A Decidable Logic for Tree Data-Structures with Measurements

PURDUE UNIVERSITY

Yanjun Wang
School of Electrical and Computer Engineering, Purdue University

Collaborated with
Advisor: Xiaokang Qiu

Overview

Logical reasoning about tree data-structures has been needed in various scenarios such as program verification, compiler optimization and program synthesis.

WHAT are measurements?

Height, Size, Black Height for Red-Black Trees…

- tree balancing routine does not increase the height of tree
- compiler optimizer always reduces the size of the program

WHY are they challenging?

- Aggregate functions determined by whole tree
- Tangled with data properties (e.g., sortedness) and shape properties (e.g., balancedness)

HOW we solve this problem?

$\text{Dryad}_{\text{dec}}$: A decidable logic that handles shape, data and measurements in tandem

$\text{Dryad}_{\text{dec}}$

A decidable fragment of Dryad logic

- Capable of describing various tree data-structures, e.g., AVL trees, red-black trees…
- Allow user-provided recursive predicates/functions defined on trees

Examples for binary trees rooted by x:

- **Non-Measure Function** $non\_measure\_f^*$:
  monotonically increasing/decreasing

$$max\_key(x) \stackrel{\text{def}}{=} \text{ite}\left(isNil(x), -\infty, \max\begin{pmatrix} max\_key(x.left), \\ max\_key(x.right), \\ x.key \end{pmatrix}\right)$$

- **General Predicate** gp^*:
  involve $non\_measure\_f^*$ + gp^*

$$sorted(x) \stackrel{\text{def}}{=} \text{ite}\left(isNil(x), true, \bigwedge\begin{pmatrix} sorted(x.left), \\ sorted(x.right), \\ \max(x.left) \le x.key \le \min(x.right) \end{pmatrix}\right)$$

- **Measure Function** $measure\_f^*$:
  allow **only differences** between the same $measure\_f^*$

$$black\_heightt(x) \stackrel{\text{def}}{=} \text{ite}\left(isNil(x), 0, \max\begin{pmatrix} black\_height(x.left) \\ black\_height(x.right) \end{pmatrix} + ite(x.isblack, 1, 0)\right)$$

- **Measure-Related Predicate** mp^*:
  involve $measure\_f^*$ + $non\_measure\_f^*$

$$avl(x) \stackrel{\text{def}}{=} \text{ite}\left(isNil(x), true, \bigwedge\begin{pmatrix} avl(x.left), \\ avl(x.right), \\ |height(x.left) - height(x.right) \le 1| \end{pmatrix}\right)$$

Decidability

Crux of decidability Proof: Small Model Property

A $\text{Dryad}_{\text{dec}}$ formula is satisfiable only if it is satisfied by a model of bounded size.

Obtain Small Model Property:

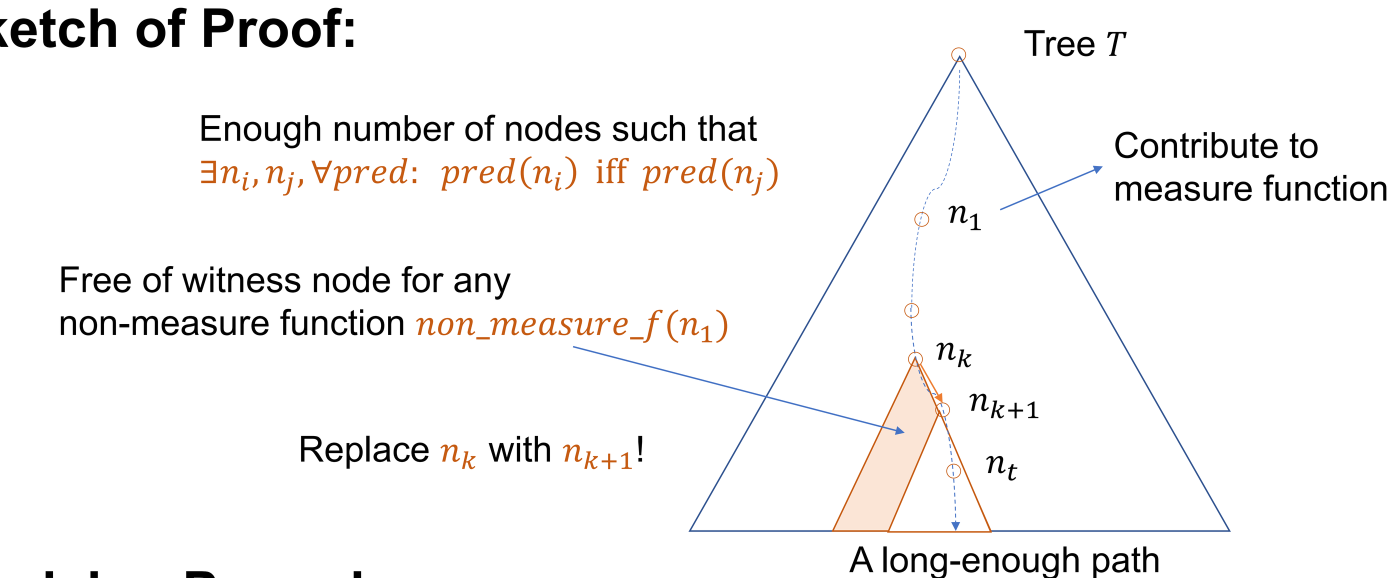
- Preserve witness node for non-measure functions

  Example: $max\_key(x)$: node store maximal key in tree
- Preserve measurement differences by tailoring both two trees

$$height(x_1) - height(x_2)$$

why only differences between measure functions are allowed

Sketch of Proof:



Decision Procedure:

- Analytically compute height/size bound
- Search every possible tree within bound
- Reduce to linear arithmetic formula
- The satisfiability is in *NEXPTIME*

Application: Checking Fusibility of Tree Traversals

Requirements for fused traversals:

- Perform identical operations
- Do not violate dependencies

Encode fusibility into predicates:

- dp: dependency of unfused traversals
- $schd$: schedule of fused traversal
- Check $schd(x) \wedge \neg dp(x)$
  for every possible schedule

```
A(n)
  if (n == nil) return
  B(n.l); B(n.r)
  int ls = n.l ? 0 : n.l.s
  int rs = n.r ? 0 : n.r.s
  n.v = ls + rs + 1

B(n)
  if (n == nil) return
  A(n.l); A(n.r)
  int lv = n.l ? 0 : n.l.v
  int rv = n.r ? 0 : n.r.v
  n.s = lv + rv + 1

main(n)
  A(n); B(n)
```

Fuse

```
fused(n)
  if (n == nil) return
  fused(n.l); fused(n.r)
  int ls = n.l ? 0 : n.l.s
  int rs = n.r ? 0 : n.r.s
  n.v = ls + rs + 1
  int lv = n.l ? 0 : n.l.v
  int rv = n.r ? 0 : n.r.v
  n.s = lv + rv + 1
```

Experiment:

- Mutual recursion
- Post-order before pre-order

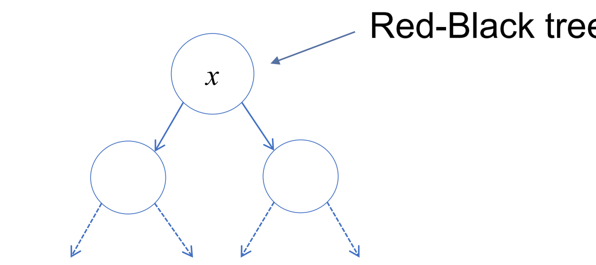
Category	# Schedule	Time/Schedule(s)	Fusible?
Mutual-rec	4	<3	Yes
	20	6-12	No
Post-pre	2	<1	Yes
	22	<1	No

Application: Verifying Tree-Manipulating Programs

VC Generation:

- Unfold recursive definitions across footprint
- Unfold remaining recursive definitions for bounded times
- Example: rbt_insert

```
Node rbt_insert(Node x, int k) {
 if (x == NULL) {…} else if (k < x.key) {
  Node p = rbt_insert(x.left, k);
  if (p.color == 0) {
    x.left = p; return x;
  } else {
   Node xr = x.right;
   if (xr.color != 0) {
    x.left = p;
    … //recolor
    return x;
   } else {
    Node pl = p.left;
    Node pr = p.right;
    if (pr.color != 0) {
     … //left right rotate and recolor
     return pr;
    } else if (pl.color != 0) {
     … //right rotate and recolor
     return p;
    } else {
     x.left = p; return x;
    }
   }
  }
 } else {…}
}
```



$(x \neq nil \wedge rbt(x) \wedge k < x.key) \rightarrow (rbt(x.left))$

Unfold $rbt(x)$

$$\begin{pmatrix} k < x.key \wedge rbt(x.left) \wedge rbt(x.right) \\ black\_height(x.left) = black\_height(x.right) \\ (x.color = 1 \rightarrow isBlack(x.left) \wedge isBlack(x.right)) \end{pmatrix} \rightarrow (rbt(x.left))$$

Let $x.left, x.right$ be two disjoint trees of bounded size

Solved by theorem prover（Z3）

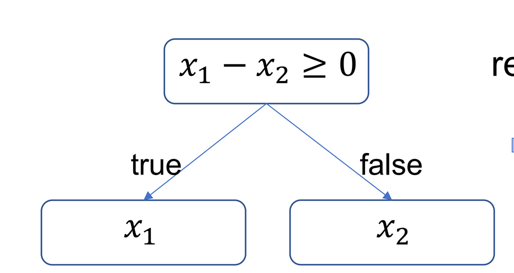
Experiment: 4 basic routines

Category	#VC	Time/VC(s)	Category	#VC	Time/VC(s)
AVL-insert(balancedness)	11	<1	AVL-insert(sortedness)	7	<1
RBT-insert(balancedness)	13	<1		2	<2
Treap-insert	3	<1	RBT-insert(sortedness)	11	<1
	3	<100		2	<2
	1	153.59	BST-insert	5	<1

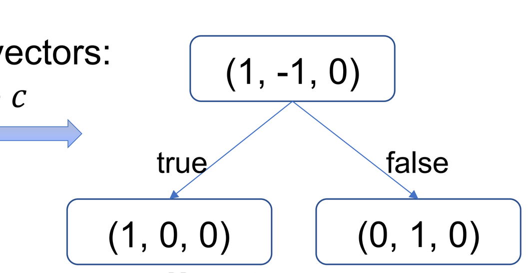
Application: Synthesizing CLIA Functions

- CEGIS framework:
  $\exists f.\forall \vec{x}.spec_f(\vec{x}) \longrightarrow \exists f.\wedge_{e \in G} spec_f(e)$
- Decision tree representation

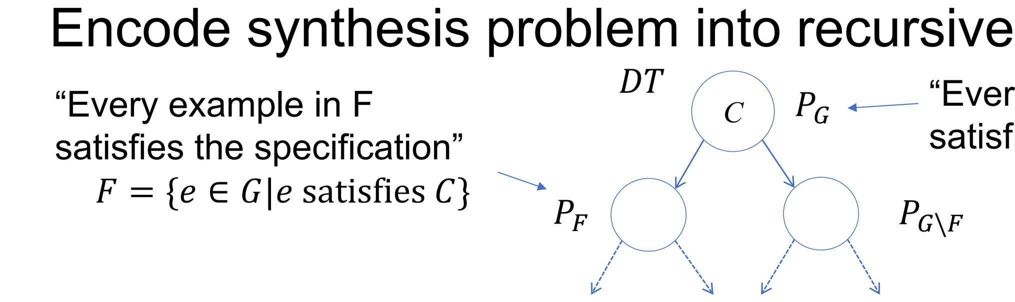
  Example: specification: $max2(x_1, x_2) \stackrel{\text{def}}{=} if\ x_1 \ge x_2\ then\ x_1\ else\ x_2$ counterexample set: G



represented as coefficient vectors:
$(a, b, c) \stackrel{\text{def}}{=} ax_1 + bx_2 + c$



- Encode synthesis problem into recursive predicates



Task: Find a decision tree DT which root satisfy predicate P_G

Experiment: $max15$
obtained and solved
15 formulas