Learning to Decipher the Heap for Program Verification

† Microsoft

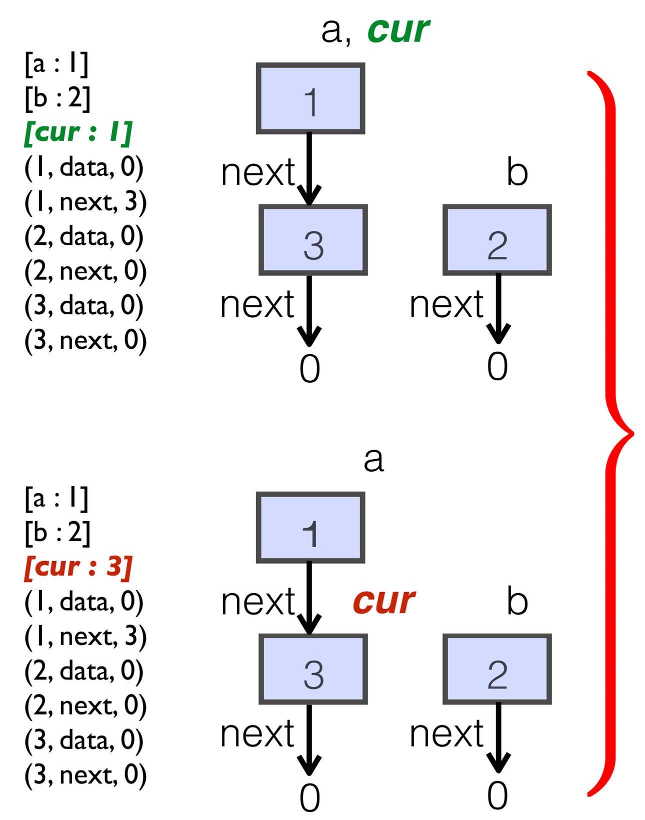
Marc Brockschmidt†, Yuxin Chen‡, Byron Cook¶, Pushmeet Kohli†, Daniel Tarlow†

‡ ETH zürich ¶ UCL

Motivating Application

```
procedure concat(a: Node, b: Node)
  returns (res: Node)
  requires lseg(a, null) &*& lseg(b,null)
  ensures lseg(res, null)
{
  if (a == null) {
    return b;
  } else {
    var cur := a;
    while (cur.next != null) {
      cur := cur.next;
    }
    cur.next := b;
    return a;
  }
}
```

Execute



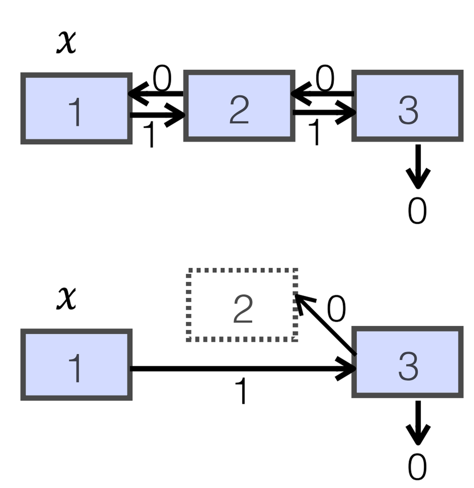
Memory safe?

Predict

GRASShopper ← Instrument program ← ls(*cur*, null) * ls(a, cur) * ls(b, null)

```
// Remove an element from a doubly-linked list
void RemoveItemFromDLL(DLL* x, int itm)
{
    // if Item is at 1st node
    if (x->item == itm) { /* ... */ }
    node *current, *del = x;
    while(del->next != NULL && del->item != itm) {
        del = del->next;
    }
    // if target not found || target at the end
    if (del->next == NULL) { /* ... */ }
    current = del->previous;
    current->next = del->next;
    del->next->previous = current;
    delete(del);
}
```

An example code snippet which is memory unsafe



Problem Description

To find **formal descriptions** of the **data structures** that are instantiated, which are used as input to a proof procedure that verifies the program.

Input Representation

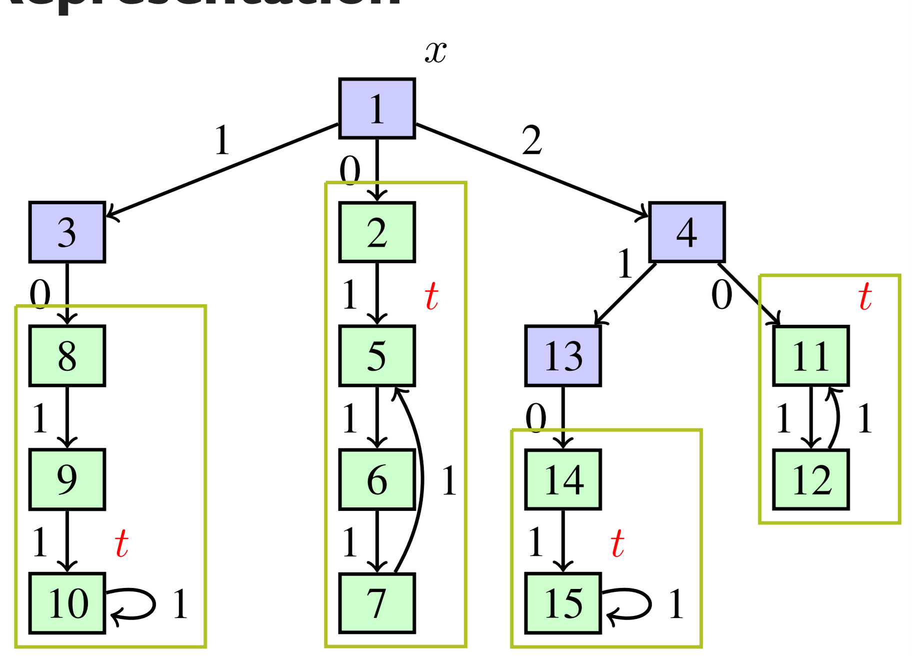
Heap graphs

Nodes:

address in memory at which a sequence of pointers are stored

Edges:

the values of these pointers



Binary tree of "panhandle lists"

Output Representation

logical description of the instantiated data structures, (i.e., formal description of a set of allowed heaps)

Inductive predicates:

e.g., $ls(x, y, \varphi)$, $tree(x, \varphi)$

Given a predicate, we allow *nested subformulas*

Binary tree of "panhandle lists"

$$\psi = \mathsf{tree}(x, \lambda i_1, i_2, i_3, i_4 \rightarrow \exists t.\mathsf{ls}(i_2, t, \lambda i_5, i_6, i_7, i_8 \rightarrow \top) * \mathsf{ls}(t, t, \lambda i_9, i_{10}, i_{11}, i_{12} \rightarrow \top)).$$

Problem Formalization

To automatically predict a separation logic formula from a given heap H

Separation logic formulas

Formula $\rightarrow$ Heaplets | $\exists$Var.Heaplets | $\exists$Var, Var.Heaplets | ...

Heaplets $\rightarrow \top$ | Heaplet $*$ Heaplets

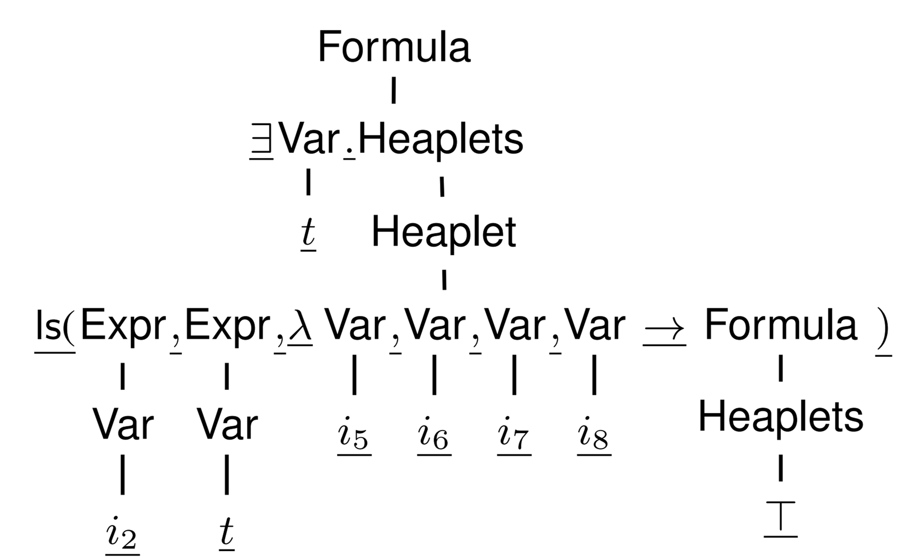
Heaplet $\rightarrow$ ls(Expr, Expr, λVar, Var, Var, Var $\rightarrow$ Formula)

| tree(Expr, λVar, Var, Var, Var $\rightarrow$ Formula)

Expr $\rightarrow$ NULL | Var

$\exists t.\mathsf{ls}(i_2, t, \lambda i_5, i_6, i_7, i_8 \rightarrow \top)$

Parse tree



parse tree

$$\mathcal{T} = (\mathcal{A}, g(\cdot), ch(\cdot))$$

nodes $\mathcal{A} = \{1, \ldots, A\}$

nodes $\mapsto$ terminal / non-terminal in separation logic grammar $g : \mathcal{A} \mapsto \mathcal{S}$

nodes $\mapsto$ children tuple $ch : \mathcal{A} \mapsto \mathcal{A}^*$

Partial tree $\mathcal{T}_{<a}$: *Parse tree restricted to nodes {1, ... , a}*

Problem Statement

Sequentially predict the next node, conditional upon everything that has been predicted so far:

$$P(\mathcal{T}) = \prod_{a:g(a)\in\mathcal{N}} p(ch(a) \mid H, \mathcal{T} < a).$$

Maximal likelihood training

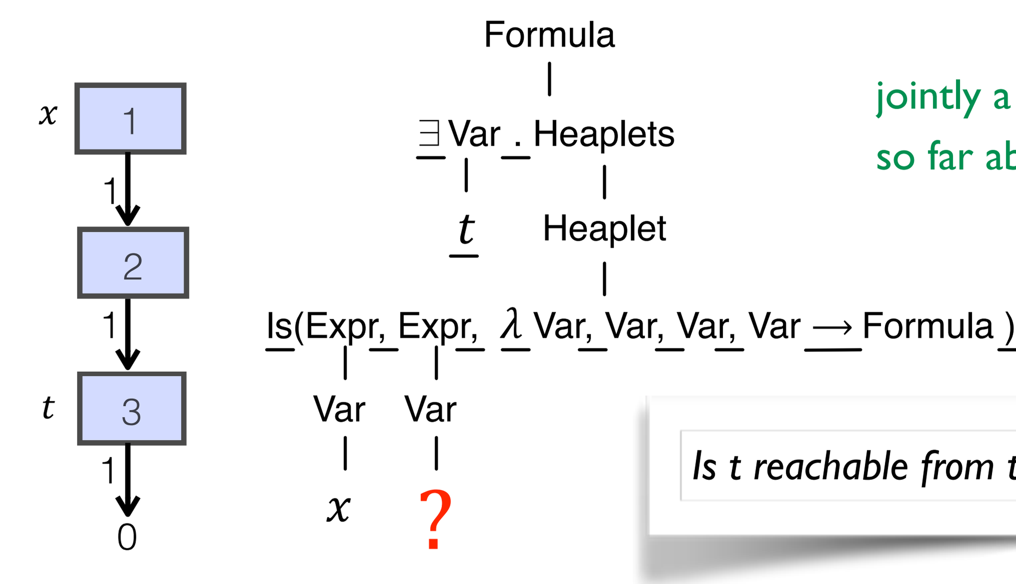
⇕

Independent classification

Nonterminal ↦ Children tuple

Features

Heap features	Partial tree features	Joint features
(int list) **1-gram** (frequency of different pointer types)	*(int)* **depth** depth of the current nonterminal node	*(bool)* **if variable "y" is reachable**
(int list) **2-gram** "adjacent" pointers	*(int)* **frequency**	*(bool)* **if variable "y" is accounted for**
(bool) **graphic is cyclic**	*(string)* **predecessor token**	*(string)* **relation to defined predicates**



jointly a function of decisions we have made so far about the formula and the heap graph

Is t reachable from the current heap graph? — Yes.

Experimental Results

Prediction model: Random Forests, **Multiclass Logistic Regression**, Multiclass Neural Network

Predicates used: lists, cyclic lists, panhandle lists and trees.

Nesting level	Number of free variables	Number of Formulas
1	1	127
1	2	33254
0	4	3515

We sample **1757** formulas from the above table (last row)

500 heap graphs are generated per formula.

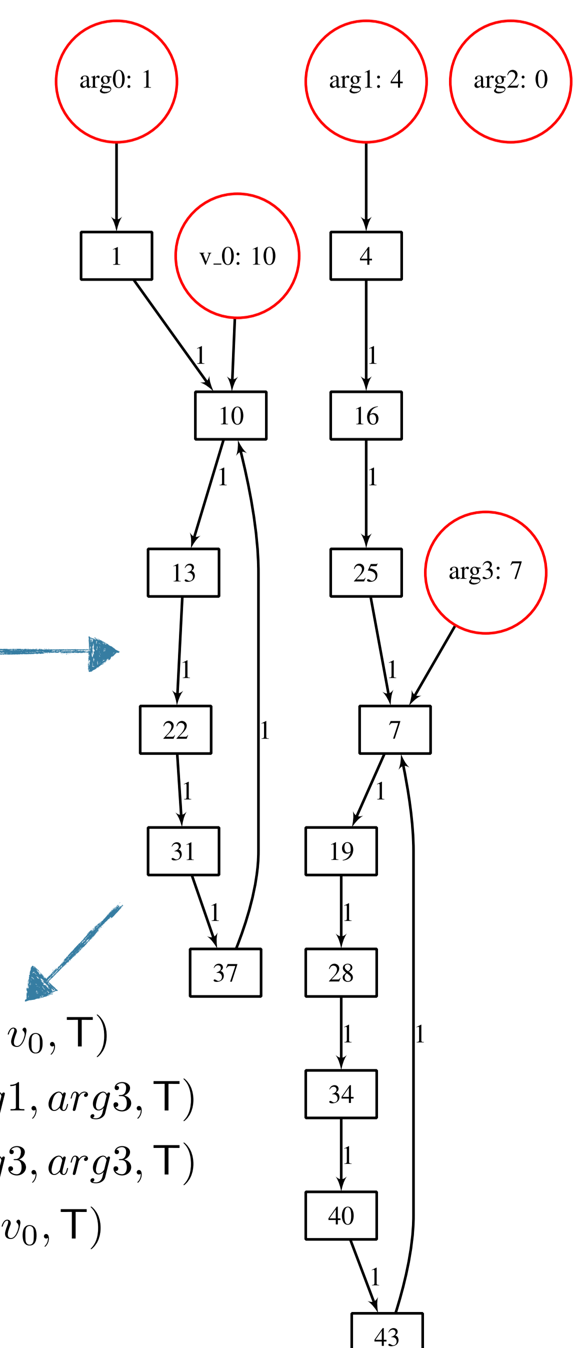
878,500 formula/heap graph combinations.

Example input graph from dataset which is corrected predicted.

(nested level 0, number of free variables 4)

Training / validation / testing sets: 6:2:2

Method	Top 1 Acc.	Top 10 Acc.
Concatenation features	0.05%	0.07%
Joint features	**91.5%**	**91.6%**



$$\exists v_0.\mathsf{ls}(arg0, v_0, \top) * \mathsf{ls}(arg1, arg3, \top) * \mathsf{ls}(arg3, arg3, \top) * \mathsf{ls}(v_0, v_0, \top)$$

One can then use static program verification tools to determine whether the description is accurate and whether the program satisfies is memory safe.