A Survey on Heap Analysis

PhD Qualifying Examination

Chengpeng Wang

Supervisor:

Dr. Charles Zhang

Committee Members:

Prof.Shing-Chi Cheung, Prof.Ke Yi, and Dr.Wei Wang

Oct 5th, 2020

Outline

- Background
- Problem
- Existing works
- Conclusion and future work

Outline

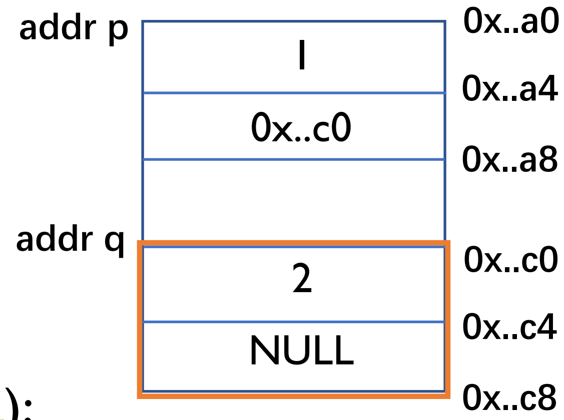
- Background
- Problem
- Existing works
- Conclusion and future work

Program Heap

- Dynamic allocation is managed by programmers
- Pointers form complex connectivity relation

program

run



heap

use after free

```
Node* p = new Node(1, NULL);
p->next = new Node(2, NULL);
q = p->next;
free(q);
cout << p->next->data;
```

Heap Memory Bug

- Heap memory bugs are common and critical
- 4079 CVE entries in total
    - Use-after-free 
    - Memory leak 

    - Sensitive information exposure 
    - etc.

Static Analysis

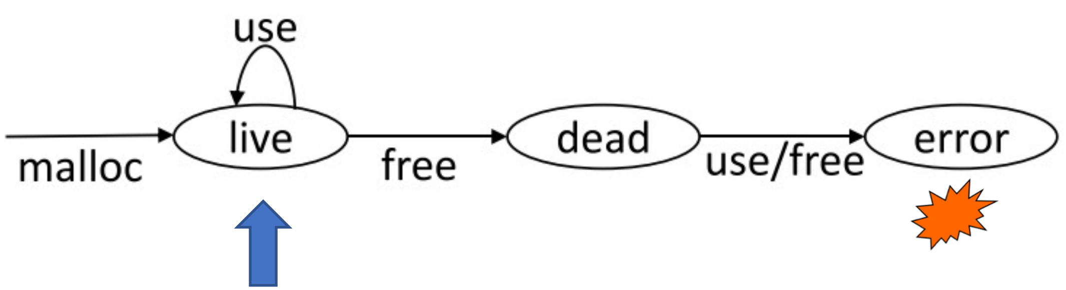
- Analyze source code without actual execution
  - Approximate runtime states by abstract states
  - Transform abstract states based on statement effect, i.e., semantics of operations
  - Cover abnormal runtime states

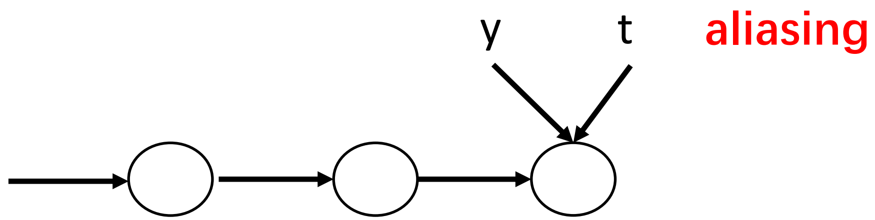
- Typical static analysis clients for heap bug detection [Nor, SAS 00][Fink, ISSTA 06][Xiao, ISSTA 14][Arzt, PLDI 14]
  - Rely on heap properties

Heap Memory Bug Detection

- UAF detection [Nor, SAS 00]
  - Heap property: aliasing

```
List* y = createList();
List* t = NULL;
while (…) {
    t = y->n;
    y = t;
}
free(y);
int d = t->data;
```



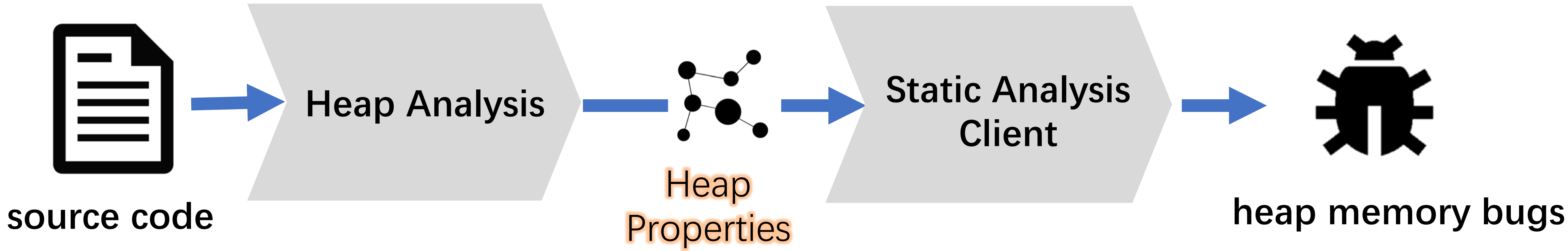


Outline

- Background
- Problem
- Existing works
- Conclusion and future work

Problem

- Problems in heap analysis can be concluded into one question
  - How to infer heap properties [Kanvar, CSUR 16]



Heap Property

- Formally, heap is a set of objects and a connectivity relation on them [Barr, ISSTA 13]
- Connectivity relation induces a variety of properties, including
    - Aliasing
    - Reachability
    - Ownership

Complexity of Heap

- Dynamic allocation
  - Unable to determine heap allocation statically
- Various data structures and operations
  - Infeasible to model semantics of operations

```
int N = input();
Node* prev = malloc();
for (int k = 0; k <= N; k++) {
    Node* f = malloc();
    f->next = prev;
    …
}
```

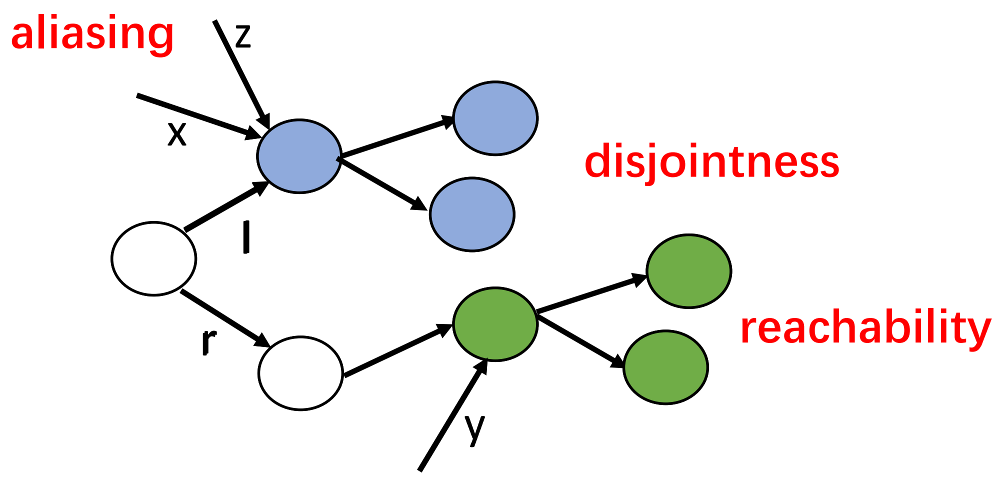
```
class A {
    Node* dataList;
    int size = 0;

   void f(Node* d) {…}
   Node* g() {…}
}
```

Two Kinds of Structural Heap

- Structural heap with pointers
- Structural heap in containers

Structural Heap with Pointers

- Connecting objects by pointers
  - Composed of same type of objects with pointer-valued fields
  - Manipulated by pointer operations
- Property
  - Aliasing
  - Reachability
  - Disjointness

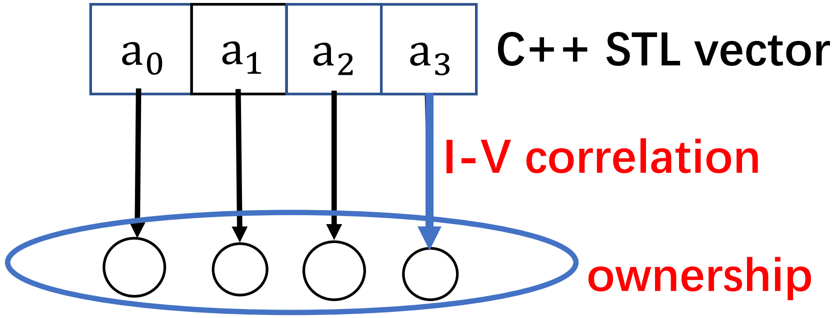


x = z
y = y->left

Structural Heap in Containers

- Storing objects in containers
  - Composed of objects with index
  - Manipulated by standard library interfaces

- Property
  - Ownership
  - Index-value correlation



v[0]
v.push_back()

Two Kinds of Structural Heap

- Structural heap with pointers
- Structural heap in containers

	Heap Property	Bug Type
Pointer	aliasing	use-after-free
	reachability	memory leak
	disjointness	data race
Container	ownership	sensitive information exposure
	I-V correlation	

Heap Analysis

- How to infer heap properties in structural heaps
  - Dynamic allocation
  - Various data structures and operations

- Two technical questions
  - Q1: How to abstract unbounded structural heap
  - Q2: How to model semantics of operations

Outline

- Background
- Problem
- Existing works
    - Structural heap with pointers
    - Structural heap in containers
- Conclusion and future work

Structural Heap with Pointers

- Two technical questions
  - Q1: How to abstract unbounded structural heap
  - Q2: How to model semantics of operations

- How to infer heap property in structural heap with pointers
  - Q1: How to abstract objects connected by pointers
  - Q2: How to encode semantics of pointer operations

Solutions

- Shape analysis based on abstract interpretation
    - Encode and update finite-sized abstract heap by logical formulas

- Typical works
    - TVLA: Three-valued logic based shape analysis [Reps, TOPLAS 02][Jeannet, TOPLAS 10]
    - Infer: Separation logic based shape analysis [Distefano, TACAS 06][Cristiano, POPL 09]

Preliminary: Abstract Interpretation

- Regard actual execution as concrete state transition system
  - Associate program locations with concrete states
  - Associate operation with concrete transformers
- Construct customized abstract state transition system
  - Iterate abstract transformers until a fixed point reaches

```
int x = 1;
while (input) {
    x = x + 1;
}
```

x: +

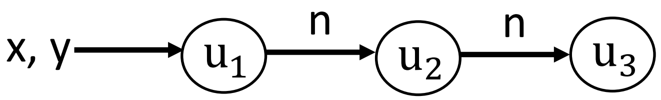
x: +

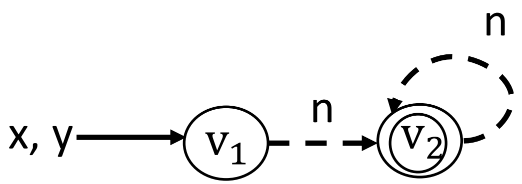
$x \in$ INT

$x \in \{+, -, 0\}$

TVLA: Three-Valued Logic Analyzer

- Q1: Heap abstraction in TVLA
  - Encode and abstract heap by predicates in three-valued logic





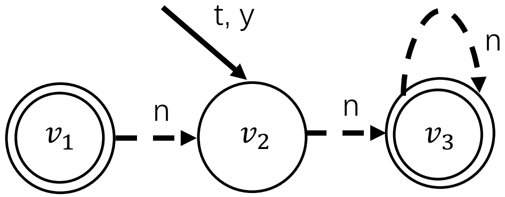
$x(u_2) = 0$ $\quad x(u_3) = 0$ $\quad n(v_1, v_2) = 1/2$

Predicate	Meaning
$\mathrm{x}(u)$	x points to u
$n(u_1, u_2)$	n field of u_1 points to u_2

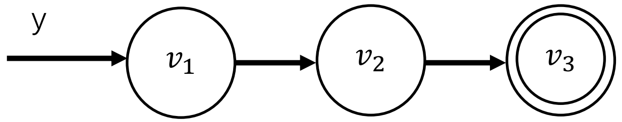
TVLA: Three-Valued Logic Analyzer

- Q2: Semantic encoding in TVLA
  - Update predicates by predicate-update formulas for fixed points

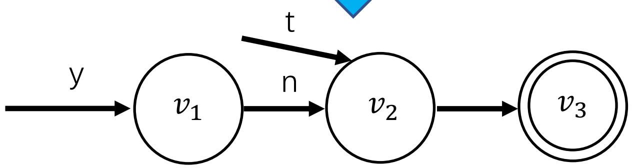
```
List* y = createList();
List* t = NULL;
while (…) {
    t = y->n;
    y = t;
}
```



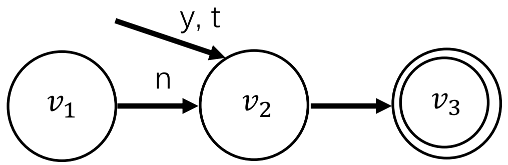
$\forall v\colon\ y(v) = \mathrm{t}(\mathrm{v})$



$t'(\mathrm{u}) = \exists v\colon\ y(v) \land \mathrm{n}(\mathrm{v}, \mathrm{u})$



$\mathrm{y}'(u) = \mathrm{t}(\mathrm{u})$



Strength

- Expressive and general
  - Depicting fine-grained connectivity relation by low-level predicates
  - Customizing logical formulas for various heap properties

Heap property	Logical formula
aliasing	$\forall v\colon\ y(v) = \mathrm{t(v)}$
reachability	$x^{+}(u) = x(u) \vee (\exists v\colon\ x^{+}(v) \wedge n(v,u))$
disjointness	$\neg(\exists v\colon\ x^{+}(v) \wedge y^{+}(v))$

Weakness

- Inefficient
  - Updating a host of predicates by solvers
    - Heap size: $3^{|A|}$ [Reps, TOPLAS 02]
    - Presence of loops
- Dependent to expertise
  - Defining predicates for heap abstraction

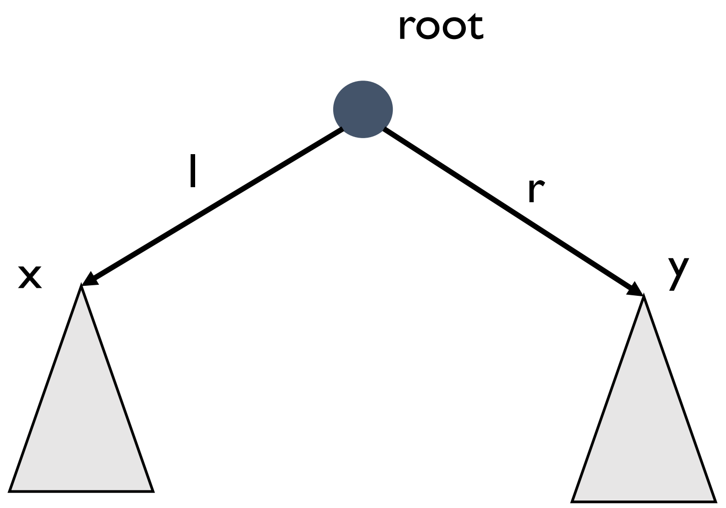
$|A|$: #predicates for abstraction

Test Program in TVLA
Single-linked list create
Single-linked list merge
Single-linked list reverse

Infer: Separation Logic Analyzer

- Q1: Heap abstraction in Infer
  - Partition heap into disjoint atomic blocks

```
Node* root = createTree();
if (root != 0) {
    Node *x = root->l;
    Node *y = root->r;
    free(root);
}
```

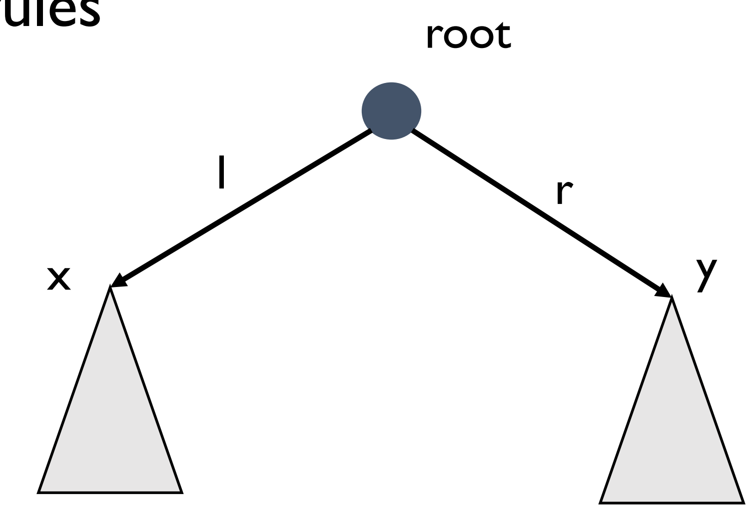


$$\mathrm{root} \rightarrow [\mathrm{l}: \mathrm{x}, \mathrm{r}: y] * \mathrm{tree}(x) * \mathrm{tree}(y)$$

Infer: Separation Logic Analyzer

- Q2: Semantic encoding in Infer
  - Update the relevant atomic blocks by SL rules

```
Node* root = createTree();
if (root != 0) {
    Node *x = root->l;
    Node *y = root->r;
    free(root);
}
```



$\text{root} \rightarrow [\text{l}: \text{x}, \text{r}: y]$

emp

$\text{root} \rightarrow [\text{l}: \text{x}, \text{r}: y] * \text{tree}(x) * \text{tree}(y)$

$\boxed{\text{tree}(x) * \text{tree}(y)}$

Strength

- Efficient
  - Local reasoning disjoint heap blocks
- Intuitive
  - Summarizing heap by high-level predicates [Distefano, TACAS 06][Rival, SAS 07]

Weakness

- Difficult to extend
  - Relying on specific rules to assure terminality

Comparison

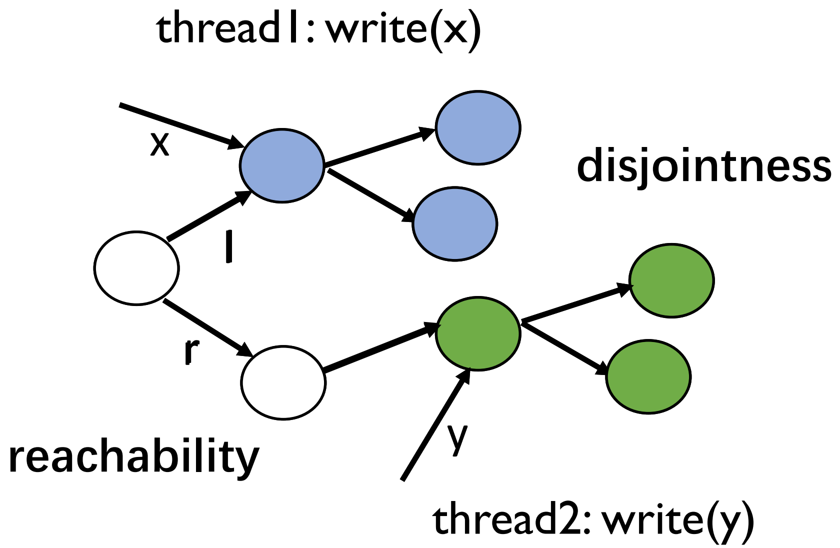
- TVLA equipped with low-level, analyzer-oriented predicates
  - Expressive and general
  - Inefficient and hard for non-expert

- Infer equipped with high-level predicates and local reasoning
  - Intuitive and efficient
  - Difficult to extend

Application

- Data race detection [W. O'Hearn, CACM 19]

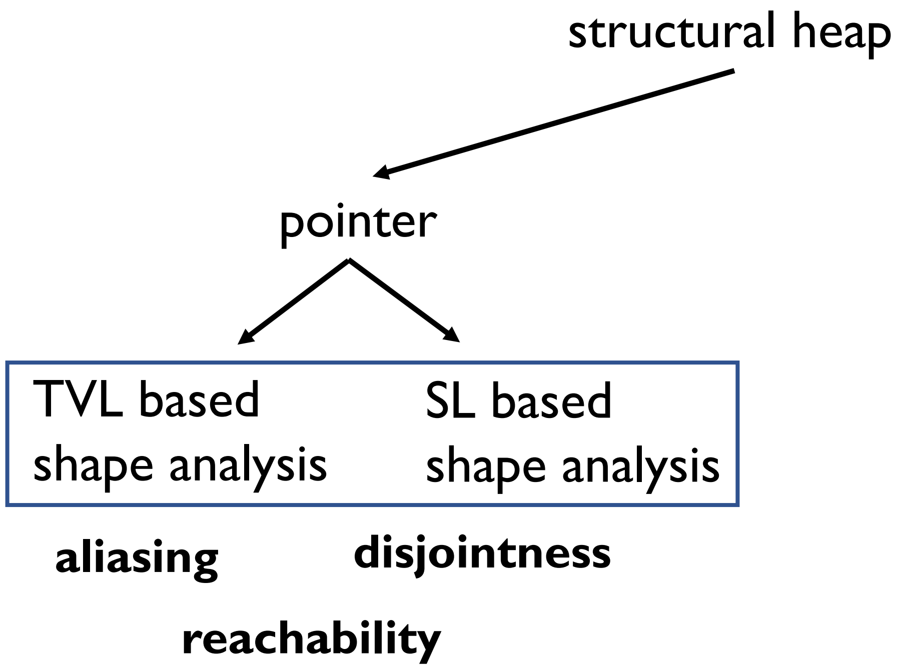
```
int main() {
   ...
  t1 = pthread_create(write, x);
  t2 = pthread_create(write, y);  //no data race
  destroy(x); destroy(y);
  return 0;  //memory leak
}
```



- Memory leak detection [Shaham, SAS 03]

Summary

- Shape analysis infers heap properties in structural heap with pointers
    - How are objects connected by pointers in the heap?



Outline

- Background
- Problem
- Existing works
    - Structural heap with pointers
    - Structural heap in containers
- Conclusion

Structural Heap in Containers

- Two technical questions
  - Q1: How to abstract unbounded structural heap
  - Q2: How to model semantics of operations

- How to infer heap property in structural heap in containers
  - Q1: How to abstract same type of objects in containers
  - Q2: How to encode semantics of standard library interfaces

Solutions

- Flow analysis
- Symbolic heap analysis

Solutions

- Flow analysis [Xu, PLDI 10] [Sridharan, OOPSLA 05]
  - Reduce property inference to CFL-reachability problem in flow graph
- Symbolic heap analysis

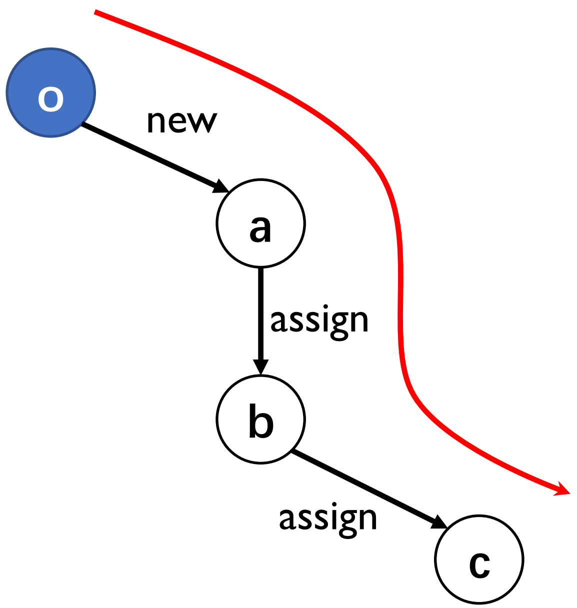
Preliminary: CFL-reachability

- Flow graph OFG = {V, E}
  - V: object set
  - E: labeled edge set
- CFL-reachability problem
  - Find paths of which label sequence is in a given context-free language

$$flowsTo \rightarrow new\ (assign)^*$$

- Infer properties by solving CFL-reachability problems in flow graph

```
A* a = new A();
A* b = a;
A* c = b;
```



Flow Analysis

- Q1: Heap abstraction in flow analysis
  - Merge objects allocated by the same statement

```
vector<A*>* v = new vector<A*>();
for (int i = 0; i < N; i++) {
    A* b = new A();
    v->push_back(b);
}
```

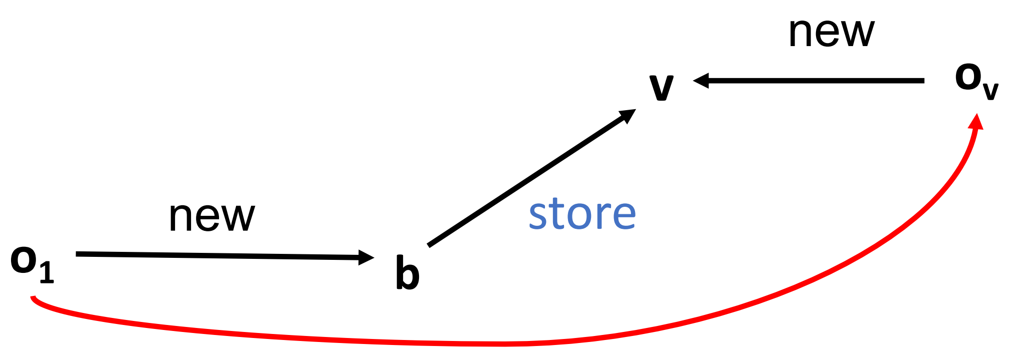
$$v \xleftarrow{\text{new}} o_v$$

$$o_1 \xrightarrow{\text{new}} b$$

Flow Analysis

- Q2: Semantic encoding in flow analysis
  - Encode library interface semantics by labeled edges

```
vector<A*>* v = new vector<A*>();
for (int i = 0; i < N; i++) {
    A* b = new A();
    v->push_back(b);
}
```



$$ownership \rightarrow flowsTo\ store\ \overline{flowsTo}$$

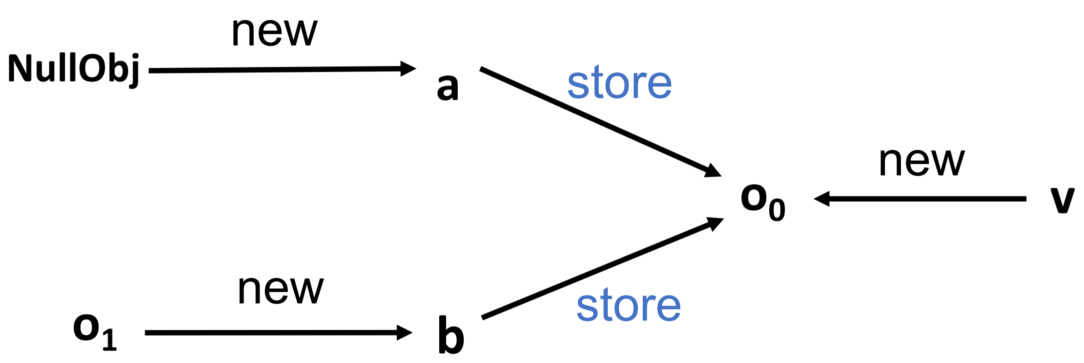
Strength

- Efficient
  - Constructing flow graph
    - $O(m)$ where m is #statements
  - Solving a given CFL-reachability problem
    - $O(n^3)$ where $n = |V|$ is a small term

Weakness

- Unable to depict inner storage
  - Insensitive to the order of statements

```
vector<A*>* v = new vector<A*>();
A* a = NULL;
v->push_back(a);
for (int i = 0; i < N; i++) {
    A* b = new A();
    v->push_back(b);
}
A* c = v[0];
```



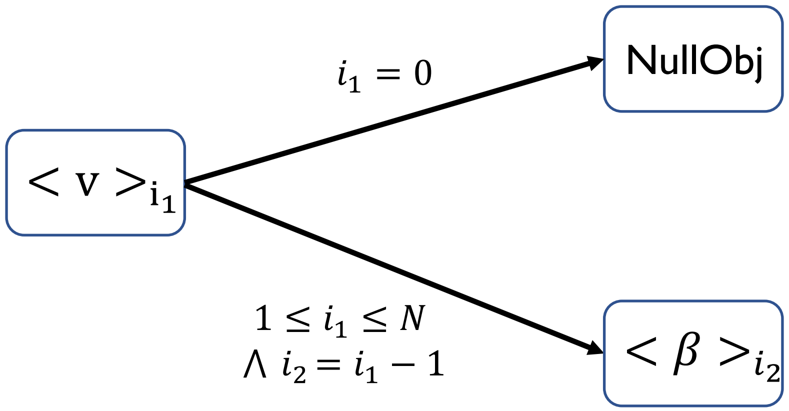
Solutions

- Flow analysis
- Symbolic heap analysis
  - Encode and update index-value correlation by constraints [Dillig, PLDI 2011] [Dillig, ESOP 2010]

Symbolic Heap Analysis

- Q1: Heap abstraction in symbolic heap analysis
  - Qualify points-to edges by constraints on index

```
vector<A*>* v = new vector<A*>();
A* a = NULL;
v->push_back(a);
for (int i = 0; i < N; i++) {
    A* b = new A();
    v->push_back(b);
}
```



Symbolic Heap Analysis

- Q2: Semantic encoding in symbolic heap analysis
  - Update edges by checking satisfiability

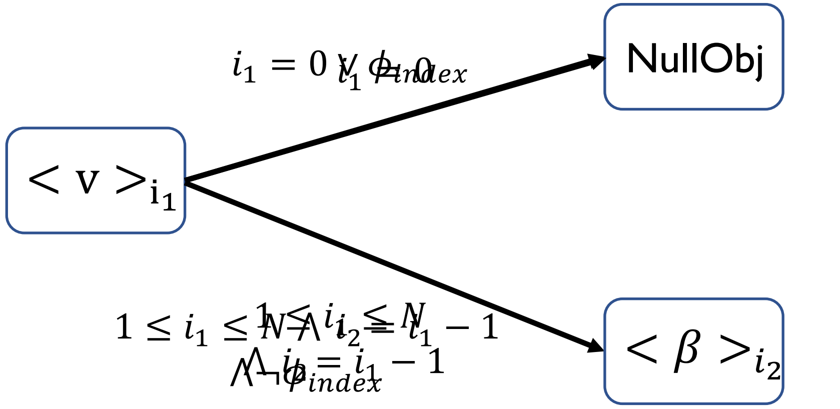
```
if (i % 2 == 0) {
    v[i] = NULL;
}

A* c = v[0];
```

$i_1 = 0 \lor \phi_{index}$

$i_1 = 0$

Solver

SAT

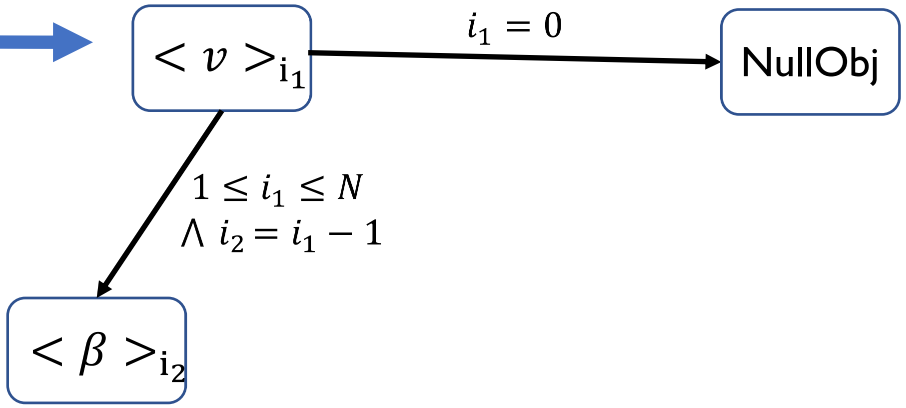


ϕ_{index}: $i_1 \% 2 = 0$

Strength

- Able to express index-value correlation
  - Reflecting the storage locations of objects
  - Establishing the connection between index and loop count

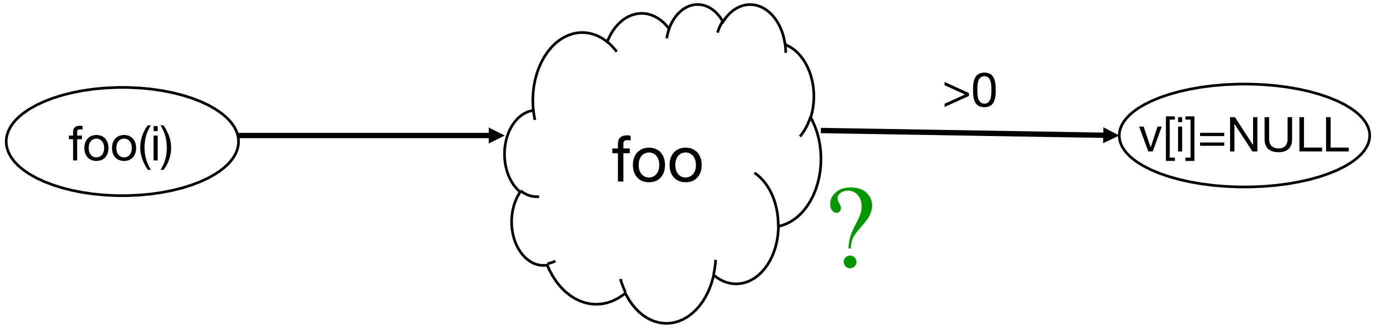
```
v->push_back(NULL);
for (int i = 0; i < N; i++) {
    A* b = new A();
    v->push_back(b);
}
```



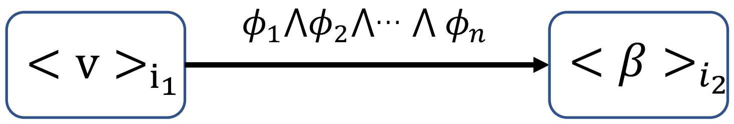
Weakness

- Imprecise encoding of complex branch conditions

```
if (foo(i) > 0) {
    v[i] = NULL;
}
```



- Conjunctive explosion in a single symbolic heap



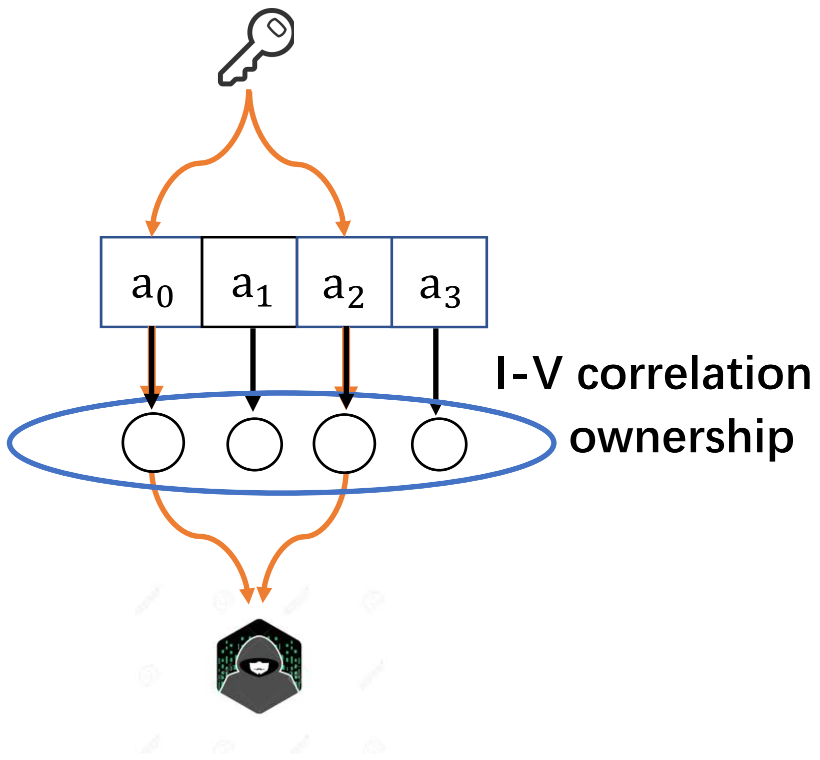
Comparison

- Flow analysis infers ownership effectively but is unable to infer index-value correlations
  - Regard container as black box
  - Insensitive to statement order
- Symbolic heap analysis reasons precise index-value correlations
  - Maintain the effects of library interfaces on indices
  - Sensitive to statement order

Application

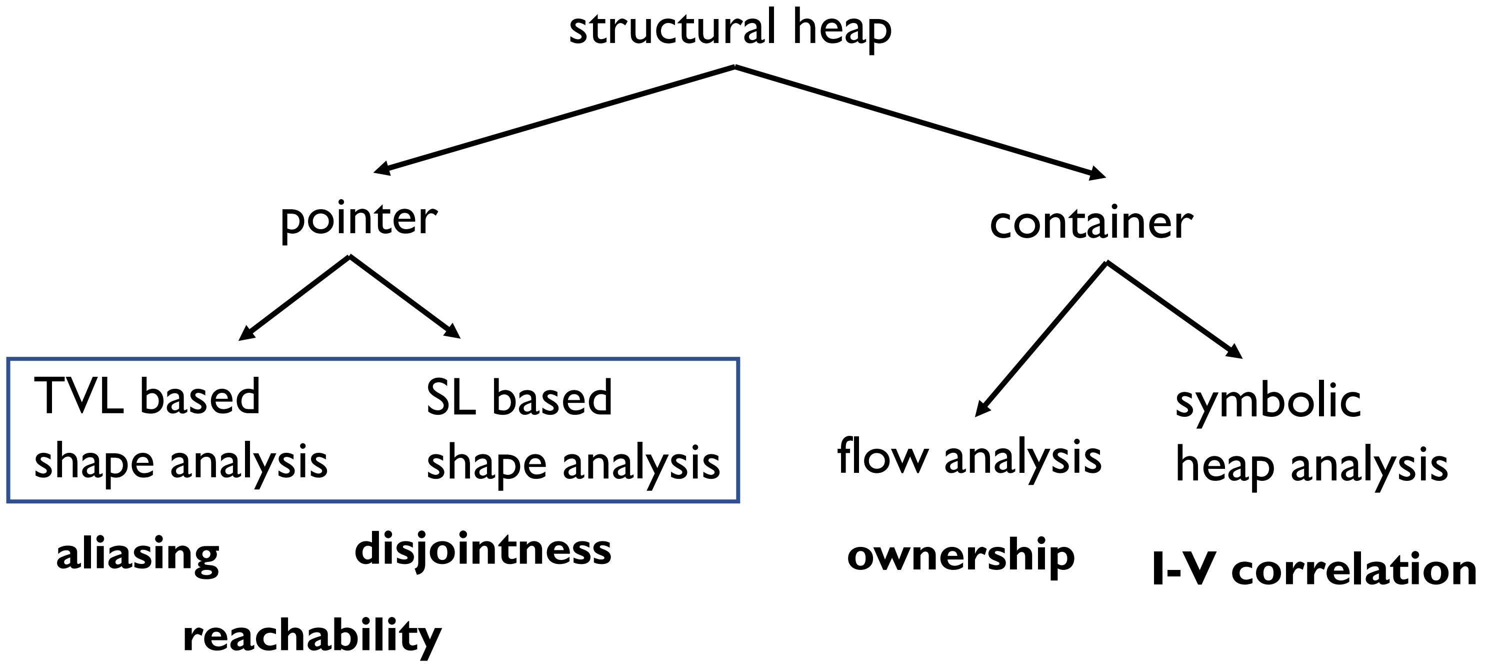
- Detecting sensitive flow exposure

```
for (int k = 0; k <= 3; k++) {
    File* f = malloc();
    if (k % 2)
        f->inputPassword();
    v.push_back(f);
}
v[0]->printAll();✗
v[1]->printAll();✓
```



Summary

- Shape analysis: How are objects linked by pointers in the heap?
- Flow analysis: How do object flow in and out of containers?
- Symbolic heap analysis: How are objects stored in containers?



Outline

- Background
- Problem
- Existing work
- Conclusion and future work

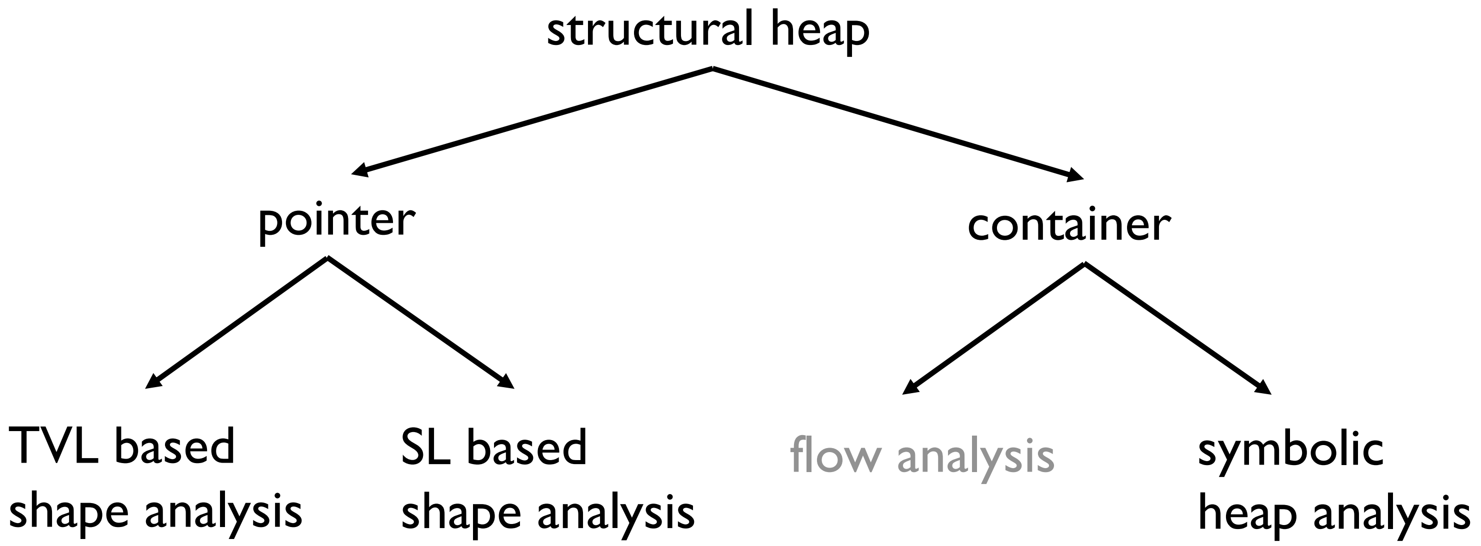
Conclusion

- The analysis of structural heap improves the precision of static analysis clients by inferring specific heap properties

Structural Heap	Heap Property	Bug Type
Pointer	aliasing	use-after-free
	reachability	memory leak
	disjointness	data race
Container	ownership	sensitive information exposure
	I-V correlation	

Conclusion

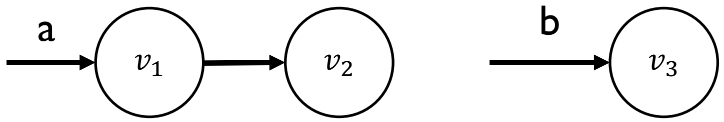
- Precise heap analysis demands constraint solvers in the analysis of real-world programs



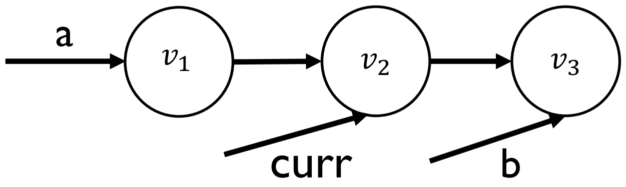
Future Work

- Multi-domain property inference
  - Layout of pointers
  - Application: library implementation verification

```
Node* curr = a;
while (curr->next != NULL) {
  curr = curr->next;
}
curr->next = b;
```



$\mathrm{sll}(a, \mathrm{NULL}) * \mathrm{sll}(b, \mathrm{NULL})$



$\mathrm{sll}(a, \mathrm{curr}) * \mathrm{sll}(b, \mathrm{NULL})$

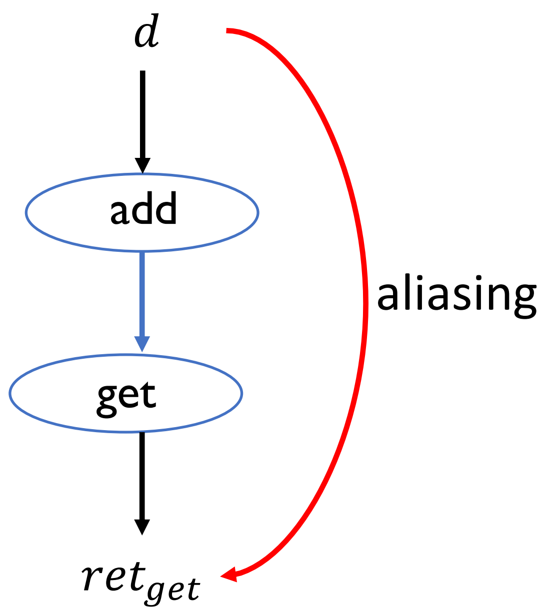
dis'(a, curr) = dis(a, NULL)-1

dis'(curr, NULL) = dis(b, NULL)+1

Future Work

- Semantic identification of interfaces
  - Load and store sequence of inner storage
  - Application: semantic fingerprint

```
class A {
    Data[] content;
    int size = 0;

    void add(Data d) { content[size]=d; size++;}
    Data get() {return content[size-1];}
}
```



Thank you for your listening!

Program Heap

- Dynamic allocation managed by programmers
  - Heap memory is allocated and deallocated in the execution
  - Lifetime lasts if the heap is not deallocated
- Complex connectivity relation formed by pointers
  - Stack pointers provide an entry to access heap
  - Objects are manipulated though pointers flexibly
- Illegal heap manipulations commonly exist

Characteristics of Heap

- Unnamed location
    - Only pointers named
    - Association between symbolic names and memory locations changes

- Unbounded size
    - Caused by loop and recursive function

- Heap escape
    - Unpredictable lifetime, dependent to the control flow

```c
int foo1() {
    int x;
    x = 5;
    int* p;
    p = &x;
    *p = 1;

    return 0;
}
```

```c
struct var {
    int addr;
    int content;
};
```

Get MAX from Syntactic Analysis

```c
int foo2() {
    var localVar[MAX];
    int cnt = 0;

    var x;
    x.addr =cnt;
    x.content = 5;
    localVar[cnt++] = x;

    var p;
    p.addr = cnt;
    p.content = x.addr;
    localVar[cnt++] = p;
    localVar[p.content].content = 1;

    return 0;
}
```

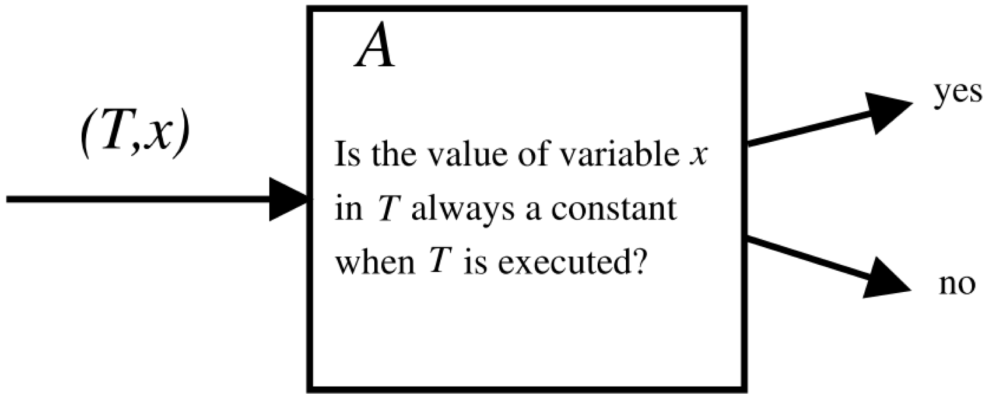
A program that is restricted only to stack and static data can be rewritten without using pointers.

Rice's Theorem

- All non-trivial questions about program behaviors are undecidable[Rice, TAMS 53]
- Approximate program states under a hypothesis specifically

Rice's Theorem

- Decidability: The language L= {(T, p(x))} is recursive language, i.e., there exists a Turing machine accepting L and rejecting $\overline{L}$



Other non-trivial properties

- Does the program terminate for all inputs?

Trivial properties

- Is the number of function parameters positive or equal to 0 ?

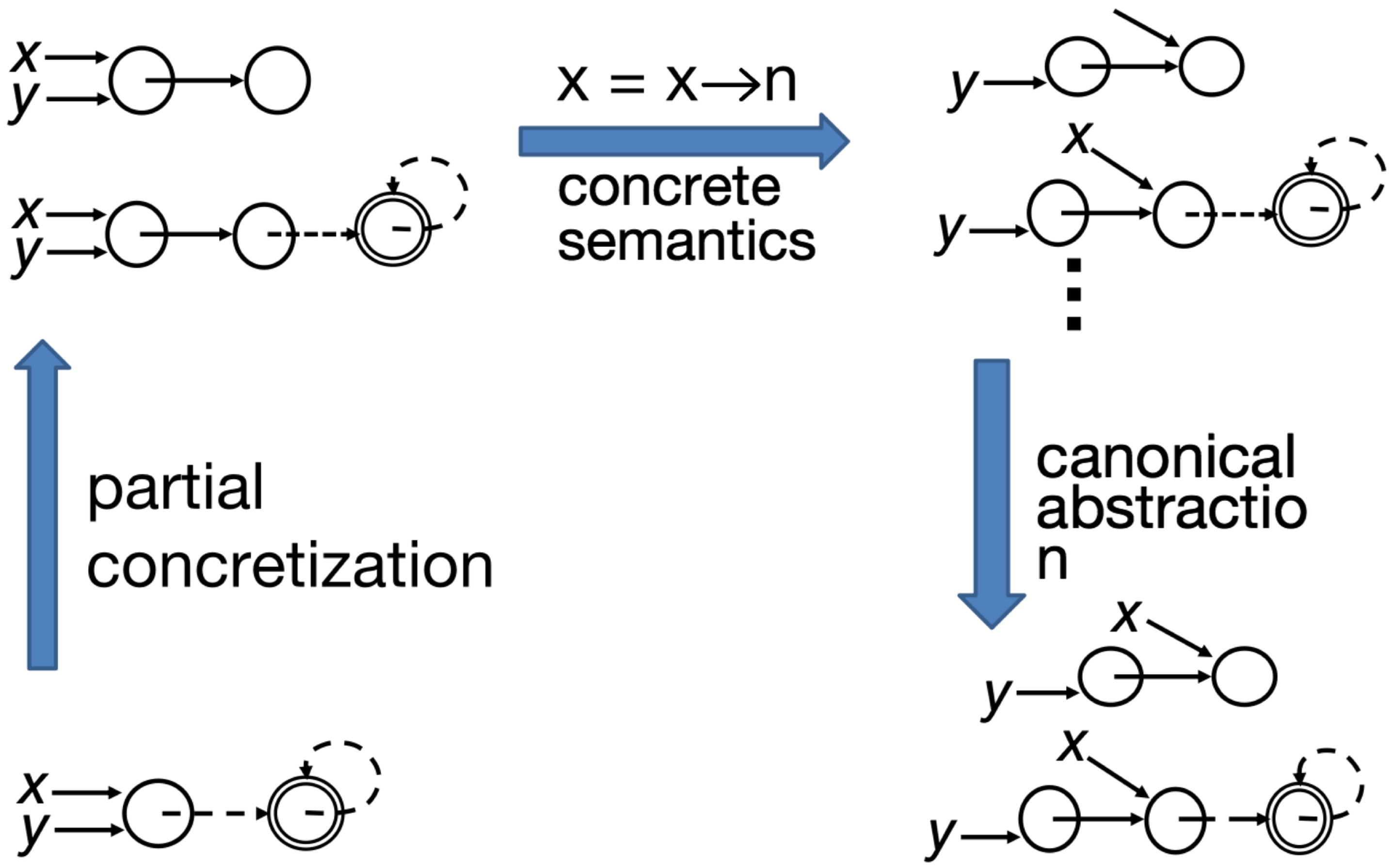
Rice's Theorem

- All non-trivial questions about program behaviors are undecidable

- Corollary: Given an arbitrary program with a heap property *P* and program location *Lc*, the problem is undecidable that whether the property *P* holds at *Lc* or not.

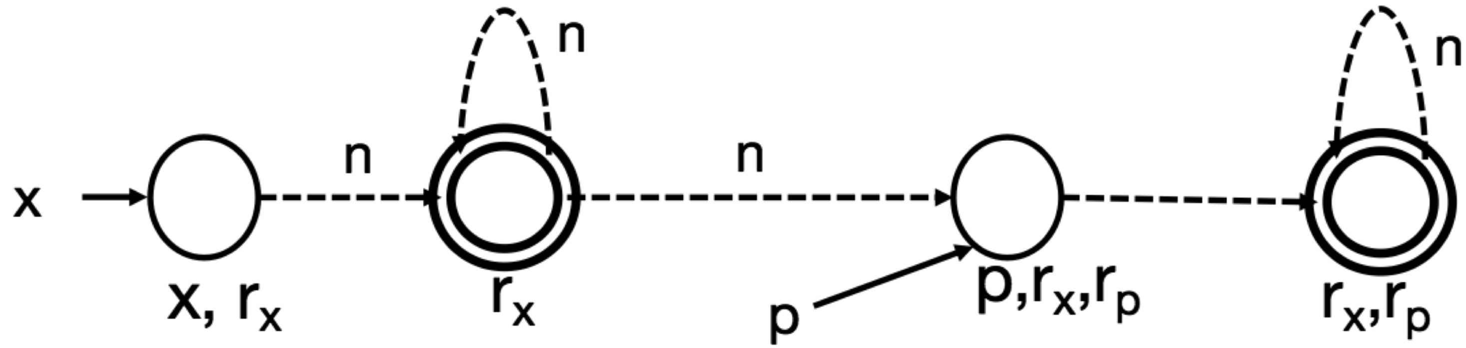
- Restrict heap analysis to two forms of structural heap
  - Structural heap with pointers
  - Structural heap in containers

TVLA: Partial Concretization



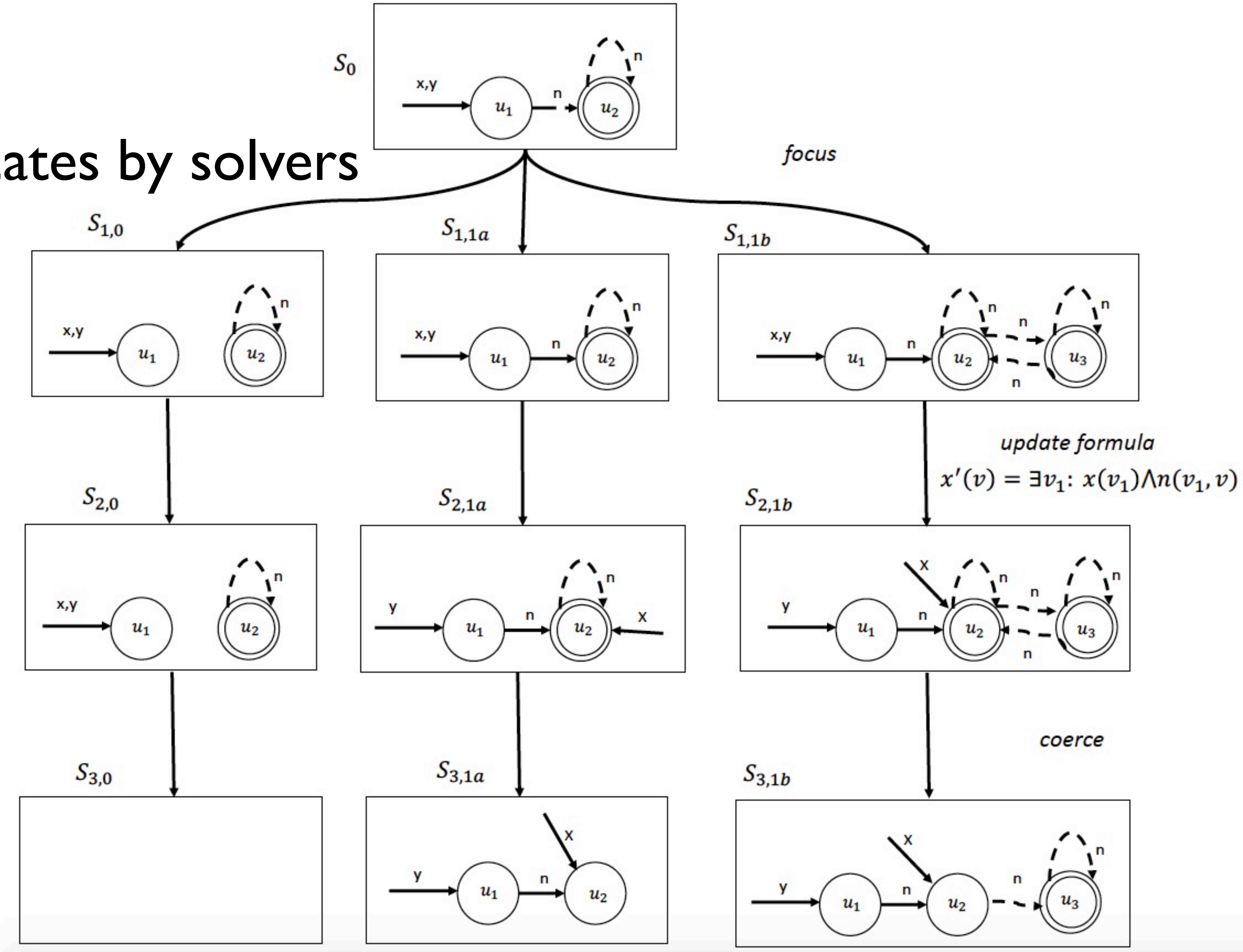
TVLA: Abstract Heap Size

- Upper bound: $3^{|A|}$
  - Three-valued logic
  - Merge objects if predicates evaluate to the same value on them



TVLA: Weakness

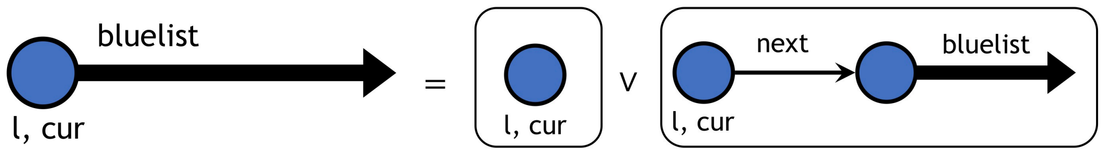
- Updating a host of predicates by solvers
  - Heap size: $3^{|A|}$
  - Presence of loops
- Example
  - x=x->n



SL based Shape analysis

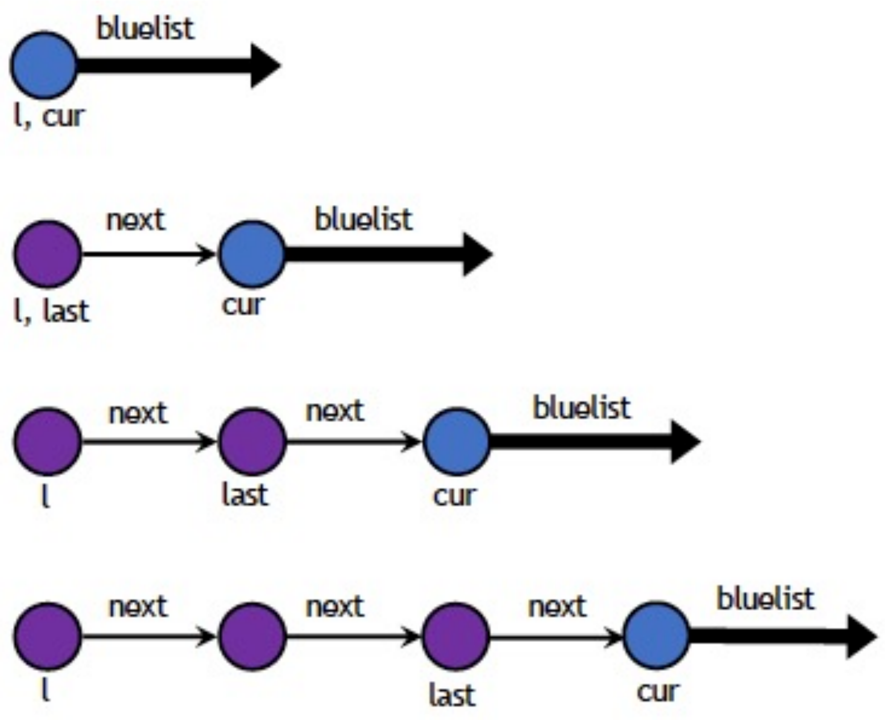
- Abstract heap model in Separation Logic [Rival, SAS 2007]:
- Pattern based abstraction
  - data structures type
  - checking function

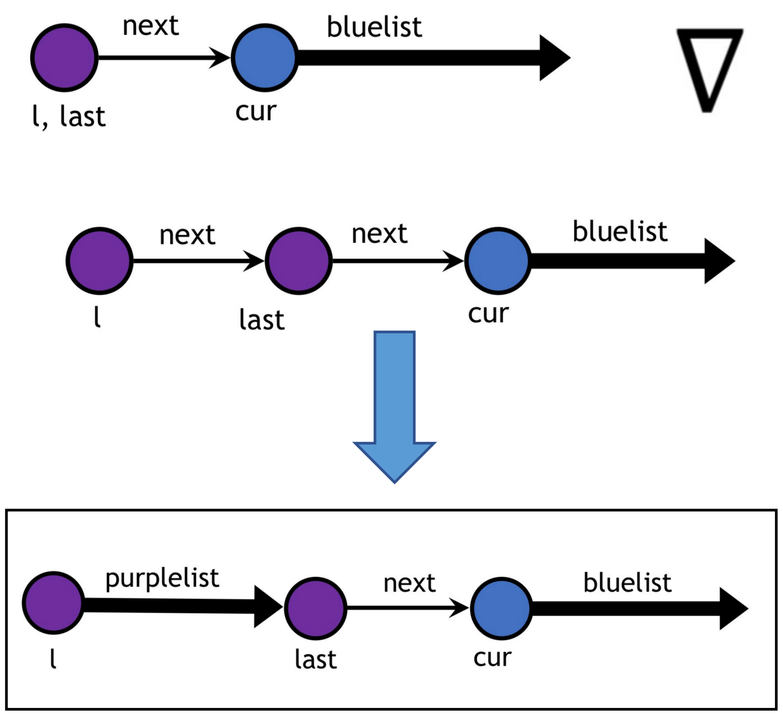
```
bool bluelist(List* l) {
    if (l == null) return true;
    else return (l−>color==blue)
           && bluelist(l−>next );
}
```



Infer: Weakness

- Difficult to extend
    - Relying on specific operator(widening) to assure terminality



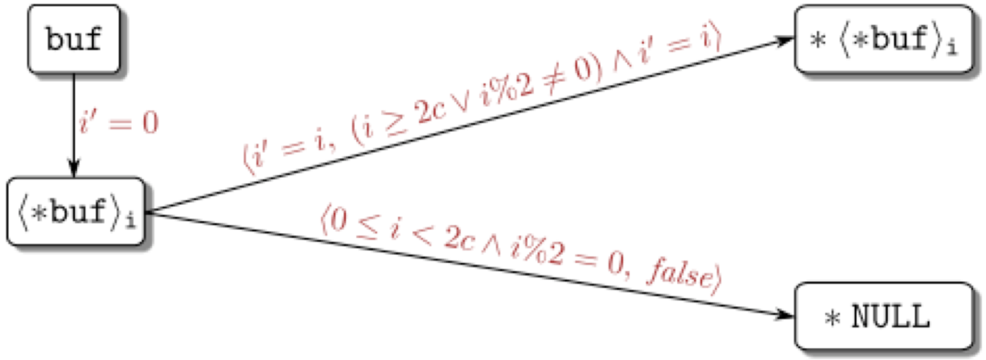


fixed-point

Symbolic Heap Analysis: Weakness

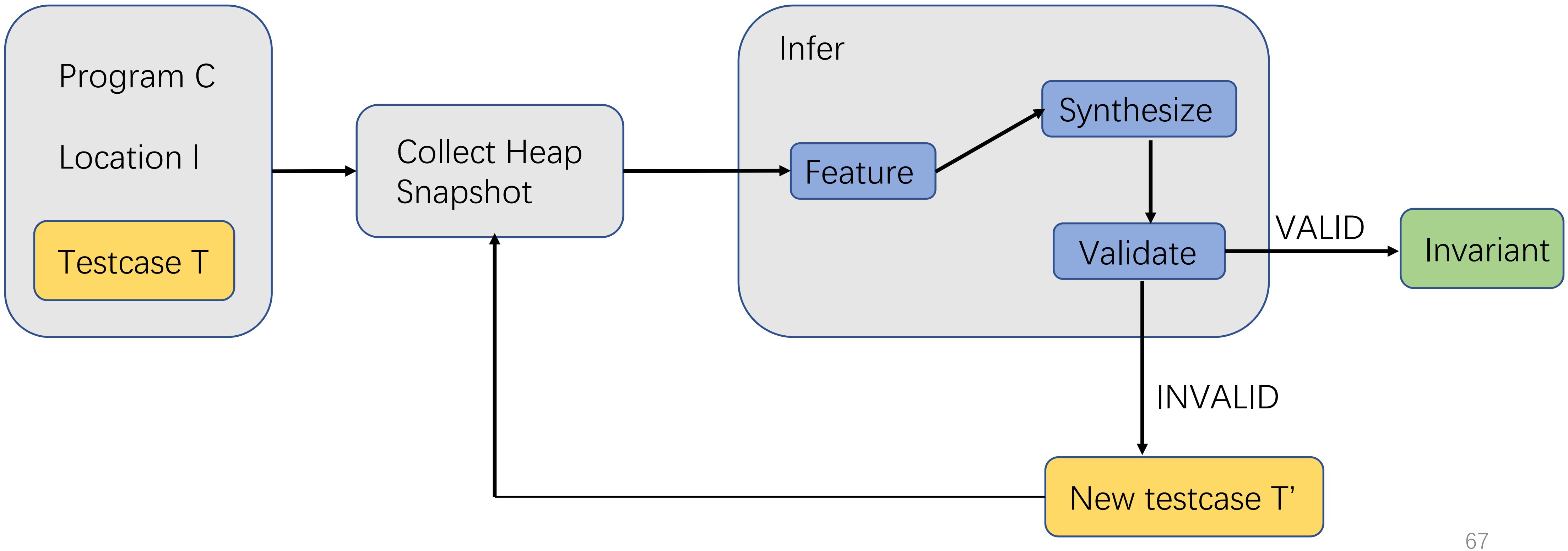
- Difficulty in determining updated edges

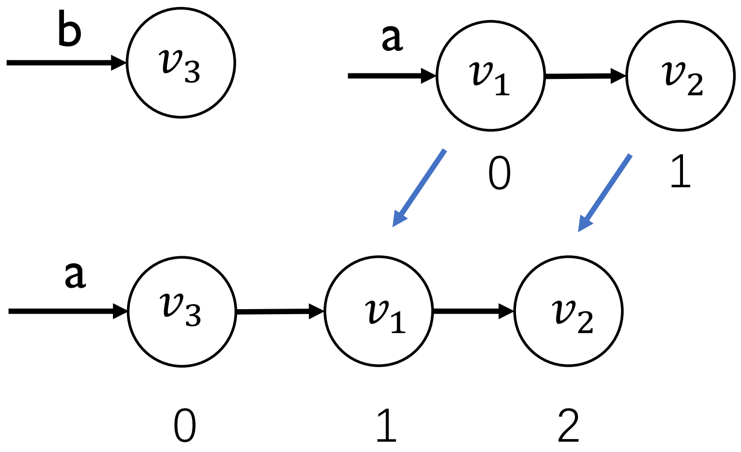
```
void send_packets(struct packet** buf, int c, int size) {
  assert(2*c <= size);
  for(int j=0; j< 2*c; j+=2)
    if (transmit_packet(buf[j]) == SUCCESS) { free(buf[j]); buf[j] = NULL; }
}
```



Data-driven Shape Analysis

- Verify and refine properties by verifier [Le, PLDI 2019] [Zhu, PLDI 2016]





$\text{sll}(a, nil) * \text{sll}(b, nil)$

insert_head

$\text{sll}(a, \text{nil})$