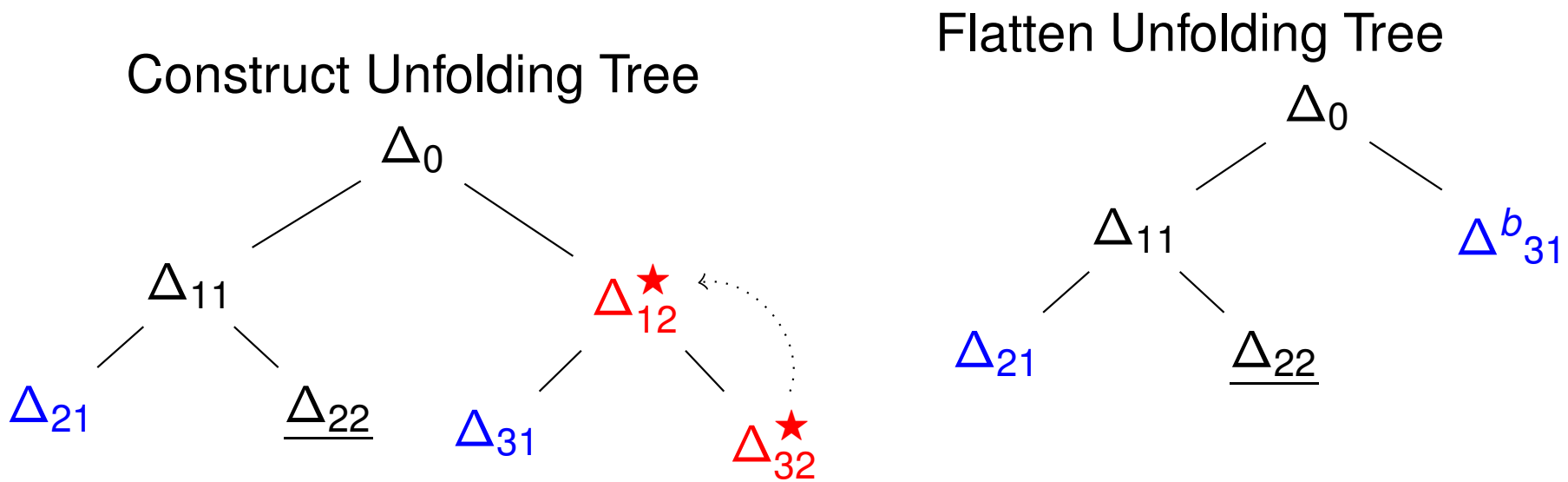
- An inductive predicate is in the proposed decidable fragment if all
  - numerical projections of base leaves; and
  - P_{cyc} predicates

  are Presburger-definable (i.e., can be computed as Presburger formulas).

- Some systems of arithmetic inductive predicates are Presburger-definable:
  - DPI (Tatsuta *et. al.* - APLAS 2016)
  - periodic sets (Bozga *et. al.* - CAV 2010)

Conclusion

- Test Input Generation using Separation Logic
- A decision procedure for an extensible decidable fragment in separation logic including general inductive predicates and arithmetic
- Base Computation:



$$\mathtt{base}^{\mathcal{P}}(\mathtt{P}(\bar{\mathtt{v}})) \equiv \{\Delta_{21}, \Delta^{b}_{31}\}$$

Future Work

SAT solver

- array separation logic with inductive predicates
- extension of separation logic with string logic

Cyclic proof: ENT to SAT and now back to ENT

- for bi-abduction problem
- completeness