THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Shape Analysis: Principles, Applications, and Challenges

Presenter: **Chengpeng Wang**

Date: Oct 28th , 2019

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

What is shape analysis?

- Definition in [Jones and Muchnick 1981]
    - Determine the possible shapes of a dynamically allocated data structure at a given program point
- Reason the geometry structures of dynamically allocated heap data and their relations
    - Geometry structures
        - Is it a Tree, a DAG, or a Cyclic Graph?
        - Self-defined properties: sorted linked lists …
    - Structural relationship
        - Overlapping or disjoint?
- In general, shape analysis aims to property reasoning on heap structures

What are we concerned about

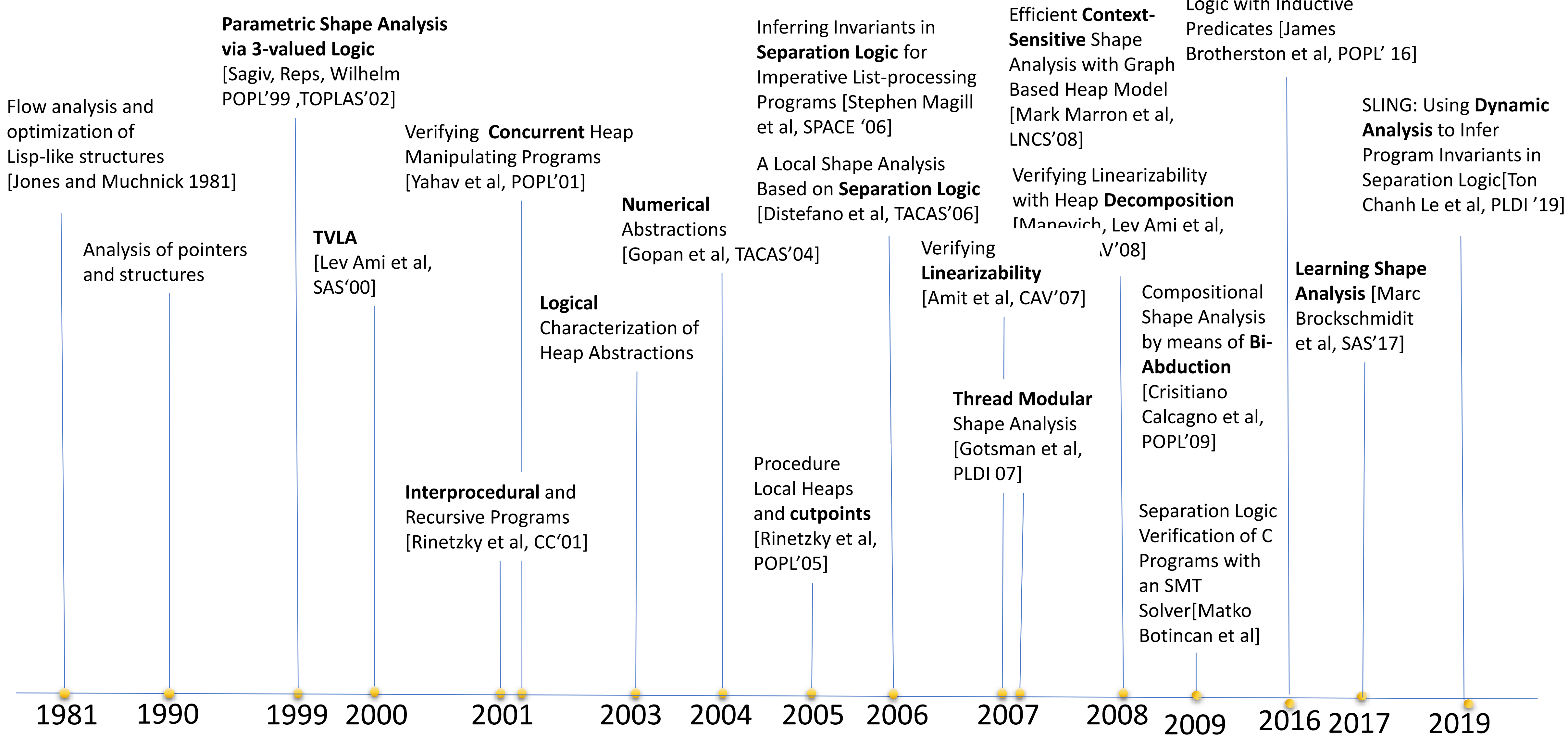
- Geometry structures
  - NULL pointers: Does a pointer have a NULL value?
  - Cyclicity: Is a heap cell part of a cycle?
  - Reachability: Is a heap cell reachable from any pointer variable?

- Structural relationship
  - Alias: Do two pointer expressions reference the same heap cell?
  - Sharing: Is a heap cell shared?
  - Memory Leak: Does a procedure or a program leave behind unreachable heap cells when it returns?

Historical Footprints

- Stage 1: Analogue pointer analysis
- Stage 2: Logical methods
  - 3-valued logic: TVLA
  - Separation Logic: INFER …
- Stage 3: Hybrid methods
  - Static analysis + dynamic analysis: SLING
  - Machine learning techniques: LOCUST

Historical Footprints



1981

Flow analysis and optimization of Lisp-like structures [Jones and Muchnick 1981]

1990

Analysis of pointers and structures

1999

Parametric Shape Analysis via 3-valued Logic [Sagiv, Reps, Wilhelm POPL'99 ,TOPLAS'02]

2000

TVLA [Lev Ami et al, SAS'00]

2001

Verifying **Concurrent** Heap Manipulating Programs [Yahav et al, POPL'01]

Interprocedural and Recursive Programs [Rinetzky et al, CC'01]

2003

Logical Characterization of Heap Abstractions

2004

Numerical Abstractions [Gopan et al, TACAS'04]

2005

Procedure Local Heaps and **cutpoints** [Rinetzky et al, POPL'05]

2006

Inferring Invariants in **Separation Logic** for Imperative List-processing Programs [Stephen Magill et al, SPACE '06]

A Local Shape Analysis Based on **Separation Logic** [Distefano et al, TACAS'06]

2007

Verifying **Linearizability** [Amit et al, CAV'07]

Thread Modular Shape Analysis [Gotsman et al, PLDI 07]

2008

Efficient **Context-Sensitive** Shape Analysis with Graph Based Heap Model [Mark Marron et al, LNCS'08]

Verifying Linearizability with Heap **Decomposition** [Manevich, Lev Ami et al, V'08]

2009

Compositional Shape Analysis by means of **Bi-Abduction** [Crisitiano Calcagno et al, POPL'09]

Separation Logic Verification of C Programs with an SMT Solver[Matko Botincan et al]

2016

Model Checking for Symbolic-heap Separation Logic with Inductive Predicates [James Brotherston et al, POPL' 16]

2017

Learning Shape Analysis [Marc Brockschmidit et al, SAS'17]

2019

SLING: Using **Dynamic Analysis** to Infer Program Invariants in Separation Logic[Ton Chanh Le et al, PLDI '19]

Stage 1: Analogue pointer analysis

- Research background
  - Automatic verification techniques were not developed
    - Constraint solving was still a tough problem
  - Static analysis based on abstract interpretation
    - Basic analyzer: pointer analysis, integer analysis…
    - Framework: Dataflow analysis
- In this stage, the aim of shape analysis is unclear
  - Focus on different properties of heap data in different application scenarios

Motivations

- Safety of Program
  - Check some properties without existing techniques, such as race condition, preserving cyclicity/acyclicity, etc..
- More efficient compilation
  - garbage collection, parallelization, etc..

Motivating example (1)

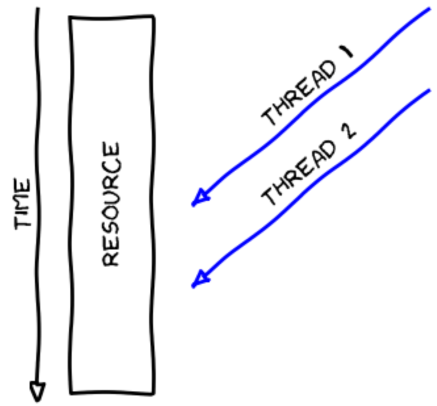
```
struct ListEntry
{
    ListEntry* next;
    ListEntry* prev;
    int data;
};
```

```
void *add(ListEntry* &h) {
    int data = rand()
    ListEntry* n;
    n = new ListEntry;
    n->data = data;
    …
}
```

```
void *remove(ListEntry* &h) {
    int data;
    ListEntry* n;
    if (n != h) {
        n->prev->next = n->next;
        n->next->prev = n->prev;
        data = n-> data;
        delete n;
    }
}
```

```
int operateTwoList(ListEntry* &a, ListEntry* &b) {
    pthread_t tids[2];
    int ret_add = pthread_create(&tids[0], NULL,  add,  &a);
    int ret_remove = pthread_create(&tids[1], NULL, remove, &b);
    pthread_exit(NULL);

    return 0;
}
```



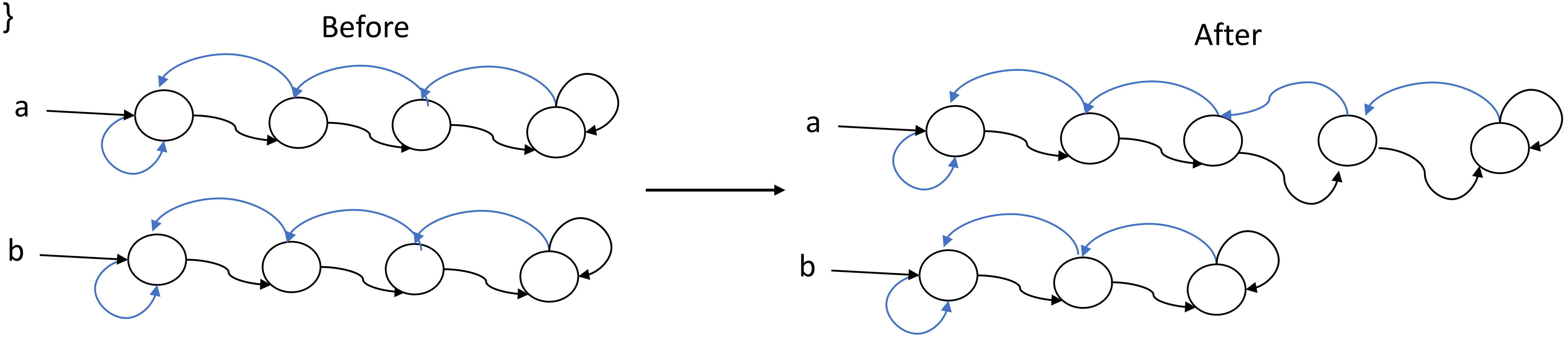
Motivating example (I)

```
int operateTwoList(ListEntry* &a, ListEntry* &b) {
    pthread_t tids[2];
    int ret_add = pthread_create(&tids[0], NULL, add,  &a);
    int ret_remove = pthread_create(&tids[1], NULL, remove, &b);
    pthread_exit(NULL);

    return 0;
}
```

Case 1: Disjoint

No data race
List a and b are deterministic after the execution of *operateTwoList*.



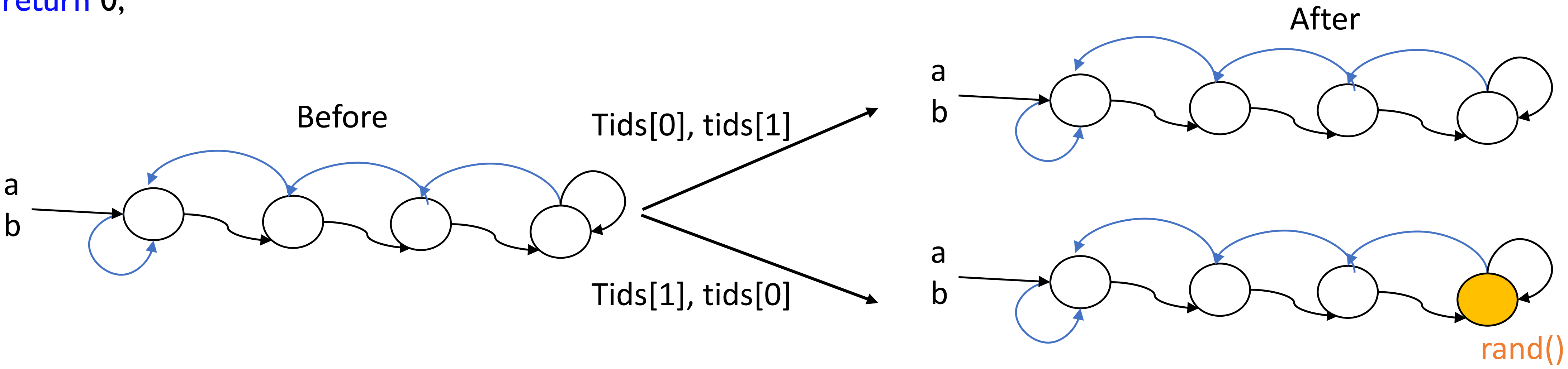
Motivating example (I)

```
int operateTwoList(ListEntry* &a, ListEntry* &b) {
    pthread_t tids[2];
    int ret_add = pthread_create(&tids[0], NULL, add,  &a);
    int ret_remove = pthread_create(&tids[1], NULL, remove, &b);
    pthread_exit(NULL);

    return 0;
}
```

Case 2:Overlapping

Data race!
List a and b are nondeterministic after the execution of *operateTwoList*.



Motivating example (I)

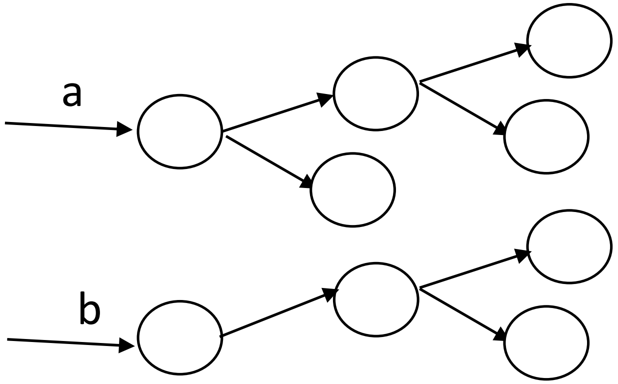
```
int operateTwoTree(TreeEntry* &a, TreeEntry* &b) {
    pthread_t tids[2];
    int ret_add = pthread_create(&tids[0], NULL, add,  &a);
    int ret_remove = pthread_create(&tids[1], NULL, remove, &b);
    pthread_exit(NULL);

    return 0;
}
```

- Even worse when operating two trees
  - Some subtrees are disjoint
  - Others are not

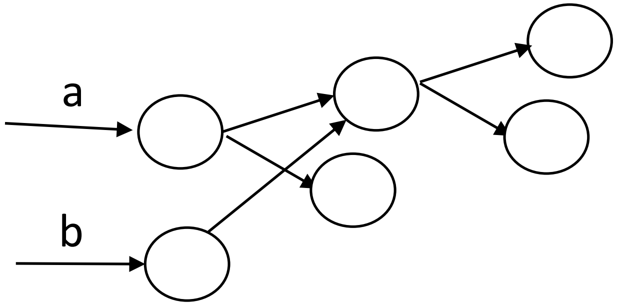
- Two structures are overlapping?

Disjoint



or

Overlapping



Motivating example (II)

```
Graph topologicalSortEntry(Graph g) {
    /*precondition: u is a DAG*/
    g = fun(g);
    assert(IsDAG(g));
    g = topologicalSort(g);
    return g;
}
```

Whether fun preserve the acyclicity?

Precondition of topological sorting: g should be a DAG

Motivating example (III)

```
int main() {
    ListEntry* a = createList();
    ListEntry* b = createList();

    /* Some operations on list a and list b*/
    …
    add(&a);
    Free the memory space of list a?

    /* memory consuming operations irrelevant to a */
    …

    add(&b)
    return 0;
}
```

- Conventional approach
  - Liveness analysis

- The recursive structures are complex
  - Some nodes are shared
  - Some nodes are not

- Two structures are overlapping?

How to handle

- Insight
  - The point-to relations between nodes are necessary and sufficient
  - Alias analysis and sharing analysis can help a lot
- Challenges
  - Unboundedness of recursive structures
    - K-bounded abstraction or On-demand abstraction

Stage 1: Analogue pointer analysis

- An extension of pointer analysis
  - Only interested in
    - Level 0: point-to analysis (Analysis of Pointers and Structures, 1990)
    - Level 1: alias analysis (Analysis of Pointers and Structures, 1990)
    - Level 2: “shape” analysis (Is it a Tree, a DAG, or a Cyclic Graph? 1996)
  - Features
    - Less powerful: **only a small set of properties can be expressed**
    - High scalability: the abstract domain is simple and easy to maintain

Stage 2：Logic Method

- Reflection
  - The limitations of shape analysis in stage 1
    - The expressivity is limited
    - The shape properties are different in previous works(lack of general approach)
- How to handle
  - Logic based approach: 3-valued logic, separation logic
  - Parametric framework: TVLA(a parametric framework via 3-valued logic)

Stage 2: Logic Method

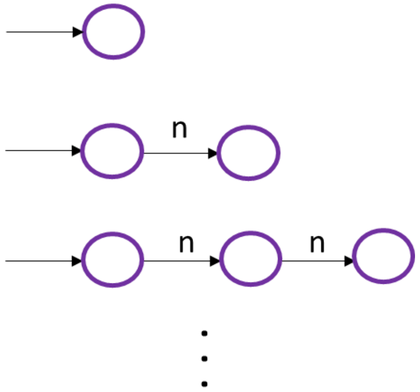
- TVLA: 3-valued logic based shape analysis
  - Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm
  - Representative works
    - Parametric Shape Analysis via 3-valued Logic(TR 1998, POPL 1999, TOPLAS 2002)
    - Semantic minimization of 3-valued propositional formulae(LICS 2002)
    - Finite differencing of logical formulas for static analysis(ESOP 2003)
    - A Relational Approach to Interprocedural Shape Analysis(SAS 2004)
    - …

TVLA

- Motivations
  - The expressivity is limited
    - Generate shape invariants by predicates
  - The approach is not general
    - Propose a parametric approach/framework to synthesize different kinds of shape invariants

TVLA

- Challenges
  - Dynamically allocated data structures are unbounded



  - The trade-off between precision and efficiency
    - More predicates are used, more precise shape information extracted while more overhead in shape analysis

TVLA

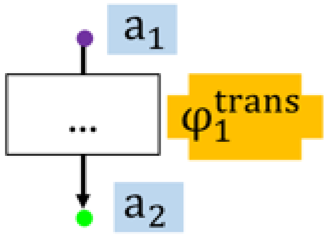
- Insight
  - Find a bounded abstract structure to represent dynamically allocated data
    - Embedding 2-valued logic structures into 3-valued logic structures
  - Find a flexible configuration of predicates.
    - Some special predicates(Core Predicates) are used to restore shape info, and others are used in an on-demand way in particular cases if necessary.

TVLA I: Intraprocedural

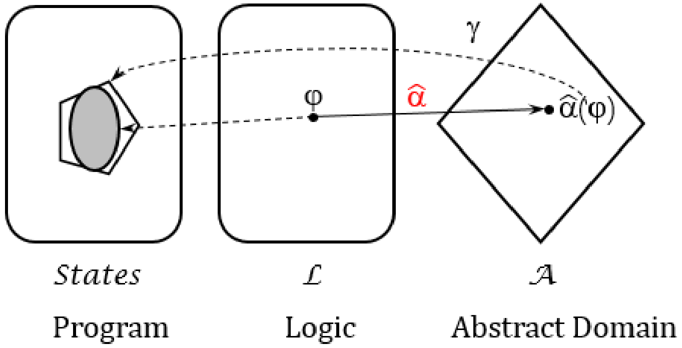
- Based on symbolic execution
  - Transfer relation
    - Statement-wise
    - *Function-wise(Interprocedural)*
  - Abstract domain
    - Unboundedness of dynamically allocated data can make symbolic execution unterminable
    - Abstract domain assures the boundedness of the abstract structure

TVLA I: Intraprocedural

- How to achieve
  - Solve these subproblems in a unified way: logic



Logic formula can encode the transfer relation by modeling the semantics of the statement



Logic constraint can encode the abstract state, in which each the concrete state satisfies the logic constraint

TVLA I: Intraprocedural

- Core task: update memory configuration
  - Choose **a set of predicates** to describe the memory configuration precisely and obtain shape graph
  - Design a **canonical abstraction** to make shape graph in a bounded size
  - Update the shape graph based on the **update formula** defined by the semantics of the statement [The most important subtask]
    - Canonical abstraction assures the terminability
    - Predicates guide canonical abstraction
- It is the essential problem to update formula as precise as possible

TVLA I: Intraprocedural

- Problem 1: How to abstract
- Problem 2: How to use predicates

Shape Abstraction

- Problem 3: How to embed

Canonical Embedding

- *Problem 4: How to update formula according to the statement*
  - *Focus and Coerce*

Formula Update

Shape Abstraction

- Motivating example

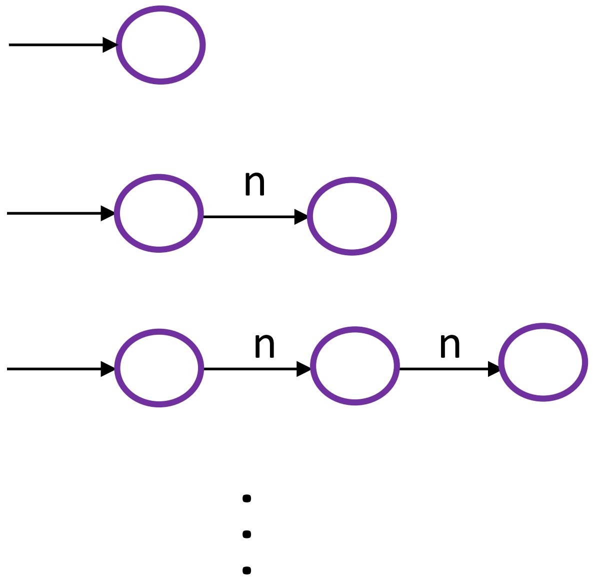
```
typedef struct node {
    struct node *n;
    int data;
} *List;
```

```
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
        y = y->n;
    }
    t = malloc();
    t->data = d;
    e = y->n;
    t->n = e;
    y->n = t;
}
```

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

- Motivating example

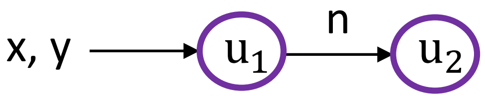


unboundedness

Shape Abstraction

- Model shape info in 2-valued logic(Standard First Order Logic)
  - S: logical structure, denoted by $< U^S,\ l^S >$
    - U^S: A universe of individuals
    - l^S maps arity-k predicate and k-tuple of individuals to 0(false) or 1(true)
  - Example
    - $q(n)$: Does pointer variable q point to element n?
    - $n(v_1, v_2)$: Does the n field of v_1 point to v_2?

Encode shape graph in a logical way



Unary predicates

Indiv.	x	y
u_1	1	1
u_2	0	0

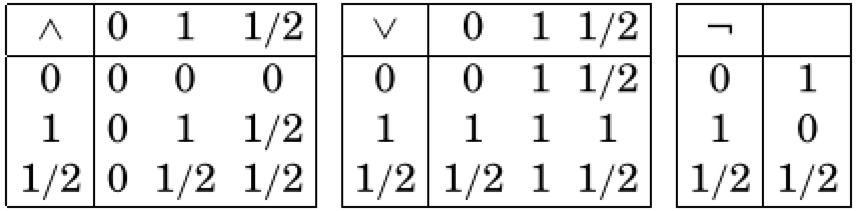
Binary predicates

n	u_1	u_2
u_1	0	1
u_2	0	0

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

- Model shape info in 3-valued logic
  - S: logical structure, denoted by $< U^S,\ \iota^S >$
    - U^S: A universe of individuals
    - ι^S maps arity-k predicate and k-tuple of individuals to 0(false), 1(true) or 1/2(unknown)



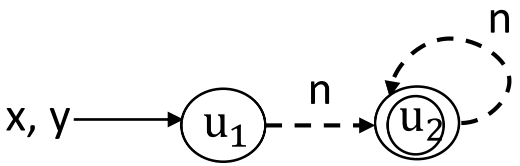
$\wedge$	0	1	1/2
0	0	0	0
1	0	1	1/2
1/2	0	1/2	1/2

$\vee$	0	1	1/2
0	0	1	1/2
1	1	1	1
1/2	1/2	1	1/2

$\neg$	
0	1
1	0
1/2	1/2

- Example

Encode shape graph in a logical way



- $sm(v)$: Does v represent more than one concrete individuals?
- $q(n)$: Does pointer variable q point to element n?
- $n(v_1, v_2)$: Does the n field of v_1 point to v_2?

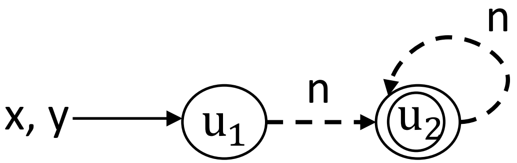
Indiv.	x	y	sm
u_1	1	1	0
u_2	0	0	1/2

n	u_1	u_2
u_1	0	**1/2**
u_2	0	**1/2**

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

- Why use 3-valued logic rather than 2-valued logic
  - 2-valued logic can not encode may point-to relation



3-valued logic

n	u_1	u_2
u_1	0	**1/2**
u_2	0	**1/2**

2-valued logic

n	u_1	u_2
u_1	0	**?**
u_2	0	**?**

Unboundedness calls for summary node

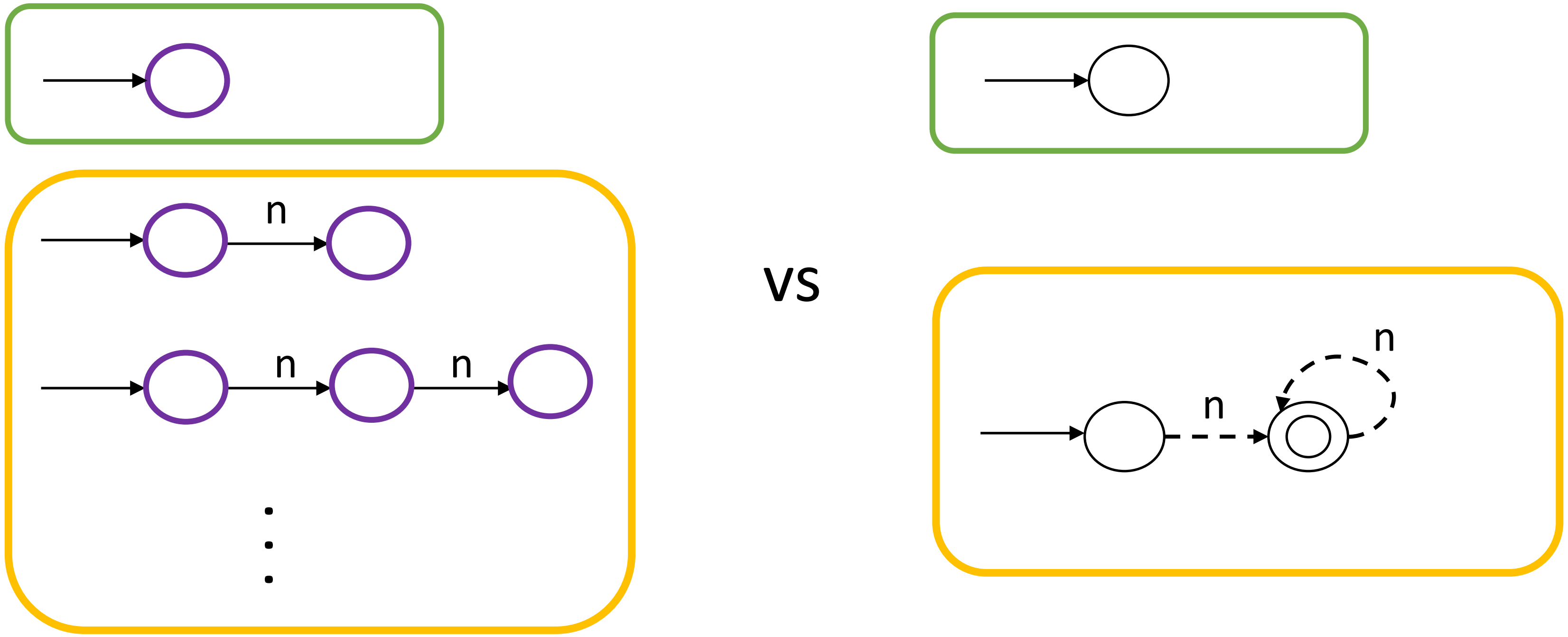
Summary node calls for 3-valued logic

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

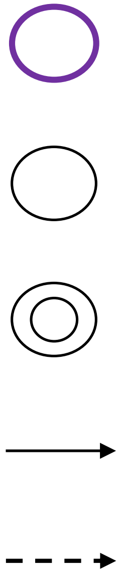
- Motivating example

```
typedef struct node {
    struct node *n;
    int data;
} *List;
```



unboundedness

boundness



Concrete Individual

Individual Node

Summary Node

Must Point-to Edge

May Point-to Edge

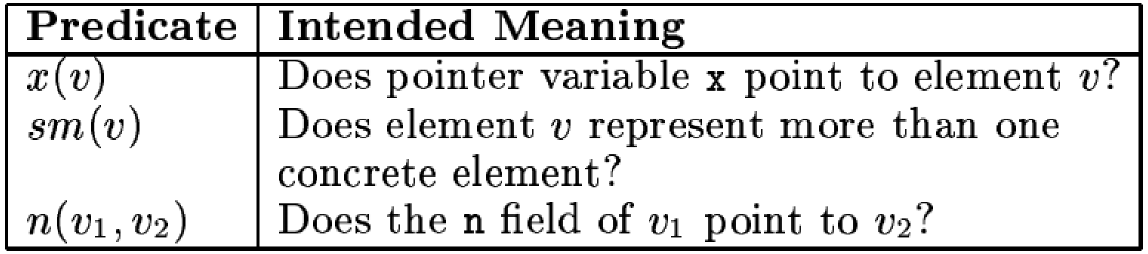
Shape Abstraction

- About predicates
  - Predicates are divided into two sets
    - Core predicates: describe the shape analysis precisely
    - Instrumentation predicates: for continence and specific use

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

- About predicates
  - Core predicates: describe the shape analysis precisely

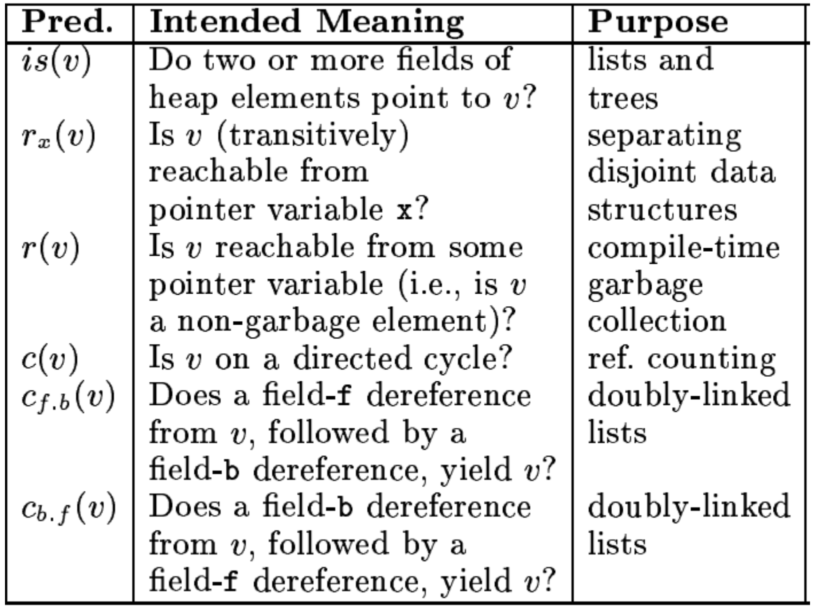


Predicate	**Intended Meaning**
$x(v)$	Does pointer variable **x** point to element v?
$sm(v)$	Does element v represent more than one concrete element?
$n(v_1, v_2)$	Does the **n** field of v_1 point to v_2?

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Shape Abstraction

- About predicates
  - Instrumentation predicates: for canonical abstraction and verification

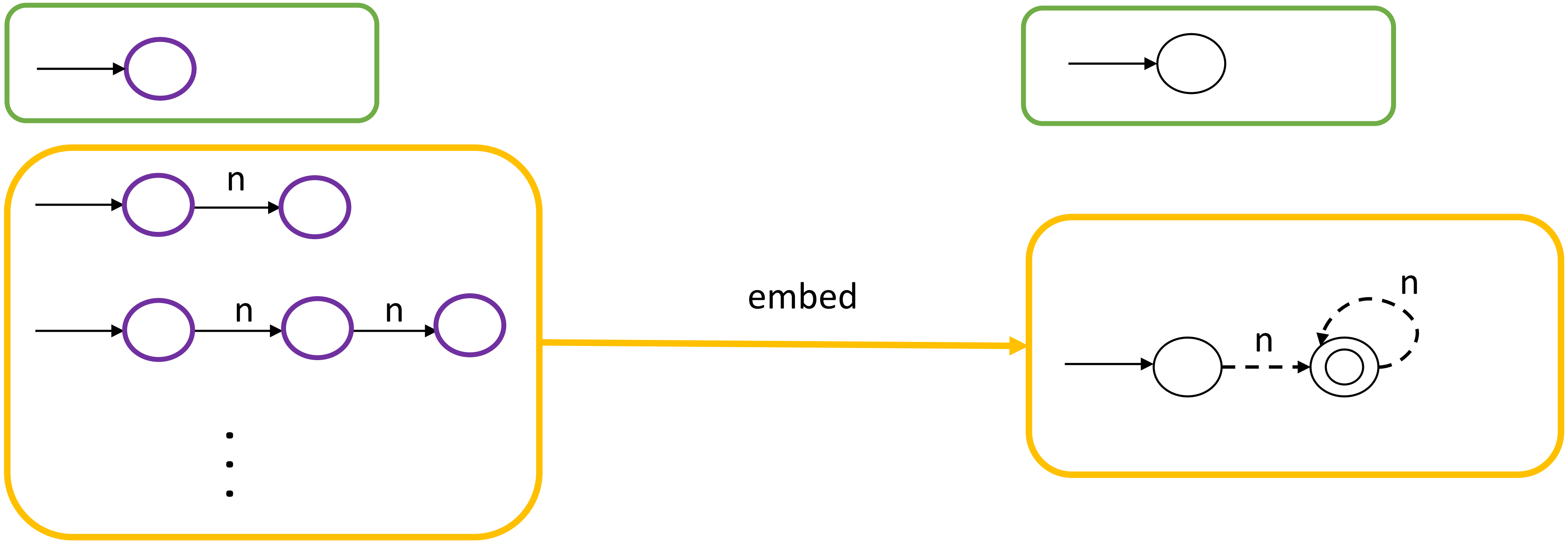


Pred.	Intended Meaning	Purpose
$is(v)$	Do two or more fields of heap elements point to v?	lists and trees
$r_x(v)$	Is v (transitively) reachable from pointer variable **x**?	separating disjoint data structures
$r(v)$	Is v reachable from some pointer variable (i.e., is v a non-garbage element)?	compile-time garbage collection
$c(v)$	Is v on a directed cycle?	ref. counting
$c_{f.b}(v)$	Does a field-**f** dereference from v, followed by a field-**b** dereference, yield v?	doubly-linked lists
$c_{b.f}(v)$	Does a field-**b** dereference from v, followed by a field-**f** dereference, yield v?	doubly-linked lists

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- Back to motivating example



Unbounded 2-valued logic structures

Bounded 3-valued logic structures

Canonical Embedding

- $S =< U^S, l^S > \quad S' =< U^{S'}, l^{S'} > \quad f: U^S \rightarrow U^{S'}$ is a surjective function

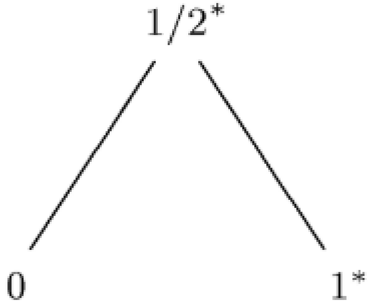
  we say that f **embeds S in S'(denoted by $S \sqsubseteq^f S'$)** if

  - For every predicate p of arity k and all $u_1, \ldots, u_k \in U^S$

    $$l^S(p)(u_1, \ldots, u_k) \sqsubseteq l^{S'}(p)\big(f(u_1), \ldots, f(u_k)\big)$$

  - For all $u' \in U^{S'}$

    $$(|\{\mathrm{u} \mid f(\mathrm{u}) = \mathrm{u}'\}| > 1) \sqsubseteq l^{S'}(sm)(\mathrm{u}')$$



$0 \sqsubseteq 1/2$

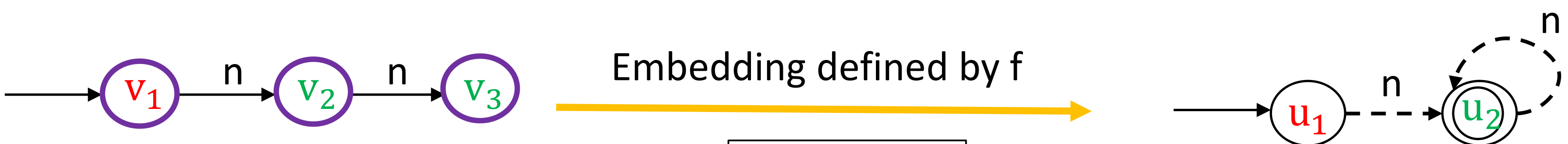
$1 \sqsubseteq 1/2$

- We say that **S can be embedded is S'(denoted by $S \sqsubseteq S'$)** if there exists a function f such that $S \sqsubseteq^f S'$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- An example $l^{S}(p)(u_1,\ldots,u_k) \sqsubseteq l^{S'}(p)\big(f(u_1),\ldots,f(u_k)\big)$



2-valued logic structures

$$\begin{aligned} f(\mathrm{v}_1) &= \mathrm{u}_1 \\ f(\mathrm{v}_2) &= \mathrm{u}_2 \\ f(\mathrm{v}_3) &= \mathrm{u}_2 \end{aligned}$$

3-valued logic structures

n	v_1	v_2	v_3
v_1	0	$\boxed{1}$	0
v_2	0	0	$\boxed{1}$
v_3	0	0	0

$$n(v_1,v_2) = 1 \sqsubseteq 1/2 = n(u_1,u_2)$$

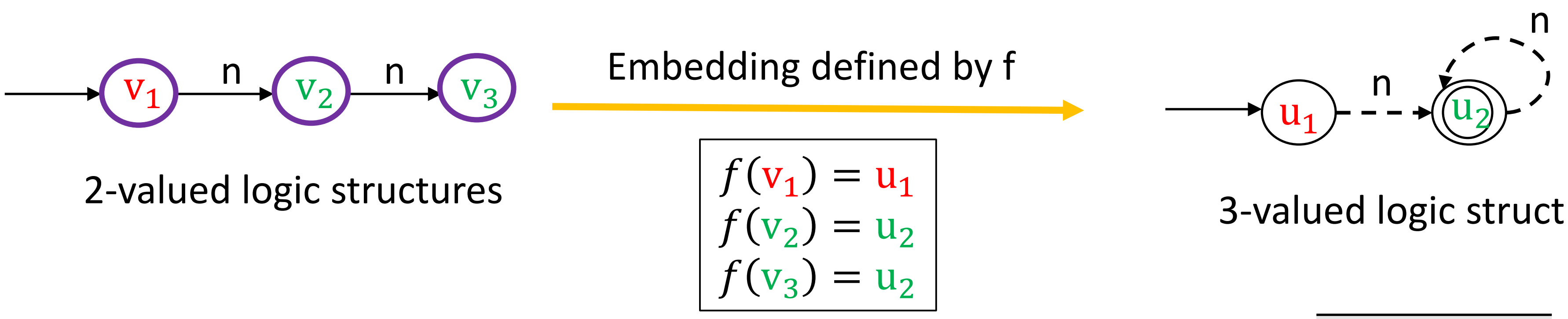
$$n(v_2,v_3) = 1 \sqsubseteq 1/2 = n(u_2,u_2)$$

n	u_1	u_2
u_1	0	$\boxed{1/2}$
u_2	0	$\boxed{1/2}$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- An example $(|\{\mathrm{u} \mid f(\mathrm{u}) = \mathrm{u}'\}| > 1) \sqsubseteq l^{S'}(sm)(\mathrm{u}')$



2-valued logic structures

Embedding defined by f

3-valued logic structures

$$f(\mathrm{v}_1) = \mathrm{u}_1$$
$$f(\mathrm{v}_2) = \mathrm{u}_2$$
$$f(\mathrm{v}_3) = \mathrm{u}_2$$

Indiv.	sm
u_1	0
u_2	1/2

$$(|\{v \mid f(\mathrm{v}) = \mathrm{u}_1\}| > 1) = (|\{\mathrm{v}_1\}| > 1) = (1 > 1) = 0 \sqsubseteq 0$$

$$(|\{v \mid f(\mathrm{v}) = \mathrm{u}_2\}| > 1) = (|\{\mathrm{v}_2, \mathrm{v}_3\}| > 1) = (2 > 1) = 1 \sqsubseteq 1/2$$

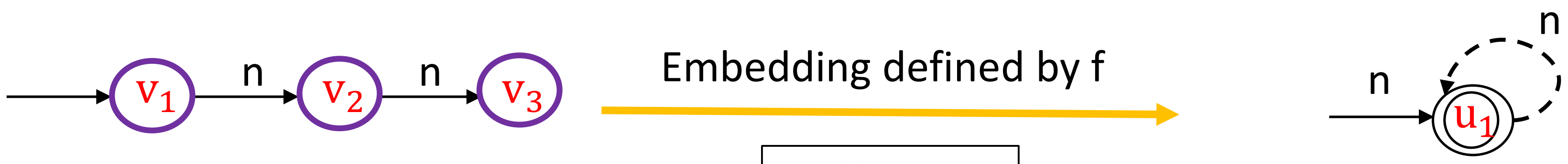
Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- Embedding is not unique

$$\iota^{S}(p)(u_1,\dots,u_k) \sqsubseteq \iota^{S'}(p)\big(f(u_1),\dots,f(u_k)\big)$$

$$(|\{\mathrm{u} \mid f(\mathrm{u}) = \mathrm{u}'\}| > 1) \sqsubseteq \iota^{S'}(sm)(\mathrm{u}')$$



$\boxed{f(\cdot) = \mathrm{u}_1}$

Severe Information Loss!

2-valued logic structures

3-valued logic structures

n	v_1	v_2	v_3
v_1	0	1	0
v_2	0	0	1
v_3	0	0	0

$$n(v_1,v_2) = 1 \sqsubseteq 1/2 = n(u_1,u_1)$$

$$n(v_2,v_3) = 1 \sqsubseteq 1/2 = n(u_1,u_1)$$

n	u_1
u_1	1/2

Indiv.	sm
u_1	1/2

$$(|\{\mathrm{v} \mid f(\mathrm{v}) = \mathrm{u}_1\}| > 1) = (|\{\mathrm{v}_1,\mathrm{v}_2,\mathrm{v}_3\}| > 1) = (3 > 1) = 1 \sqsubseteq 1/2$$

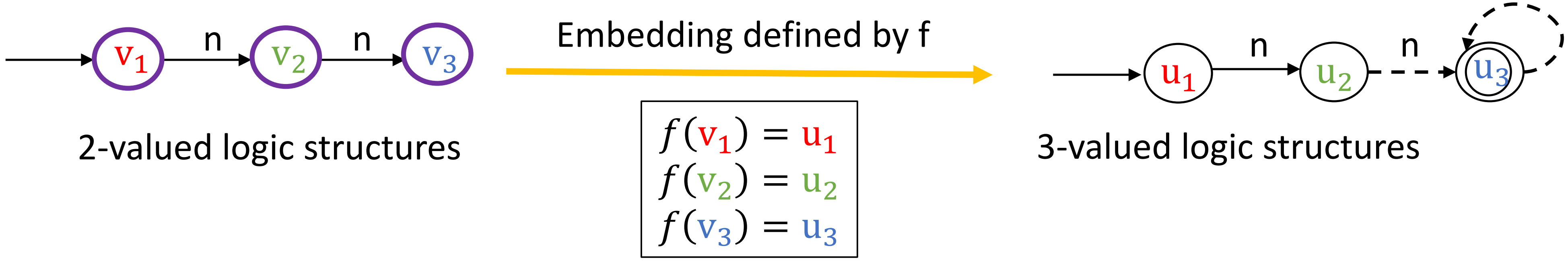
Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- Another Embedding

$$l^S(p)(u_1, \ldots, u_k) \sqsubseteq l^{S'}(p)\big(f(u_1), \ldots, f(u_k)\big)$$

$$(|\{u \mid f(u) = u'\}| > 1) \sqsubseteq l^{S'}(sm)(u')$$



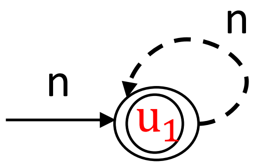
Embedding defined by f

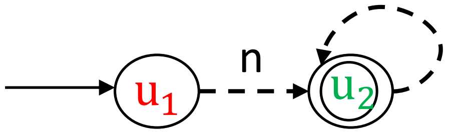
$$f(\mathrm{v}_1) = \mathrm{u}_1$$
$$f(\mathrm{v}_2) = \mathrm{u}_2$$
$$f(\mathrm{v}_3) = \mathrm{u}_3$$

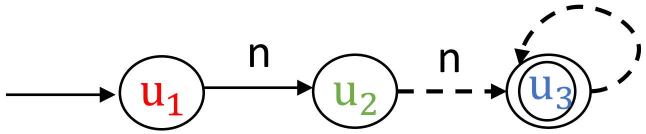
2-valued logic structures

3-valued logic structures

- Which one is the best







Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- How to embed with less loss of shape information
  - Embed based on **abstraction predicates**
- Canonical Embedding

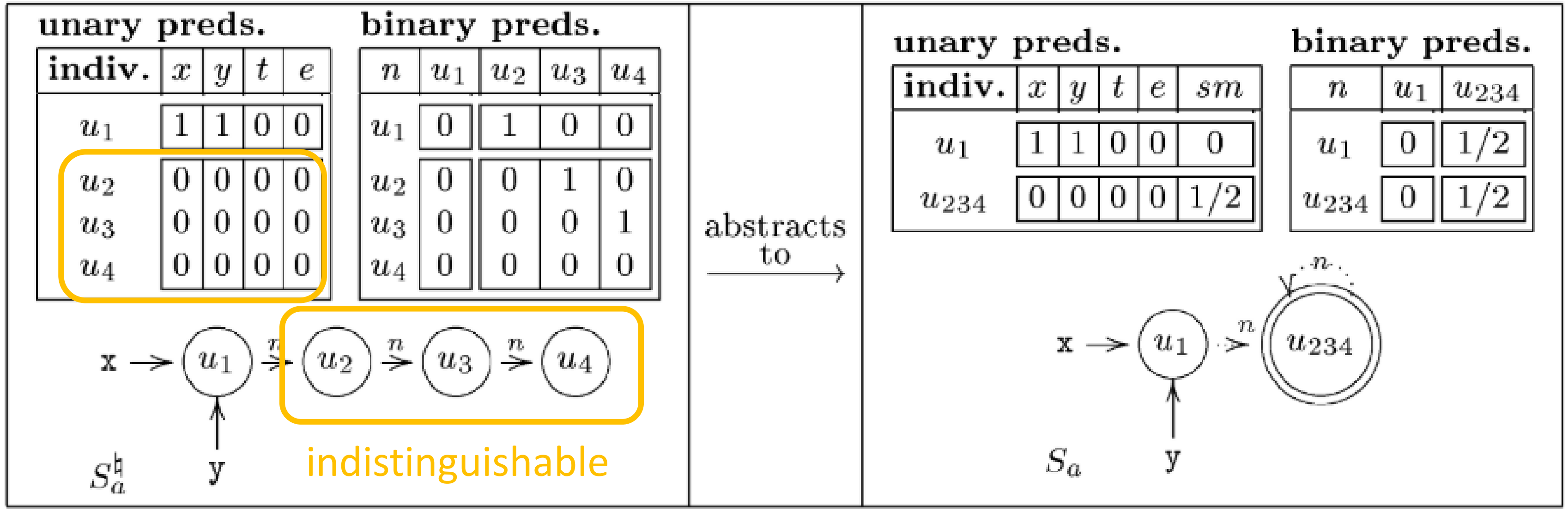
  Canonical name, just a symbol

  - $f_{embed_c}(v) = \boxed{u_{\{p \in A | l^S(p)(v) = 1\},\{p \in A | l^S(p)(u) = 0\}}}$
  - A is the set of abstraction predicates
- Abstraction predicates distinguish concrete individuals
  - $f_{embed_c}(v_1) = f_{embed_c}(v_2) \quad \Leftrightarrow p(v_1) = p(v_2) \quad \forall p \in A$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

- An example $f_{embed_c}(v_1) = f_{embed_c}(v_2) \quad \Leftrightarrow p(v_1) = p(v_2) \quad \forall p \in A$
  - Abstraction predicates: $A = \{x, y, t, e\}$



unary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

indistinguishable

abstracts to

unary preds.

indiv.	x	y	t	e	sm
u_1	1	1	0	0	0
u_{234}	0	0	0	0	1/2

binary preds.

n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Canonical Embedding

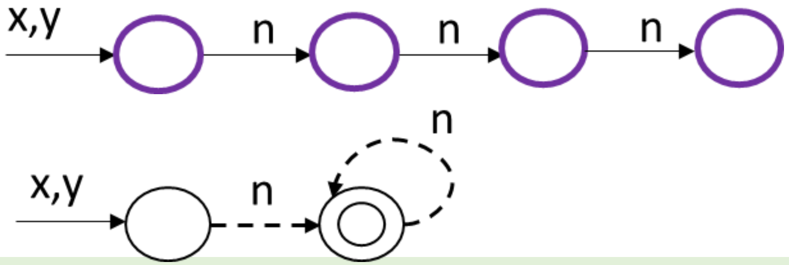
- Question: How to choose appropriate abstraction predicates?

$$f_{embed_c}(v_1) = f_{embed_c}(v_2) \quad \Leftrightarrow p(v_1) = p(v_2) \quad \forall p \in A$$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Based on symbolic execution
  - For each statement, transform each logic structure from last program location to the next one
- An example
  - Statement y = y->n
  - Logic structure



```
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
      y = y->n;
    }
    ...
}
```

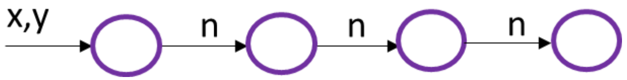
Formula Update

- An example
  - Statement y = y->n

```
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
        y = y->n;
    }
    …
}
```

Predicate Update Formulae	$x'(v) = x(v)$ $y'(v) = \exists v_1 : y(v_1) \wedge n(v_1, v)$ $t'(v) = t(v)$ $e'(v) = e(v)$ $n'(v_1, v_2) = n(v_1, v_2)$

  - 2-valued Logic structure



Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

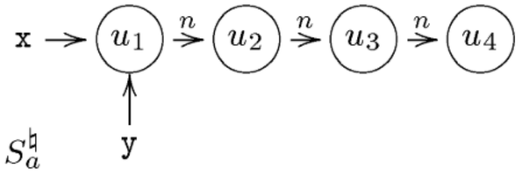
Structure Before

unary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0



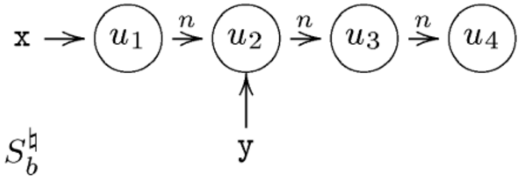
Structure After

unary preds.

indiv.	x	y	t	e
u_1	1	0	0	0
u_2	0	1	0	0
u_3	0	0	0	0
u_4	0	0	0	0

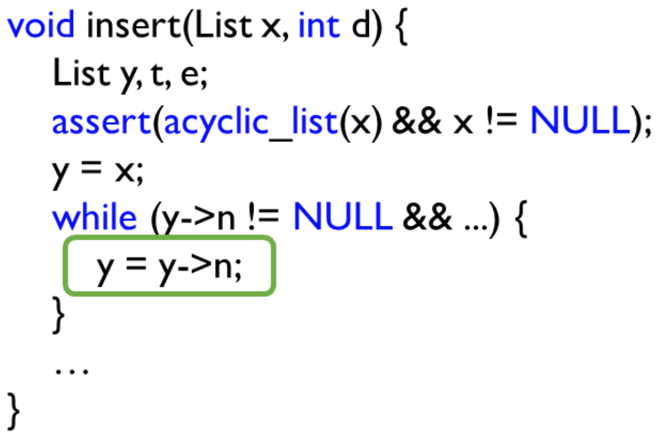
binary preds.

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

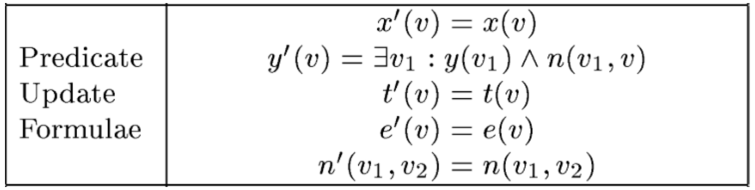


Formula Update

- Predicate Update Formulae
  - Core predicates
- How to update Instrumentation Predicate
  - Based on updated core predicates
    - Expressed by core predicates
  - Incremental update
    - Why? Reevaluation is a terrible burden
    - How? Finite differencing for instrumentation predicate update



```
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
        y = y->n;
    }
    ...
}
```



Predicate Update Formulae	$x'(v) = x(v)$
	$y'(v) = \exists v_1 : y(v_1) \wedge n(v_1, v)$
	$t'(v) = t(v)$
	$e'(v) = e(v)$
	$n'(v_1, v_2) = n(v_1, v_2)$

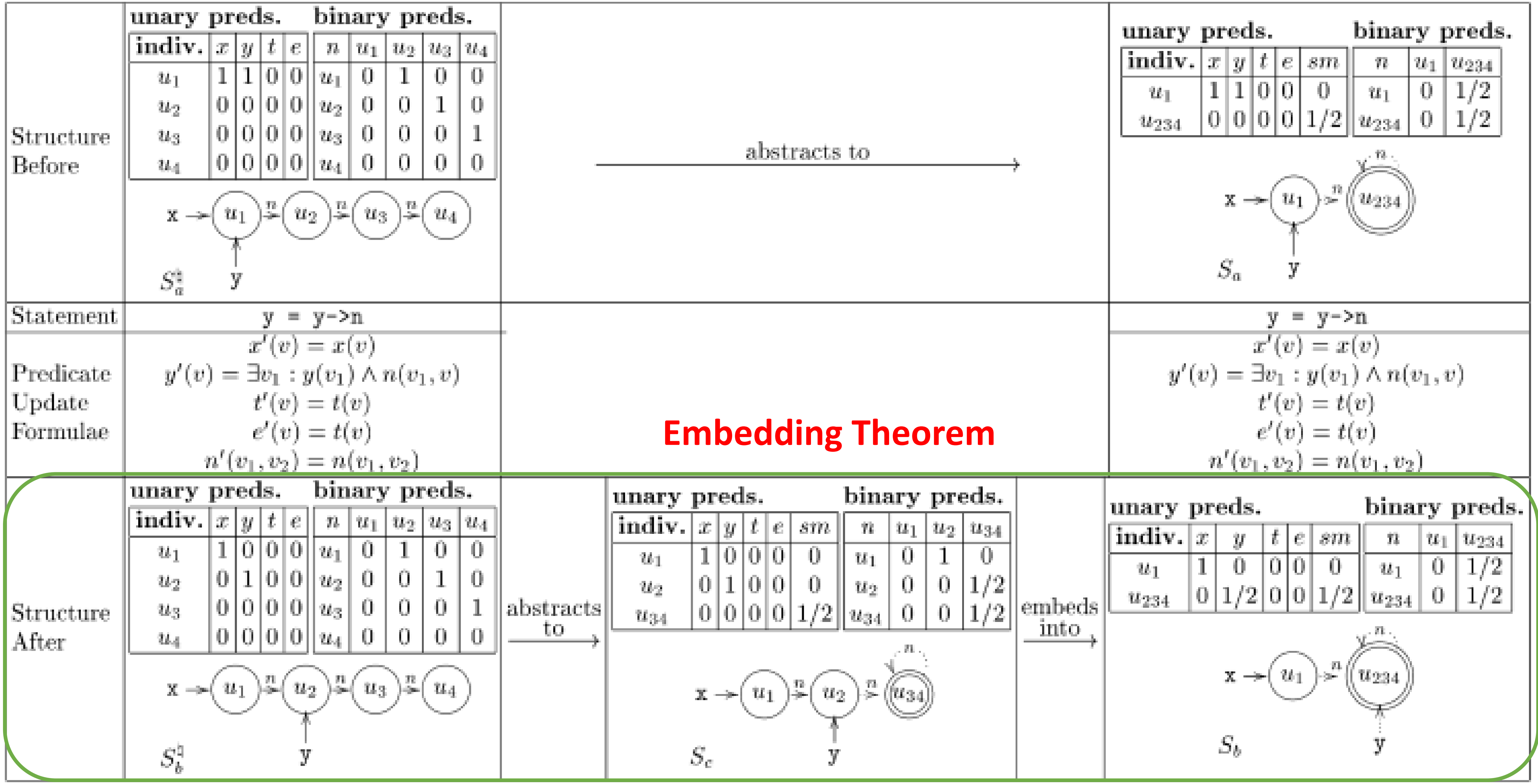
Reps T, Sagiv M, Loginov A. **Finite differencing of logical formulas for static analysis**[C] ESOP 2003

Formula Update

- Question: What happens if the program contains a bug
  - For example, y=NULL before the statement y = y->n
    - How should we transfer predicate values ?

Discussion: isNull can be defined as an instrumentation predicate. Before the formula updates of the statements like y=y->n, check whether isNull is 1 or not.

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002



Structure Before

unary preds. / binary preds.

indiv.	x	y	t	e
u_1	1	1	0	0
u_2	0	0	0	0
u_3	0	0	0	0
u_4	0	0	0	0

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

abstracts to

unary preds. / binary preds.

indiv.	x	y	t	e	sm
u_1	1	1	0	0	0
u_{234}	0	0	0	0	1/2

n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2

Statement

y = y->n

Predicate Update Formulae

$$x'(v) = x(v)$$
$$y'(v) = \exists v_1 : y(v_1) \wedge n(v_1, v)$$
$$t'(v) = t(v)$$
$$e'(v) = t(v)$$
$$n'(v_1, v_2) = n(v_1, v_2)$$

y = y->n

$$x'(v) = x(v)$$
$$y'(v) = \exists v_1 : y(v_1) \wedge n(v_1, v)$$
$$t'(v) = t(v)$$
$$e'(v) = t(v)$$
$$n'(v_1, v_2) = n(v_1, v_2)$$

Embedding Theorem

Structure After

unary preds. / binary preds.

indiv.	x	y	t	e
u_1	1	0	0	0
u_2	0	1	0	0
u_3	0	0	0	0
u_4	0	0	0	0

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0

abstracts to

unary preds. / binary preds.

indiv.	x	y	t	e	sm
u_1	1	0	0	0	0
u_2	0	1	0	0	0
u_{34}	0	0	0	0	1/2

n	u_1	u_2	u_{34}
u_1	0	1	0
u_2	0	0	1/2
u_{34}	0	0	1/2

embeds into

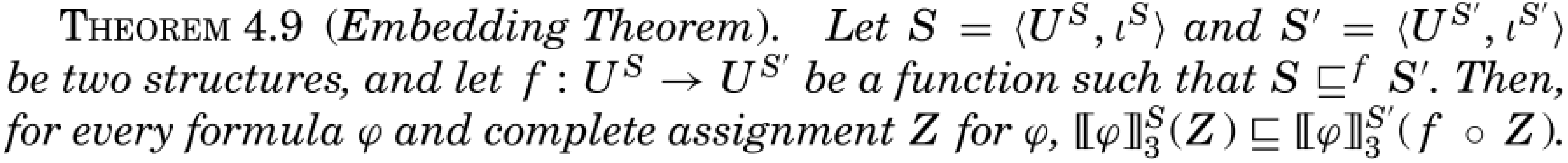
unary preds. / binary preds.

indiv.	x	y	t	e	sm
u_1	1	0	0	0	0
u_{234}	0	1/2	0	0	1/2

n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2

Formula Update

- Embedding Theorem



Theorem 4.9 (*Embedding Theorem*). *Let* $S = \langle U^S, \iota^S \rangle$ *and* $S' = \langle U^{S'}, \iota^{S'} \rangle$ *be two structures, and let* $f : U^S \to U^{S'}$ *be a function such that* $S \sqsubseteq^f S'$. *Then, for every formula* φ *and complete assignment* Z *for* φ, $[\![\varphi]\!]_3^S(Z) \sqsubseteq [\![\varphi]\!]_3^{S'}(f \circ Z)$.

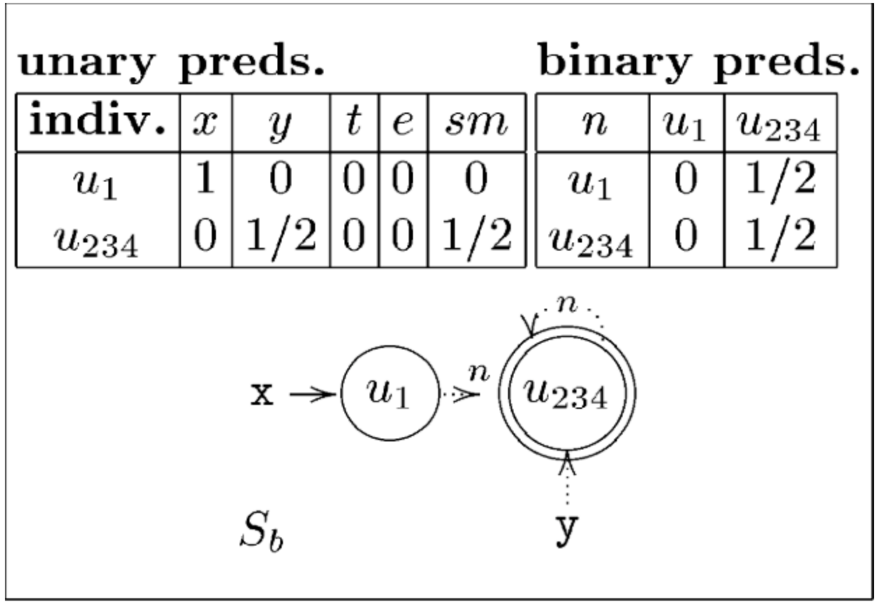
- It is sound to use an abstract(3-valued logic) structure S to answer questions about properties of the concrete(2-valued logic) structures that S represents.

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- However, how to update the following structure by y = y->n
    - The summary node u_{234} blur the truth
    - Can we recover the truth?
        - Yes!
        - Focus and Coerce

$$\begin{tikzcd}
\{S_a\} \arrow[rrr, "focus"] \arrow[d, "\llbracket st_0 \rrbracket_3"'] & & & \{S_{a,f,0}, S_{a,f,1}, S_{a,f,2}\} \arrow[d, "\llbracket st_0 \rrbracket_3"] \\
\{S_b\} & \sqsupseteq & \{S_{b,1}, S_{b,2}\} & \{S_{a,o,0}, S_{a,o,1}, S_{a,o,2}\} \arrow[l, "coerce"']
\end{tikzcd}$$

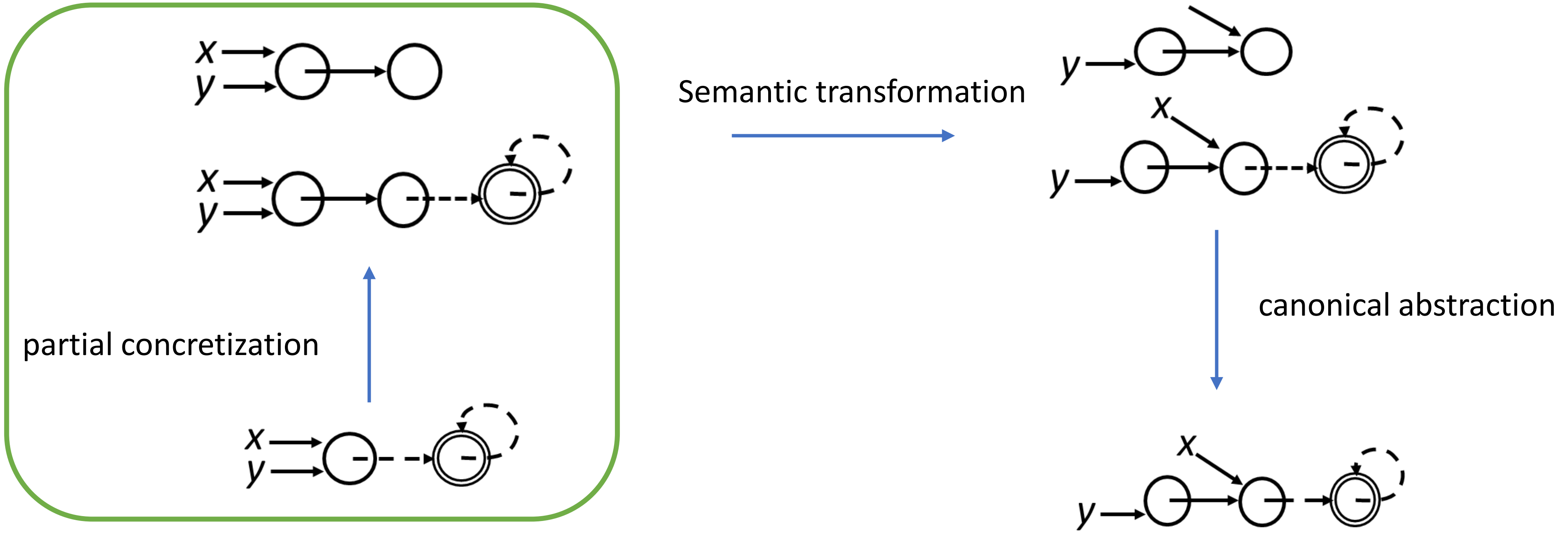


unary preds.						binary preds.		
indiv.	x	y	t	e	sm	n	u_1	u_{234}
u_1	1	0	0	0	0	u_1	0	1/2
u_{234}	0	1/2	0	0	1/2	u_{234}	0	1/2

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Recover the truth: Focus



partial concretization

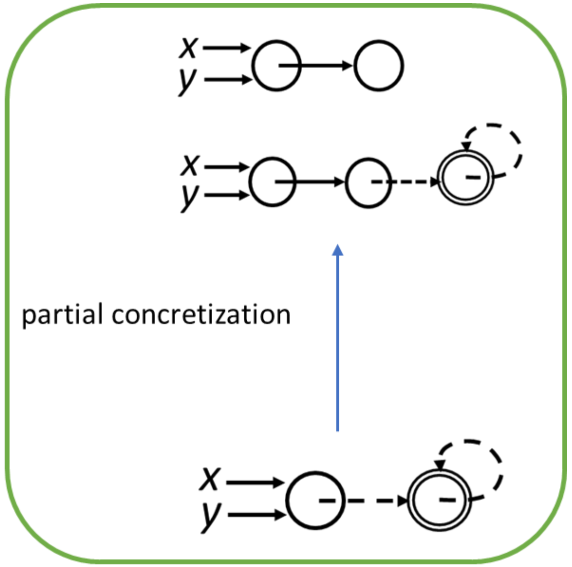
Semantic transformation

canonical abstraction

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

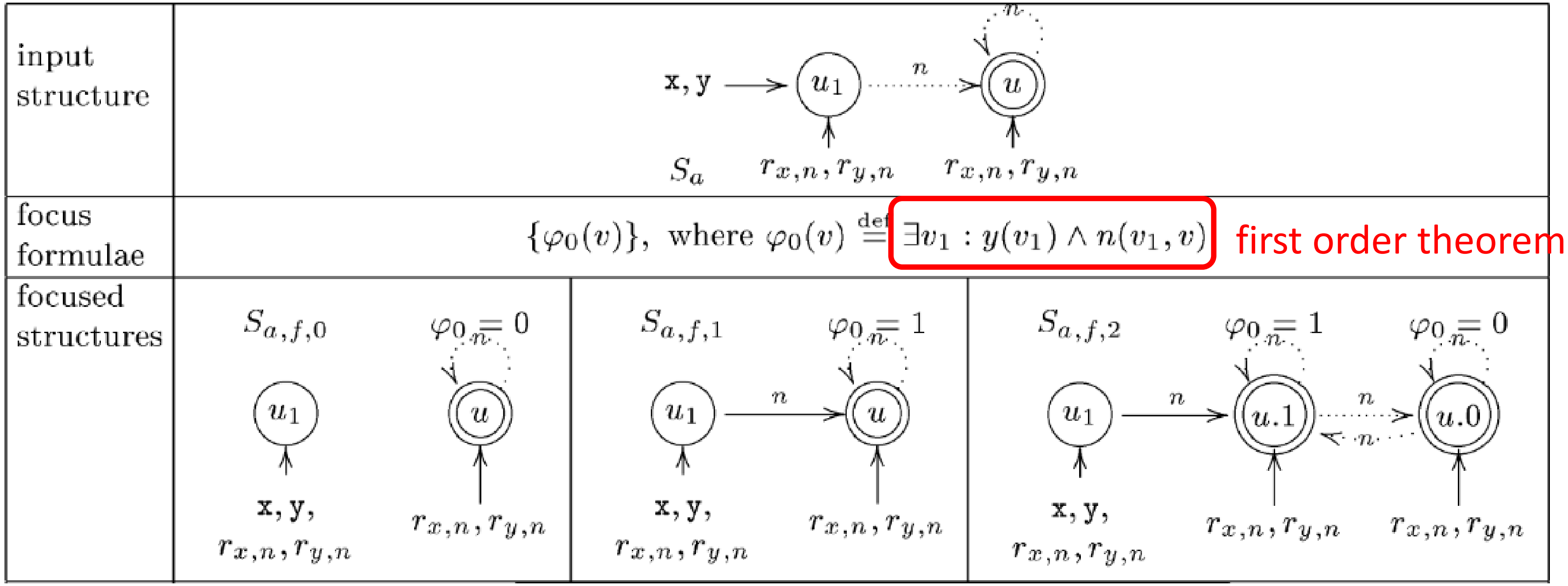
- Core idea of Focus
  - The formulae that define the meaning of st evaluate to definite values
  - The Focus operation brings these formulae “into focus”



Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- An example $st_0\colon y = y \rightarrow n$

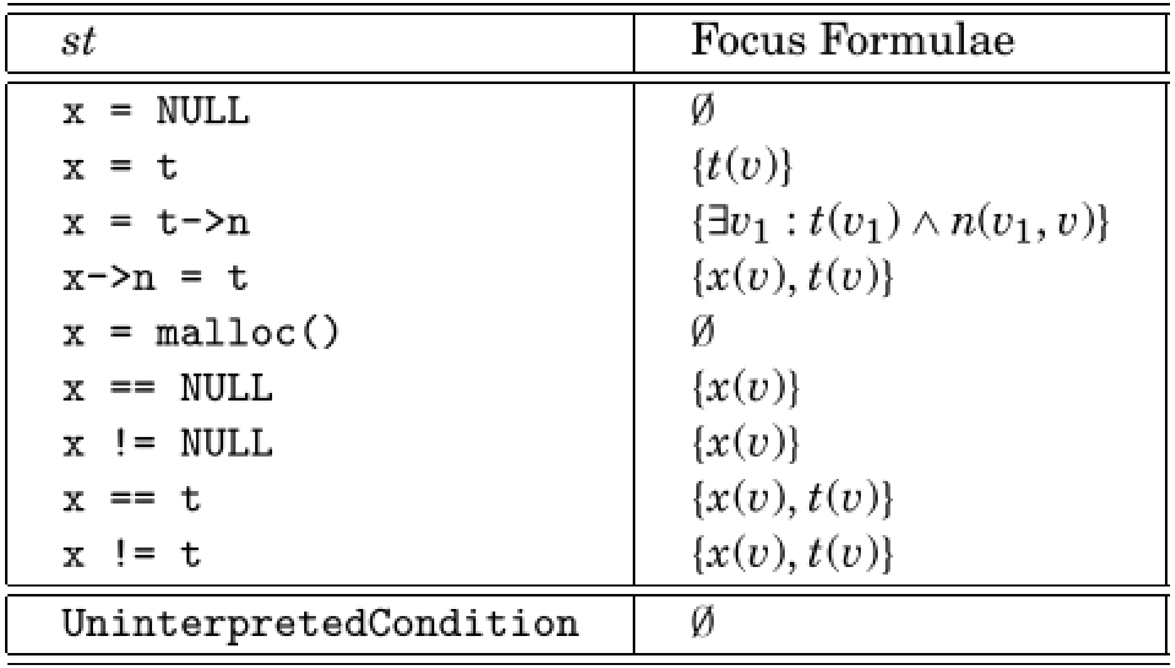


$\{\varphi_0(v)\}$, where $\varphi_0(v) \stackrel{\text{def}}{=} \boxed{\exists v_1 : y(v_1) \wedge n(v_1, v)}$ first order theorem prover

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Focus formulae

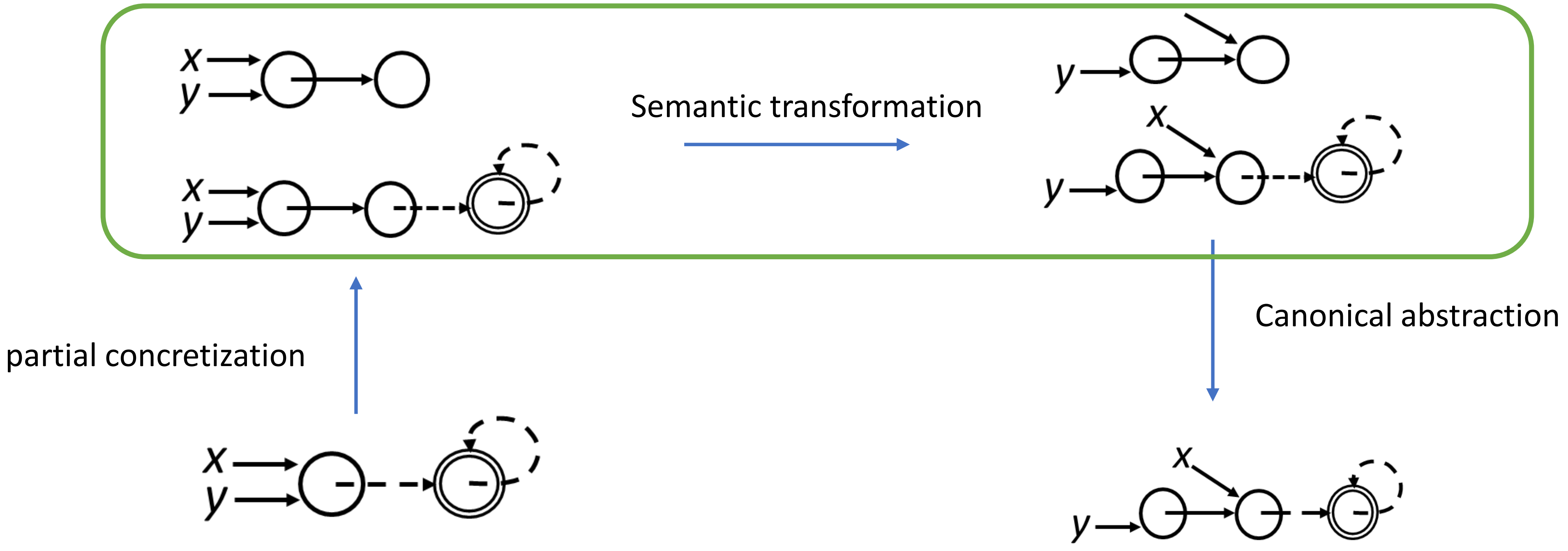


st	Focus Formulae
`x = NULL`	$\emptyset$
`x = t`	$\{t(v)\}$
`x = t->n`	$\{\exists v_1 : t(v_1) \wedge n(v_1, v)\}$
`x->n = t`	$\{x(v), t(v)\}$
`x = malloc()`	$\emptyset$
`x == NULL`	$\{x(v)\}$
`x != NULL`	$\{x(v)\}$
`x == t`	$\{x(v), t(v)\}$
`x != t`	$\{x(v), t(v)\}$
`UninterpretedCondition`	$\emptyset$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Formula update



Semantic transformation

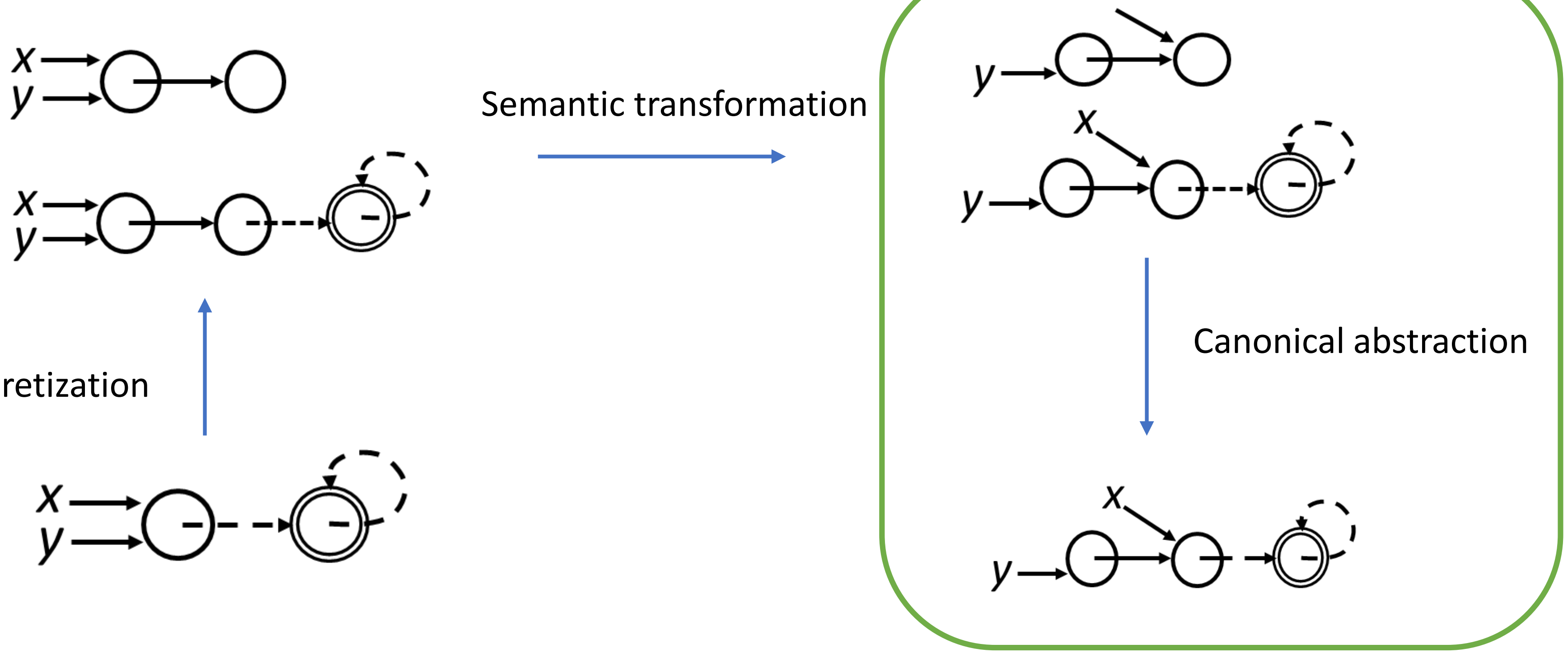
partial concretization

Canonical abstraction

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Recover the truth: Coerce



Semantic transformation

partial concretization

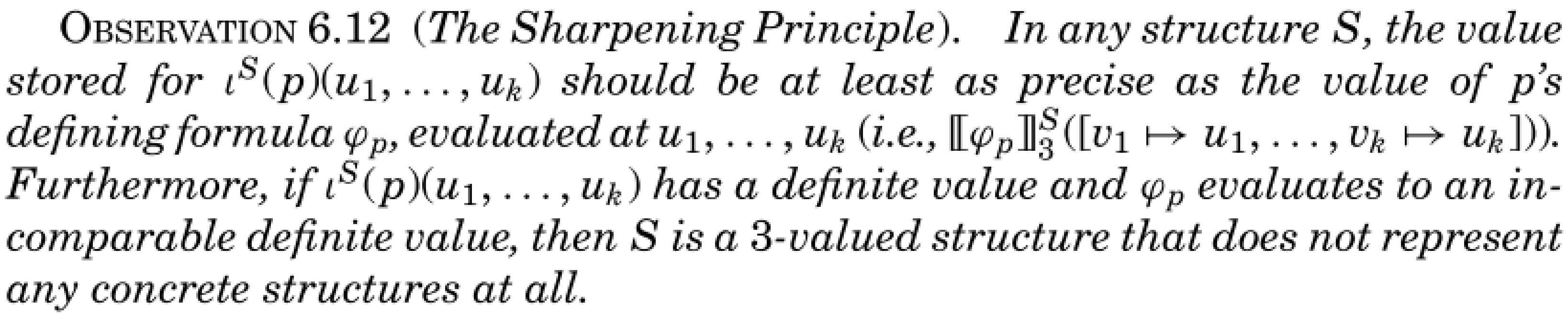
Canonical abstraction

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Remove the inconsistency

Formula Update

- Recover the truth: Coerce



OBSERVATION 6.12 (*The Sharpening Principle*). *In any structure S, the value stored for $\iota^S(p)(u_1, \ldots, u_k)$ should be at least as precise as the value of p's defining formula φ_p, evaluated at $u_1, \ldots, u_k$ (i.e., $[\![\varphi_p]\!]_3^S([v_1 \mapsto u_1, \ldots, v_k \mapsto u_k])$). Furthermore, if $\iota^S(p)(u_1, \ldots, u_k)$ has a definite value and φ_p evaluates to an incomparable definite value, then S is a 3-valued structure that does not represent any concrete structures at all.*

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

- Recover the truth: Coerce
  - There can be interdependences between different properties stored in a structure, and these interdependences are not necessarily incorporated in the definitions of the predicate-update formulae

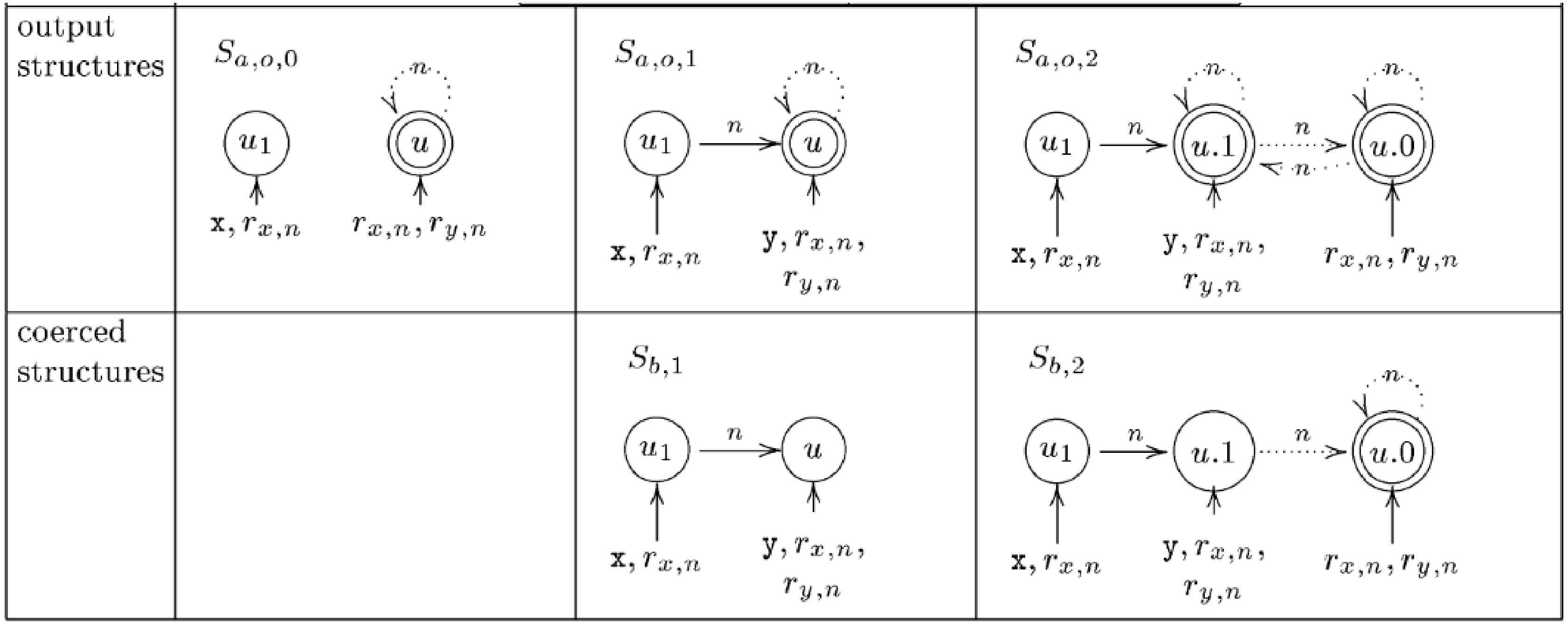
Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

Formula Update

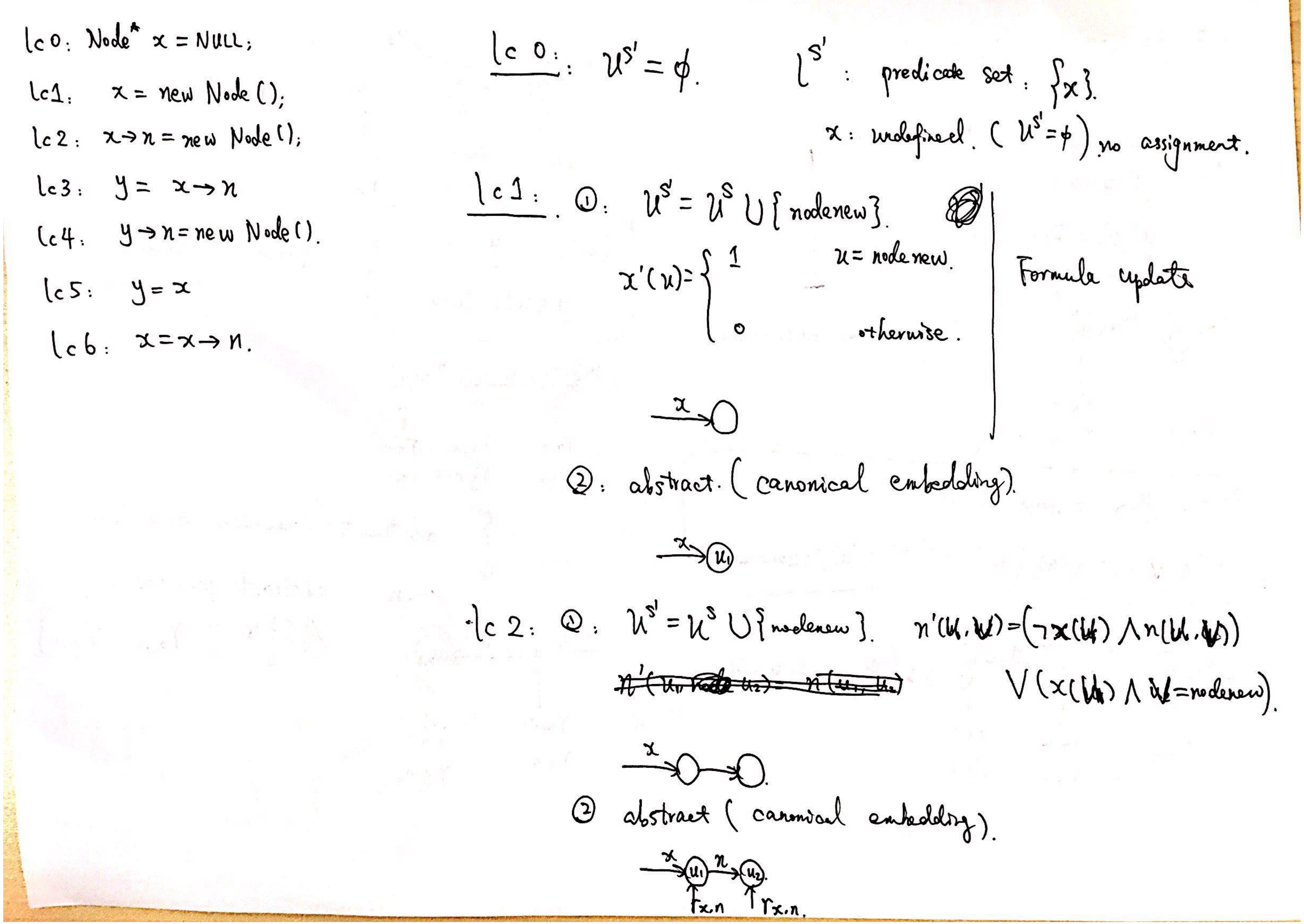
- An example $st_0 \colon y = y \to n$

y can not point to an individual representing one or more concrete individuals

first order theorem prover



Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002



```
lc0: Node* x = NULL;
lc1:   x = new Node();
lc2:  x->n = new Node();
lc3:  y = x->n
lc4:  y->n = new Node().
lc5:  y = x
lc6:  x = x->n.
```

lc 0: $U^{s'} = \phi$. $l^{s'}$: predicate set: $\{x\}$.

x: undefined. ($U^{s'} = \phi$) no assignment.

lc 1: ①: $U^{s'} = U^{s} \cup \{nodenew\}$.

$$x'(u) = \begin{cases} 1 & u = nodenew. \\ 0 & otherwise. \end{cases}$$

Formula update

$\xrightarrow{x} \bigcirc$

②: abstract. (canonical embedding).

$\xrightarrow{x} u_1$

lc 2: ①: $U^{s'} = U^{s} \cup \{nodenew\}$. $n'(v_1, v_2) = (\neg x(v_1) \wedge n(v_1, v_2))$

~~$n'(u_1, node\ u_2) = n(u_1, u_2)$~~

$\vee (x(v_1) \wedge v_2 = nodenew)$.

$\xrightarrow{x} \bigcirc \longrightarrow \bigcirc$.

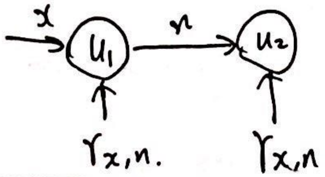
② abstract (canonical embedding).

$\xrightarrow{x} u_1 \xrightarrow{n} u_2$.

$r_{x,n}$ $r_{x,n}$.

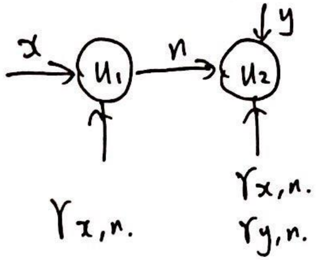
```
lc0: Node* x = NULL;
lc1:    x = new Node();
lc2:  x→n = new Node();
lc3:  y =  x→n
lc4:  y→n = new Node().
lc5:   y = x
lc6:  x = x→n.
```

~~lc 3: y = x→n.~~



lc3: $y = x \to n$.

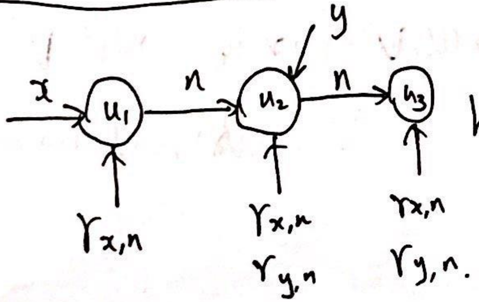
$y'(u) = \exists v,\ x(v) \wedge n(v,u)$.



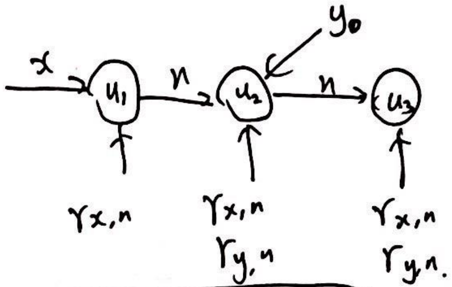
before/after abstraction.

lc 4: $y \to n = new\ Node()$.

$n'(u,v) = (\neg x(u) \wedge n(u,v)) \vee (x(u) \wedge v = nodenew)$



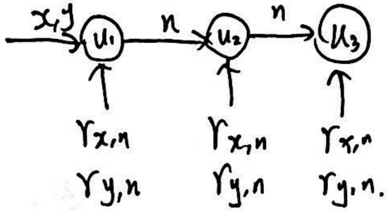
before/after abstraction.



lc5: $y = x$.

$y'(u) = x(u)$.

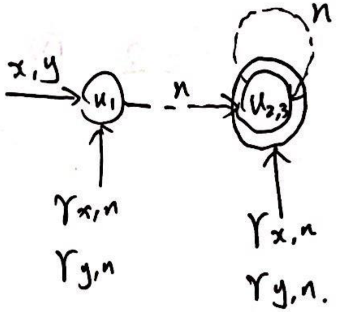
formula update.



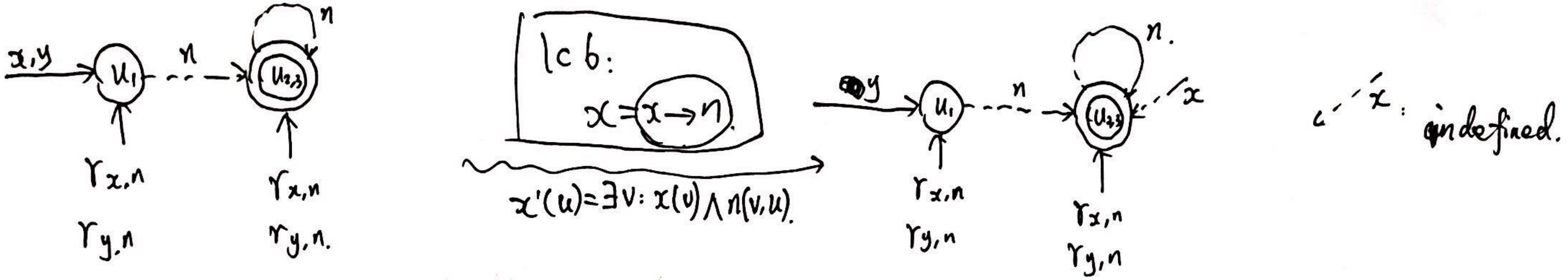
abstract (canonical embedding).

abstract predicates.

$A = \{x, y, r_{x,n}, r_{y,n}\}$.

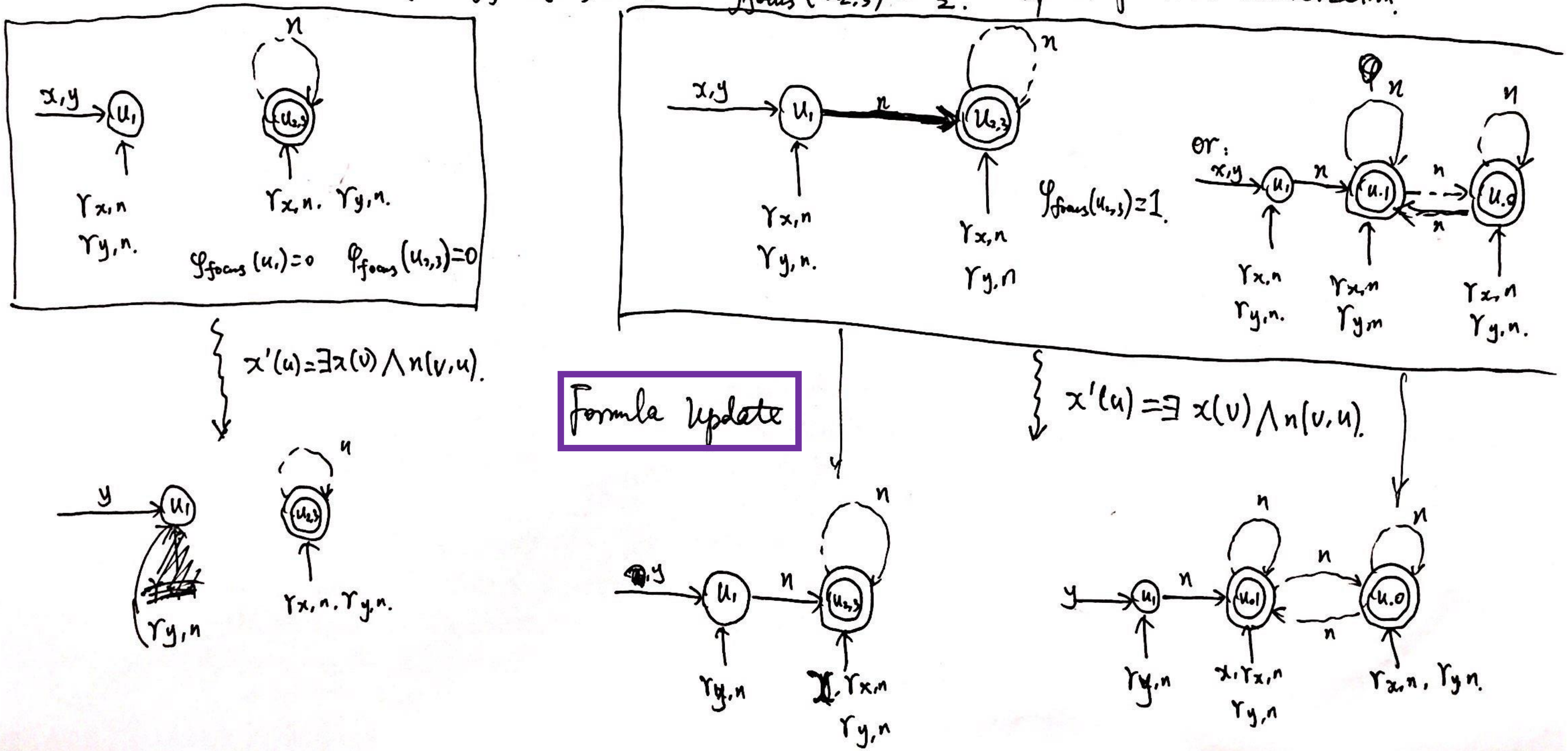


```
lc0: Node* x = NULL;
lc1:    x = new Node();
lc2:  x->n = new Node();
lc3:   y =  x->n
lc4:   y->n = new Node().
lc5:    y = x
lc6:   x = x->n.
```

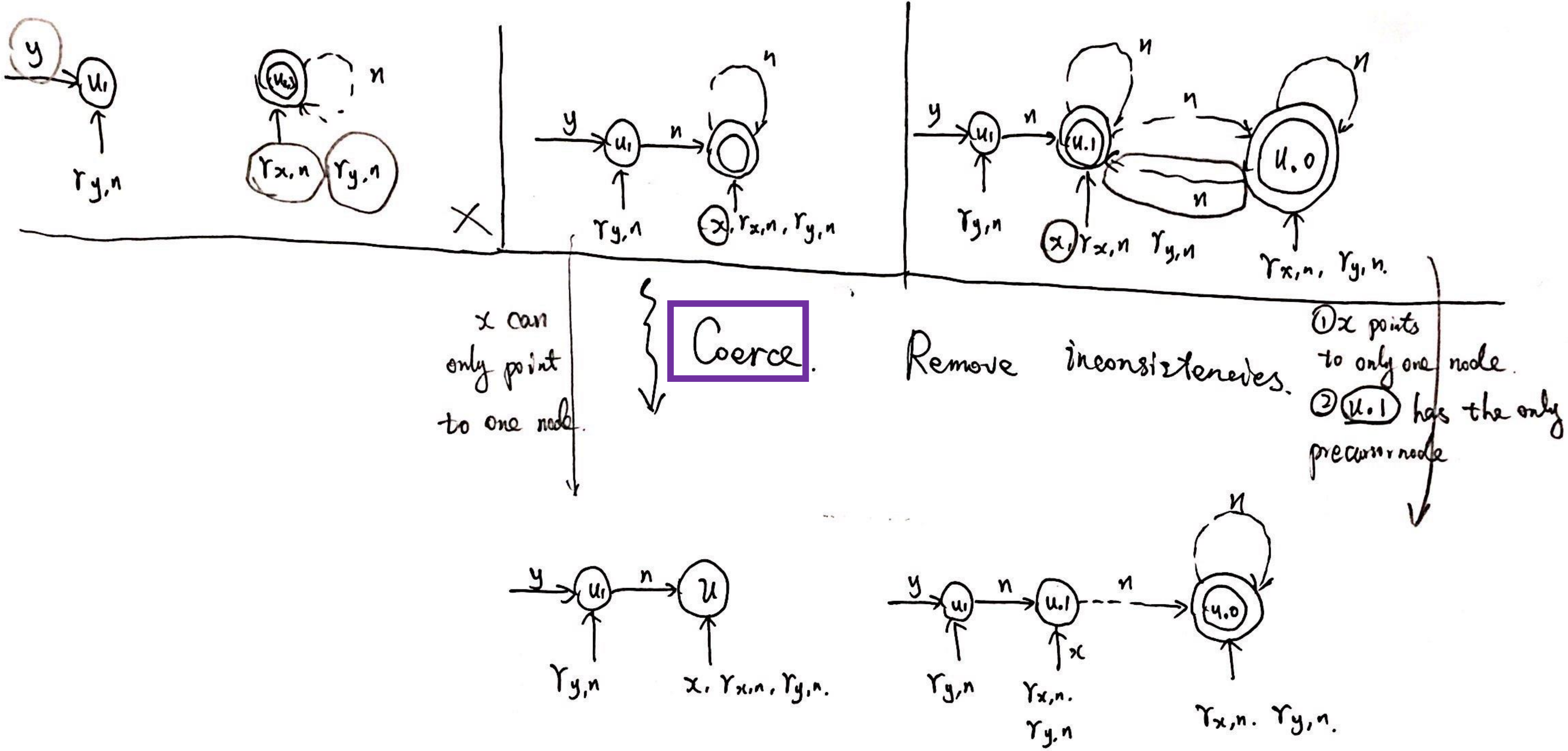


Focus Operation: $\varphi_{focus}(u) = \exists v.\ x(v) \wedge n(v,u)$. should be a defined value (0, 1)

Example: $\varphi_{focus}(u_1) = 0$. $\varphi_{focus}(u_{2,3}) = \frac{1}{2}$. split, partial concretization.



```
lc0: Node* x = NULL;
lc1:   x = new Node();
lc2:  x→n = new Node();
lc3:  y =  x→n
lc4:  y→n = new Node().
lc5:   y = x
lc6:  x = x→n.
```



x can only point to one node.

Coerce.

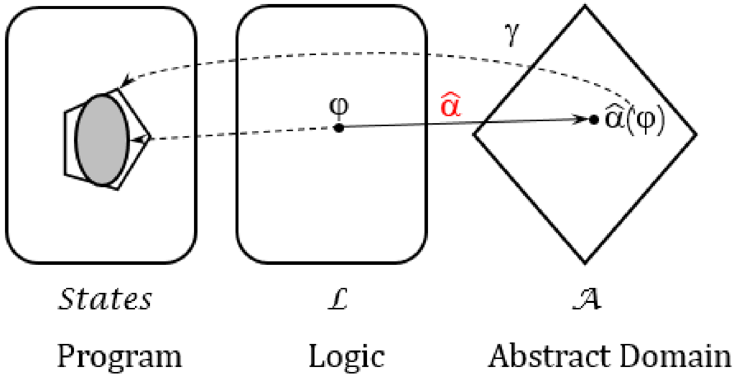
Remove inconsistencies.

① x points to only one node.

② u.1 has the only precursor node

Discussion

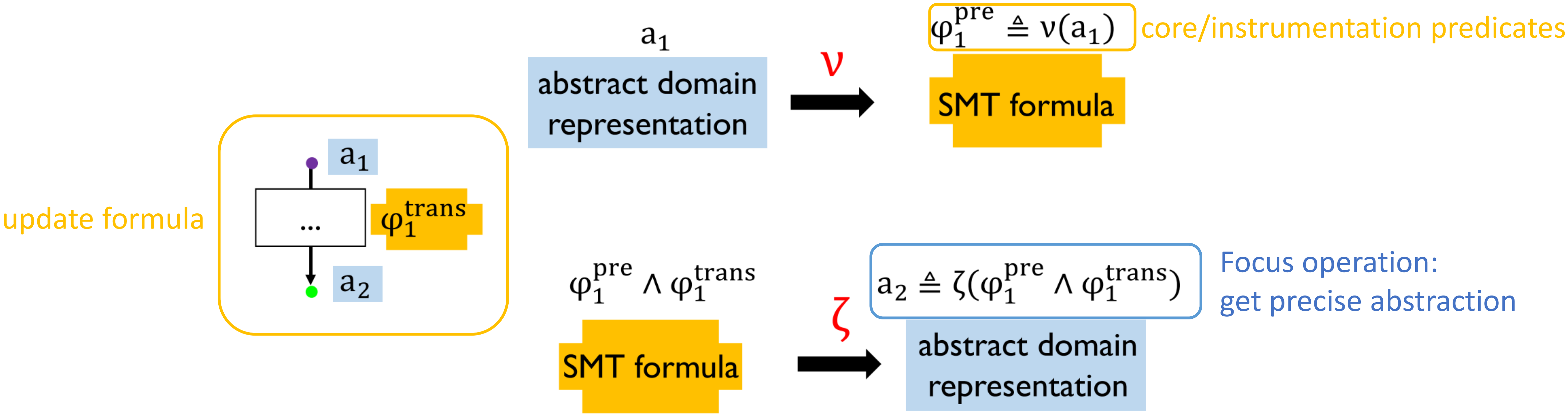
- Focus operation is an instance of **precise abstract operation in symbolic abstraction**
- Logic formula is a unified way to describe different kinds of abstract domains
  - Guide the refinement of abstract state after transformation to obtain best abstraction



Yorsh G, Reps T, Sagiv M. **Symbolically computing most-precise abstract operations for shape analysis**[C] TACAS 2004

Discussion

- Logic is rigorous and powerful
  - Another recent work: combine abstract domain with SMT



a_1

abstract domain representation

$\xrightarrow{\nu}$

$\boxed{\varphi_1^{pre} \triangleq \nu(a_1)}$ core/instrumentation predicates

SMT formula

update formula

a_1

...

φ_1^{trans}

a_2

$\varphi_1^{pre} \wedge \varphi_1^{trans}$

SMT formula

$\xrightarrow{\zeta}$

$\boxed{a_2 \triangleq \zeta(\varphi_1^{pre} \wedge \varphi_1^{trans})}$

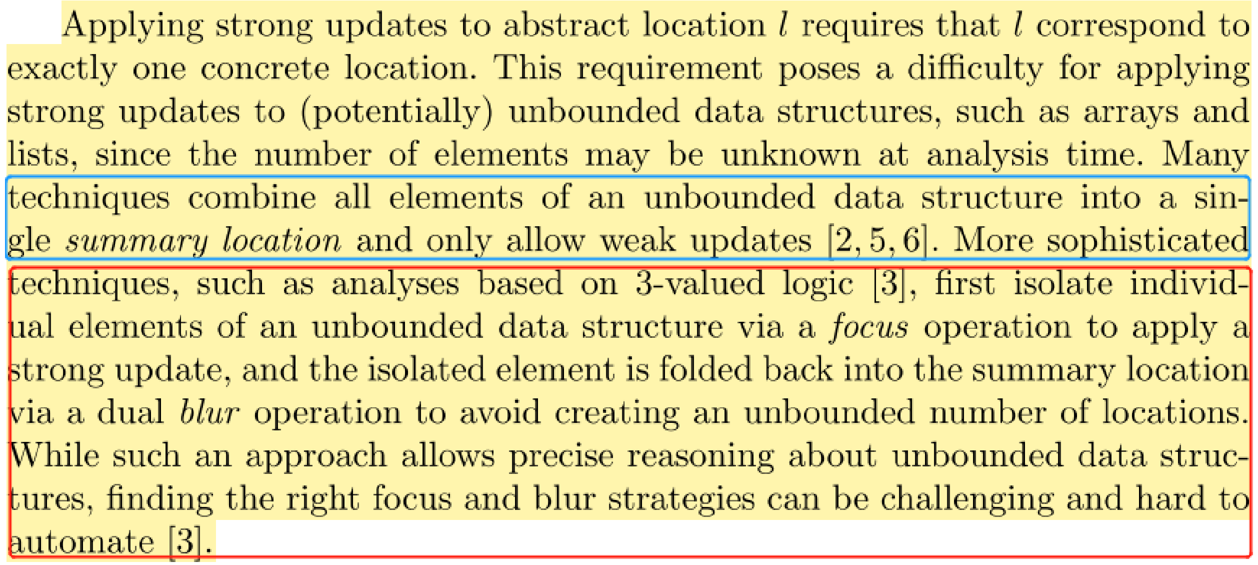
abstract domain representation

Focus operation:
get precise abstraction

Jiang J, Chen L, Wu X, et al. **Block-Wise Abstract Interpretation by Combining Abstract Domains with SMT**[C] VMCAI 2017

Discussion

- Another view: Strong Updates
  - Comments by Isil Dillig



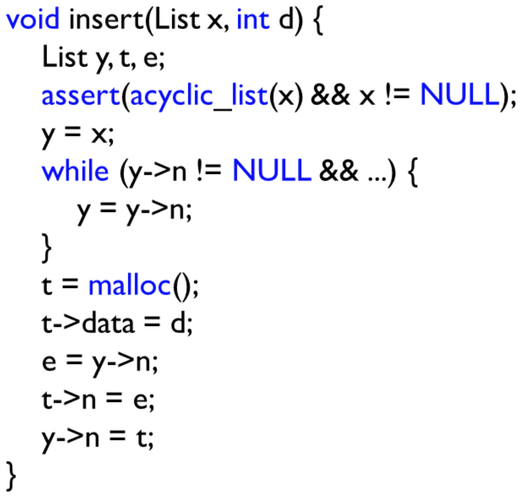
Applying strong updates to abstract location l requires that l correspond to exactly one concrete location. This requirement poses a difficulty for applying strong updates to (potentially) unbounded data structures, such as arrays and lists, since the number of elements may be unknown at analysis time. Many techniques combine all elements of an unbounded data structure into a single *summary location* and only allow weak updates [2, 5, 6]. More sophisticated techniques, such as analyses based on 3-valued logic [3], first isolate individual elements of an unbounded data structure via a *focus* operation to apply a strong update, and the isolated element is folded back into the summary location via a dual *blur* operation to avoid creating an unbounded number of locations. While such an approach allows precise reasoning about unbounded data structures, finding the right focus and blur strategies can be challenging and hard to automate [3].

Dillig I, Dillig T, Aiken A. **Fluid updates: Beyond strong vs. weak updates**[C] ESOP 2010

More Discussions

- How like in Anderson analysis ? predicates transfer for load & store statement ? Does shape analysis has transfer rule for load/store statement

TVLA is a framework of program verification and do not handle load/store statement.

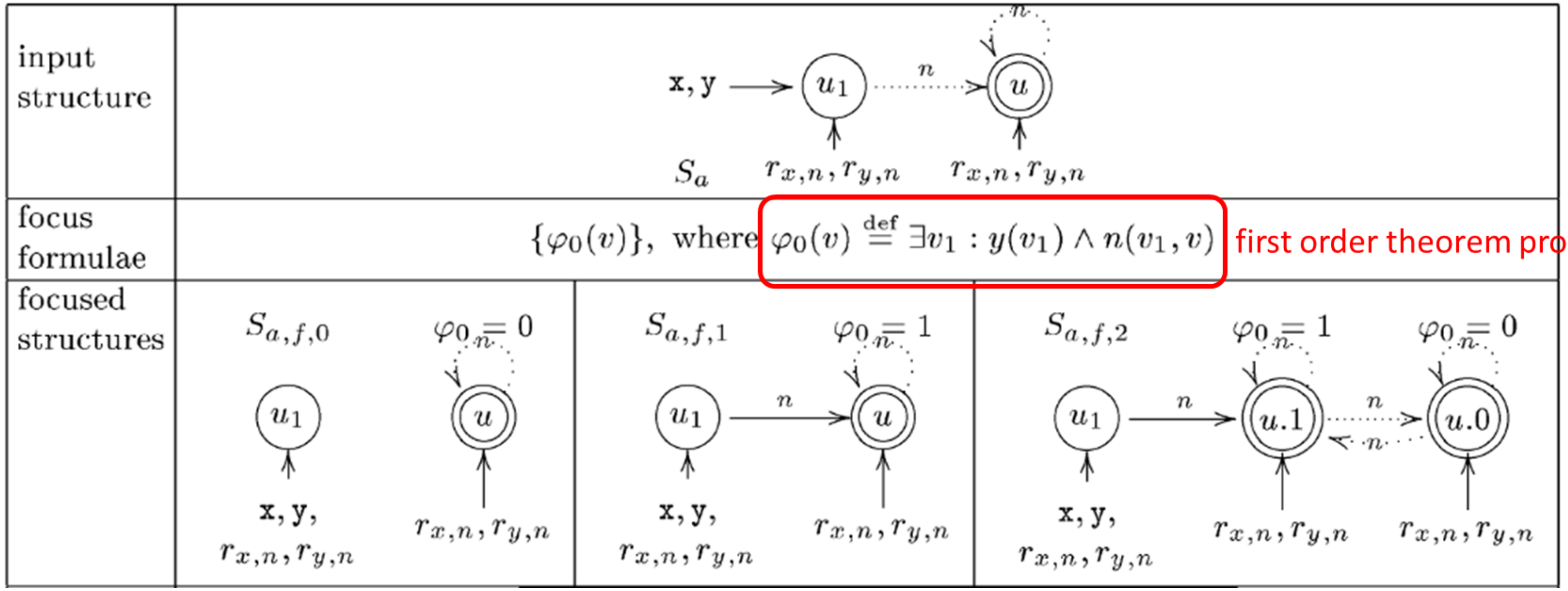


```
void insert(List x, int d) {
    List y, t, e;
    assert(acyclic_list(x) && x != NULL);
    y = x;
    while (y->n != NULL && ...) {
        y = y->n;
    }
    t = malloc();
    t->data = d;
    e = y->n;
    t->n = e;
    y->n = t;
}
```

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

More Discussions

- Why is TVLA unscalable?
    - Focus and Coerce need a large amounts of constraint solving



$\{\varphi_0(v)\}$, where $\varphi_0(v) \overset{\text{def}}{=} \exists v_1 : y(v_1) \wedge n(v_1, v)$ first order theorem prover

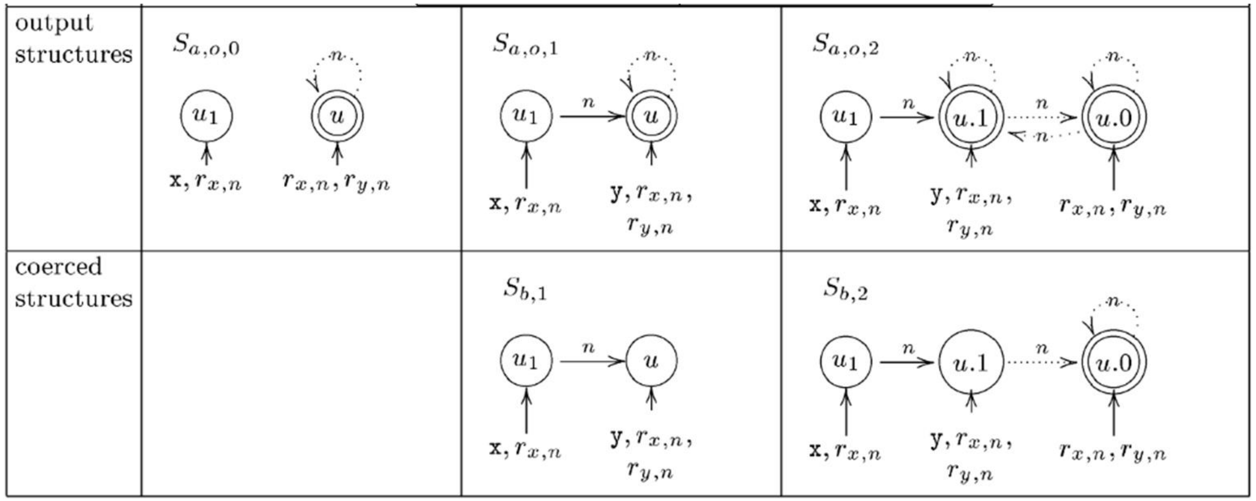
Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

More Discussions

- Why is TVLA unscalable?
  - Focus and Coerce need a large amounts of constraint solving

y can not point to an individual representing
one or more concrete individuals

first order theorem prover



Remove the inconsistency

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

More Discussions

- Why is TVLA unscalable?
  - Focus construct explicit partitions of unbounded structures, and generate new logical structures, which causes state space explosion problem.

$$\begin{tikzcd}
\{S_a\} \arrow[rrr, "focus"] \arrow[d, "{[\![st_0]\!]_3}"'] & & & \{S_{a,f,0}, S_{a,f,1}, S_{a,f,2}\} \arrow[d, "{[\![st_0]\!]_3}"] \\
\{S_b\} & \sqsupseteq & \{S_{b,1}, S_{b,2}\} & \{S_{a,o,0}, S_{a,o,1}, S_{a,o,2}\} \arrow[l, "coerce"']
\end{tikzcd}$$

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

More Discussions

- What' s the application of shape analysis in static bug detection
  - Detect list manipulation bugs
  - Will be discussed later

Dor N, Rodeh M, Sagiv M. **Checking cleanness in linked lists**, ISSTA 2000: 115-134.

More Discussions

- How to choose appropriate abstraction predicates ?
- What happens if the program contains a bug (e.g a=NULL; the statement is a->next = n; Then how should we transfer predicate values) ?
- What' s the application of shape analysis in static bug detection (Any papers in this area?) ? (Why Facebook Infer scales ?)
- How like in Anderson analysis ? predicates transfer for load & store statement ? Does shape analysis has transfer rule for load/store statement

Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. TOPLAS 2002

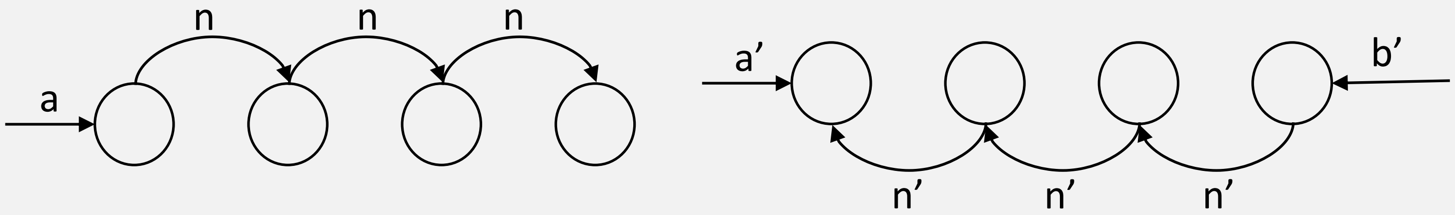
TVLA II: Interprocedural

- Two main approaches to interprocedural static analysis
    - Functional
    - Operational
- Functional approach
    - Compute procedure summaries
    - Obtain the best transformer

Reps T, Sagiv M, Yorsh G**. Symbolic implementation of the best transformer**[C] VMCAI 2004

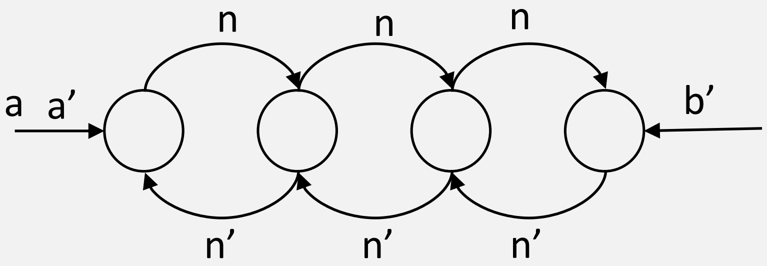
Transformer Abstraction

- An example: Transformer of list *reverse* function



Pair of one-vocabulary structures

✓

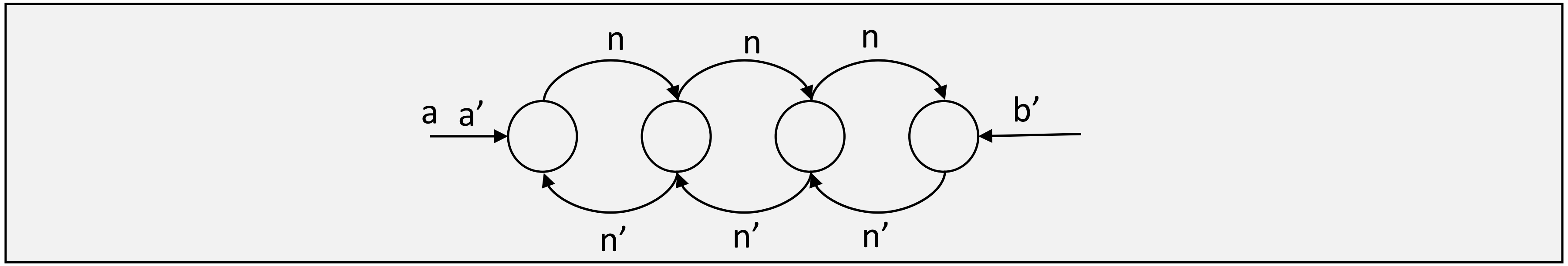


Pair of one-vocabulary structures

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Abstraction

- Canonical abstraction of transformer



- Two-vocabulary structure: a logical structure
- Based on abstraction predicates

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Abstraction

- Relation Instrumentation Predicate
  - Defined in specific verification problem
  - Establish the connection between *input and output*

$$\begin{aligned} id\_succ[n, m_1, m_2](v) &= \forall v_1 : (n[m_1](v, v_1) \Rightarrow n[m_2](v, v_1)) \\ id\_pred[n, m_1, m_2](v) &= \forall v_1 : (n[m_1](v_1, v) \Rightarrow n[m_2](v_1, v)). \end{aligned}$$

$\boldsymbol{m_1 \ \ m_2 \in \{inp, out, tmp\}}$

- *Additional constraint rule*

$$id\_succ[n, m_1, m_2](v) \wedge id\_succ[n, m_2, m_3](v) \Rightarrow id\_succ[n, m_1, m_3](v)$$

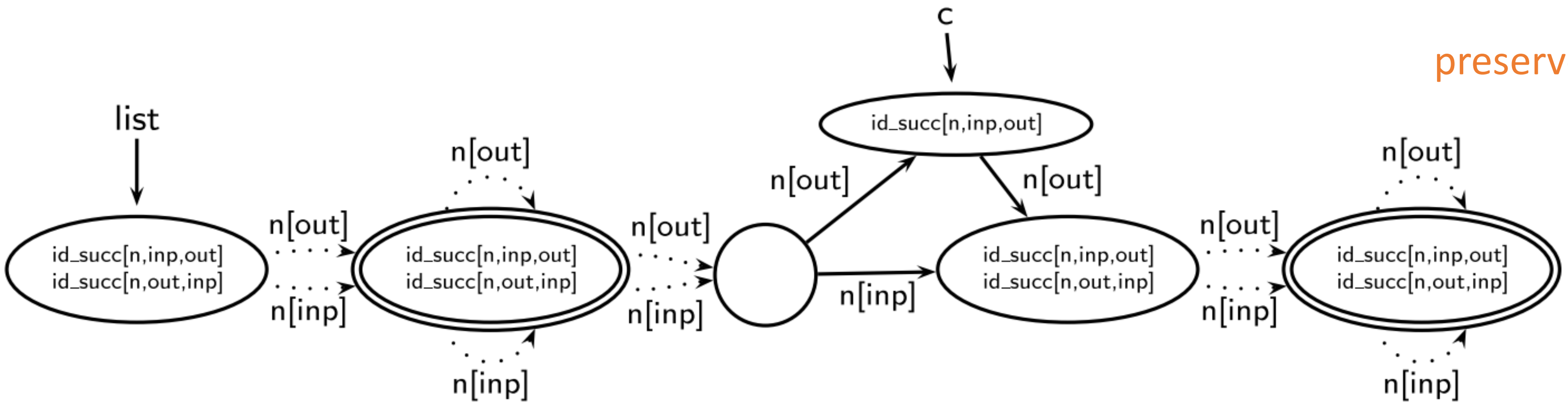
Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Abstraction

- An example: insert

$$
\begin{aligned}
id\_succ[n, m_1, m_2](v) &= \forall v_1 : (n[m_1](v, v_1) \Rightarrow n[m_2](v, v_1)) \\
id\_pred[n, m_1, m_2](v) &= \forall v_1 : (n[m_1](v_1, v) \Rightarrow n[m_2](v_1, v)).
\end{aligned}
$$

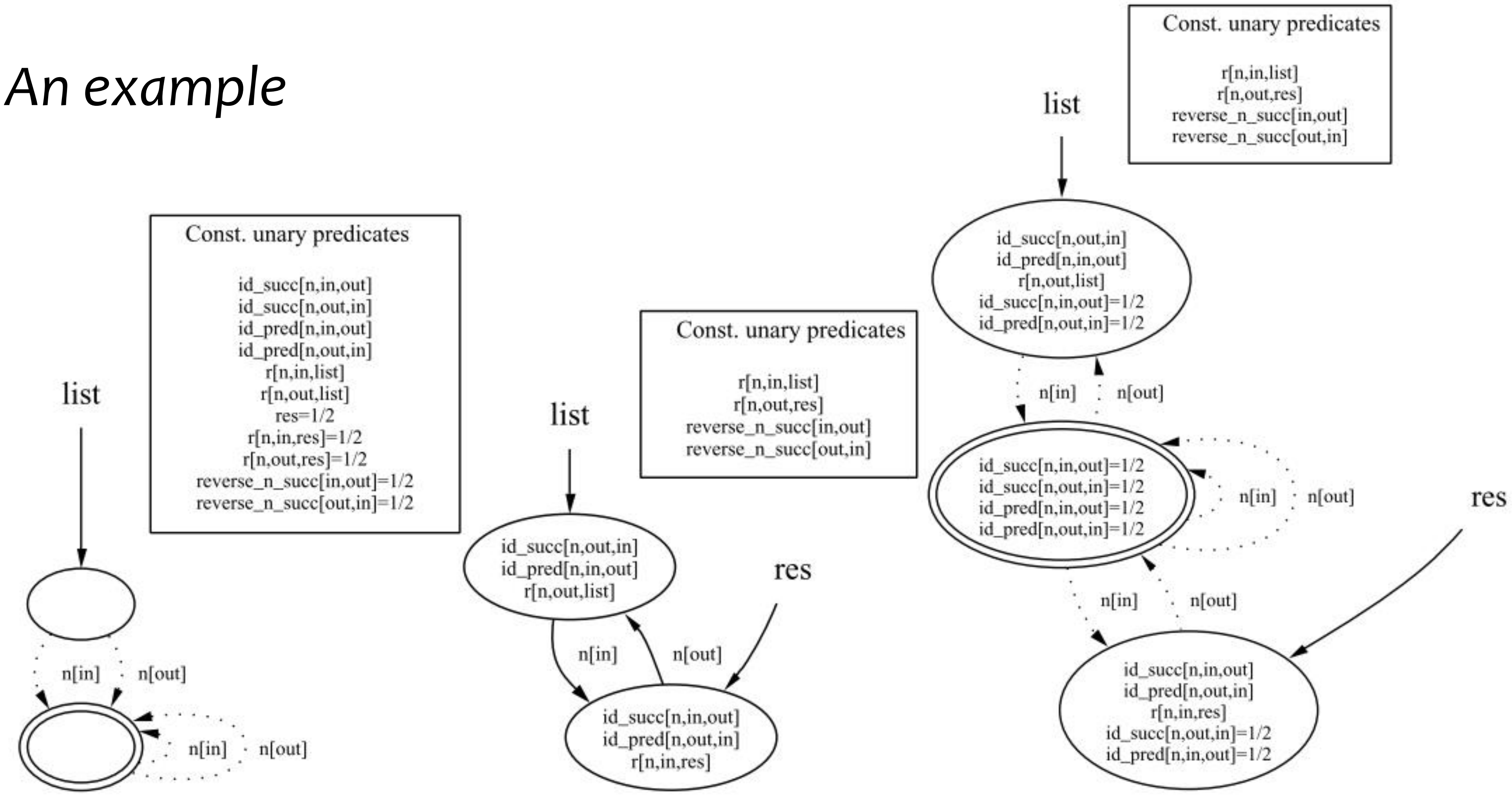
$m_1 \ \ m_2 \in \{inp, out, tmp\}$



preserve the order

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

- *An example*



(a) S_0 (b) S_1 (c) S_2

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Operations

- Relational approach to model function call and return
  - Combine two abstract transformer
  - Obtain a precise combined transformer
- How to combine
  - Two transformer operations
    - Composition
    - Meet

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Operations

- Composition
  - No local variable and function parameters
    - $T_2 \circ T_1 \overset{\text{def}}{=} (T_1[tmp \leftarrow out; out \leftarrow 1/2] \sqcap T_2[tmp \leftarrow in; in \leftarrow 1/2])[tmp \leftarrow 1/2]$
    - tmp bridges the postcondition of T_1 and the precondition of T_2
    - in and out are exposed interfaces of $T_2 \circ T_1$
    - $\sqcap$ captures the (largest) common parts of two logical structures

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Operations

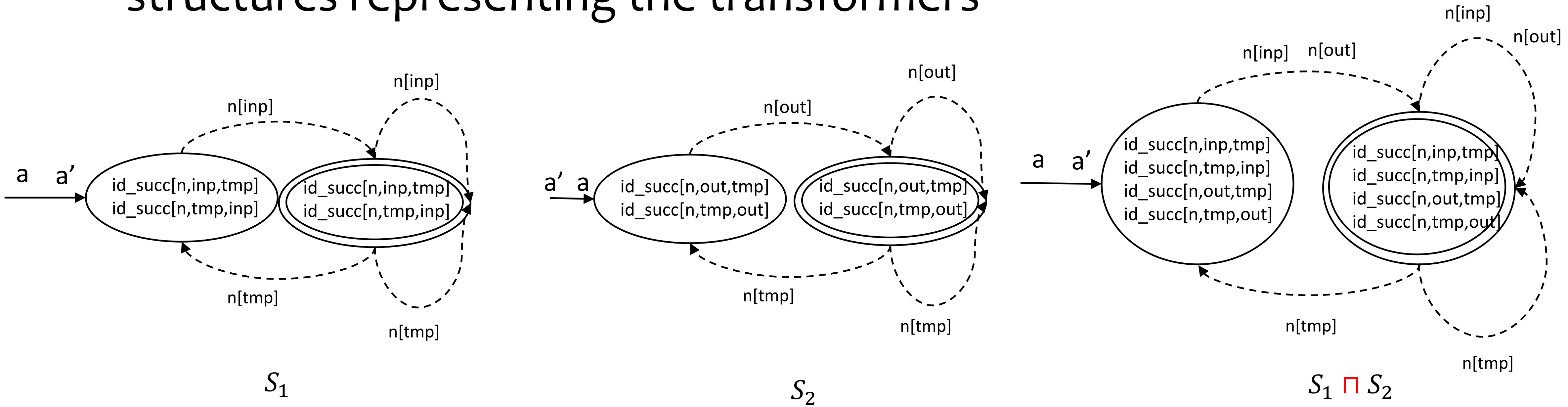
- Composition
  - No local variable and function parameters
    - $T_2 \circ T_1 \overset{\text{def}}{=} (T_1[tmp \leftarrow out; out \leftarrow 1/2] \sqcap T_2[tmp \leftarrow in; in \leftarrow 1/2])[tmp \leftarrow 1/2]$
  - Local variables and functions parameters

$$T_2 \circ T_1 \overset{\text{def}}{=} \left(\tau_{y:=\mathsf{fpo}} \circ \left(\begin{array}{l} (\tau_{\mathsf{fpi}:=x} \circ T_1)[tmp \leftarrow out; out \leftarrow 1/2] \\ \sqcap \\ (\tau_{\substack{\mathsf{fpi}:=\mathsf{fpi}_q; \\ \mathsf{fpo}:=\mathsf{fpo}_q}} \circ T_2) \begin{bmatrix} tmp \leftarrow in; in \leftarrow 1/2; \\ loc \leftarrow 1/2 \end{bmatrix} \end{array} \right) \right) \begin{bmatrix} tmp \leftarrow 1/2; \\ \{\mathsf{fpi}, \mathsf{fpo}\} \leftarrow 1/2 \end{bmatrix}$$

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Transformer Operations

- Meet ⊓ : get the greatest lower-bound operation of two logical structures representing the transformers



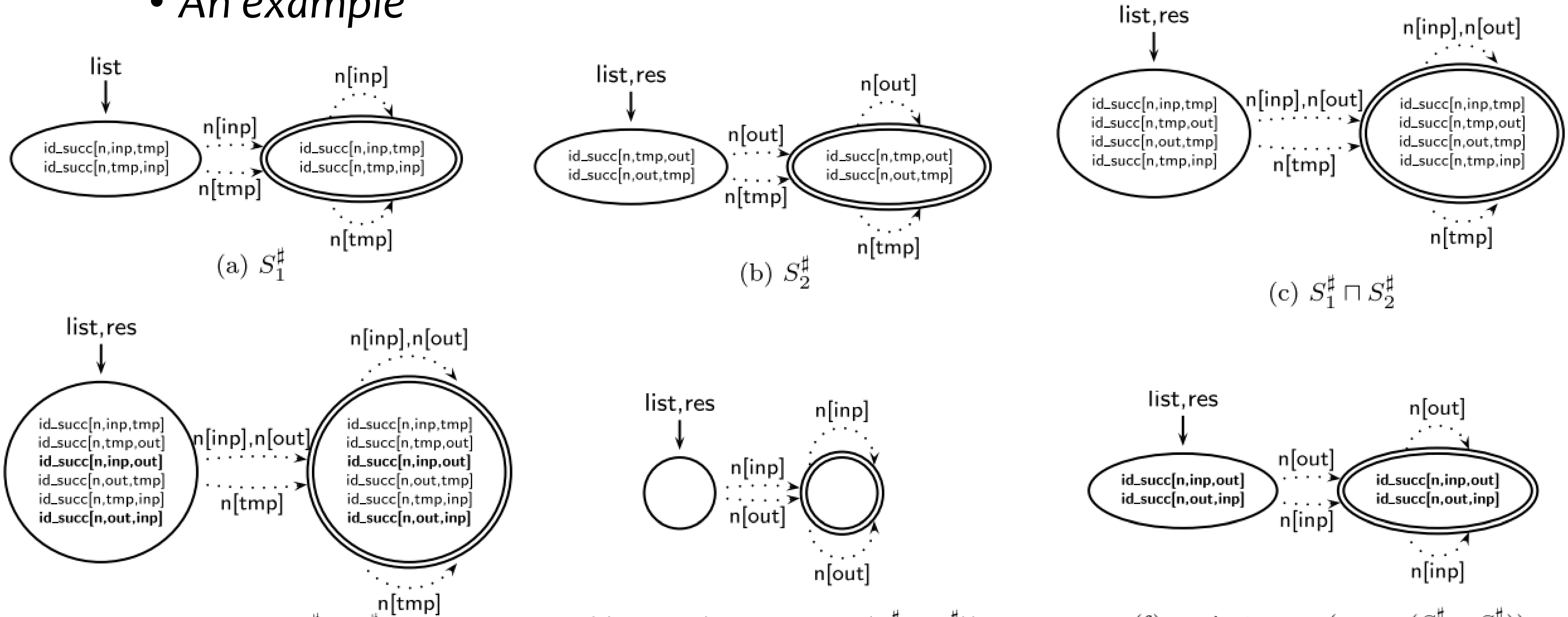
S_1

S_2

$S_1 \sqcap S_2$

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

- *An example*



(a) $S_1^\sharp$

(b) $S_2^\sharp$

(c) $S_1^\sharp \sqcap S_2^\sharp$

(d) $\mathrm{coerce}(S_1^\sharp \sqcap S_2^\sharp)$

(e) $\mathrm{coerce}(\mathrm{project}_{inp,out}(S_1^\sharp \sqcap S_2^\sharp))$

(f) $\mathrm{project}_{inp,out}(\mathrm{coerce}(S_1^\sharp \sqcap S_2^\sharp))$

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

TVLA II: Summary

- Relational Interprocedural shape analysis
  - Functional summary based approach
  - Best transformer of function

Reps T, Sagiv M, Yorsh G**. Symbolic implementation of the best transformer**[C] VMCAI 2004

Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C] TOPLAS 2004

Numerical Domain Summarization

- TVLA handles the unbounded dynamically allocated data connected by the pointers and pointer fields
- How to obtain the properties of numerical fields in dynamically allocated data
  - The set of numerical fields of all the nodes is unbounded
  - Conventional numerical domain, including boxes, octagons, and polyhedra, can only handle finite number of numerical variables

Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C] TACAS 2004

Numerical Domain Summarization

- An motivating example

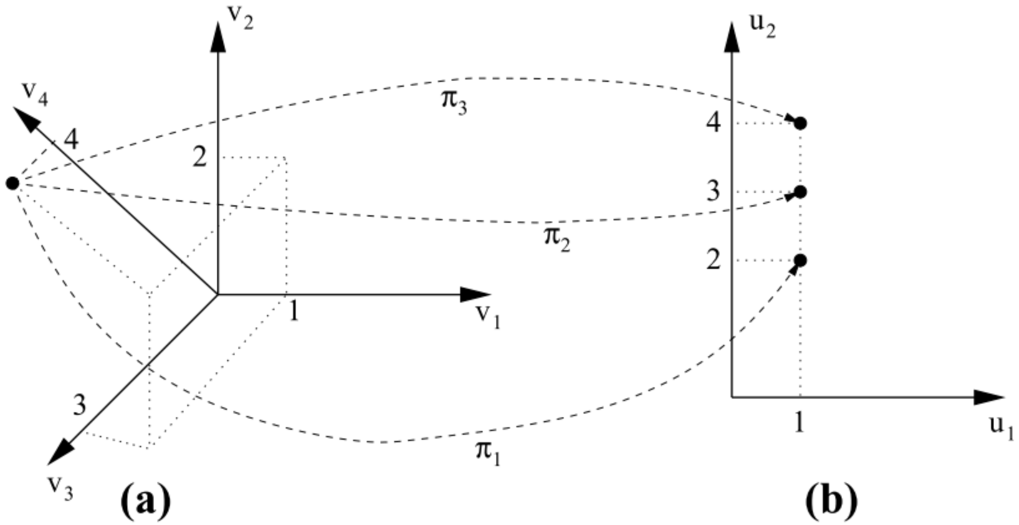
```
public static Stack<Integer> generateRandomIntStack(int n) {
    Stack<Integer> s = new Stack<Integer>();
    for (int i = 0; i < n; i++) {
        s.push(new Integer((int)Math.random()));
    }
    ...
    return s;
}
```

- Abstraction is necessary

Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C] TACAS 2004

Numerical Domain Summarization

- Embedding for Abstraction



$$\begin{tikzcd}
(v_1, v_2, v_3, v_4) = (1,2,3,4) \arrow[d, "\pi"] \\
(u_1, u_2) = \{(1,2), (1,3), (1,4)\} \arrow[d, "\text{over-approximation}"] \\
(u_1, u_2) \in [1,1] \times [2,4]
\end{tikzcd}$$

Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C] TACAS 2004

Numerical Domain Summarization

- Embedding for Abstraction
  - Linear space decomposition
    - $V^4 = span(x_1) + span(x_2, x_3, x_4)$
    - $v_1 \in span(x_1) \quad (v_2, v_3, v_4) \in span(x_2, x_3, x_4)$
  - Subspace approximation
    - Find abstract domains to over-approximate the point set in each subspace
  - Connections between two subspace
    - Numerical variables in two subspace respectively might have correlations

$$\begin{tikzcd} (v_1, v_2, v_3, v_4) = (1,2,3,4) \arrow[d, "\pi"] \\ (u_1, u_2) = \{(1,2), (1,3), (1,4)\} \arrow[d, "\text{over-approximation}"] \\ (u_1, u_2) \in [1,1] \times [2,4] \end{tikzcd}$$

Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C] TACAS 2004

Numerical Domain Summarization

- How to describe the correlations among numerical variables
  - Inner subspace: conventional numerical domain
  - Cross subspace: Not described in the paper(future work)
- Several Open Problems
  - How to decompose infinite dimensional space
    - Clustering
  - How to over-approximate each subspace
    - Compositional numerical domain

Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C] TACAS 2004

Jeannet B, Miné A. **Apron: A library of numerical abstract domains for static analysis**[C] CAV 2009

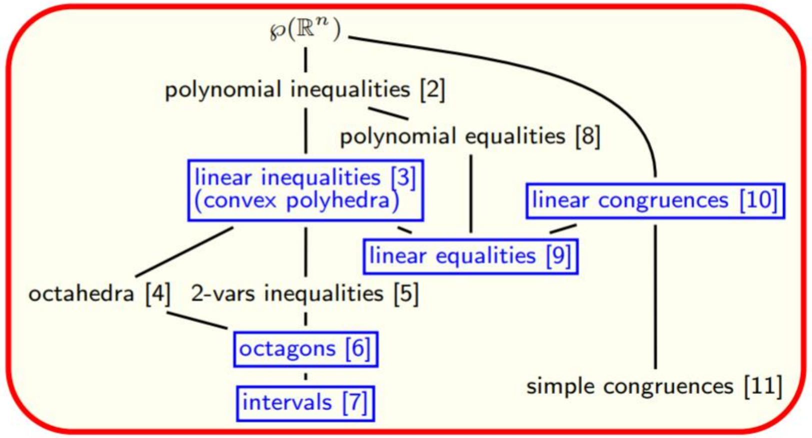


Fig. 2. Some abstract domains for numerical variables, partially ordered w.r.t. their expressiveness.

Application

- Concurrent Program Verification
  - Verifying safety properties of concurrent Java programs using 3-valued logic
  - Verifying safety properties using separation and heterogeneous abstractions
- List Manipulation Bug Detection
  - Checking cleanness in linked lists
  - Putting static analysis to work for verification: A case study
- Memory Management
  - Establishing local temporal heap safety properties with applications to compile-time memory management
- Testcase Generation
  - Generating Concrete Counterexamples for Sound Abstract Interpretation

Concurrent Program Verification

- Want to verify and detect:
  - Deadlock
    - Reach a deadlock?
  - Thread state errors
    - Read-Write Race(RW)
    - Write-Write Race(WW)
- Challenge
  - Unbounded data structure
  - Unbounded number of threads

```
class Main {
  public static void main (String[] args) {
    Queue q = new Queue();
    Thread prd = new Thread(new Producer(q));
    Thread cns = new Thread(new Consumer(q));
    prd.start();
    cns.start();
  }
}
```

Example: Java concurrent program. Producers add elements to Queue, Consumers remove elements from Queue

Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

- Insight
  - Abstract all the unbounded “objects” in canonical embedded 3-valued logic structures
  - Unbounded “objects” include threads, locks, and dynamically allocated data structures
- Predicate
  - *is_thread(t)*: *t* is a thread
  - *blocked(t, l)*: the thread *t* is blocked on the lock *l*
  - *held_by(l, t)*: the lock *l* is held by the thread *t*
  - *rvalue[fld](o1, o2)*: field *fld* of the object *o1* points to the object *o2*
  - *at[lab](t)*: thread *t* is at label *lab*(*lab* is a program location)
  - *Is_waiting(t), is blocked(t), wait_for(t1, t2)…*

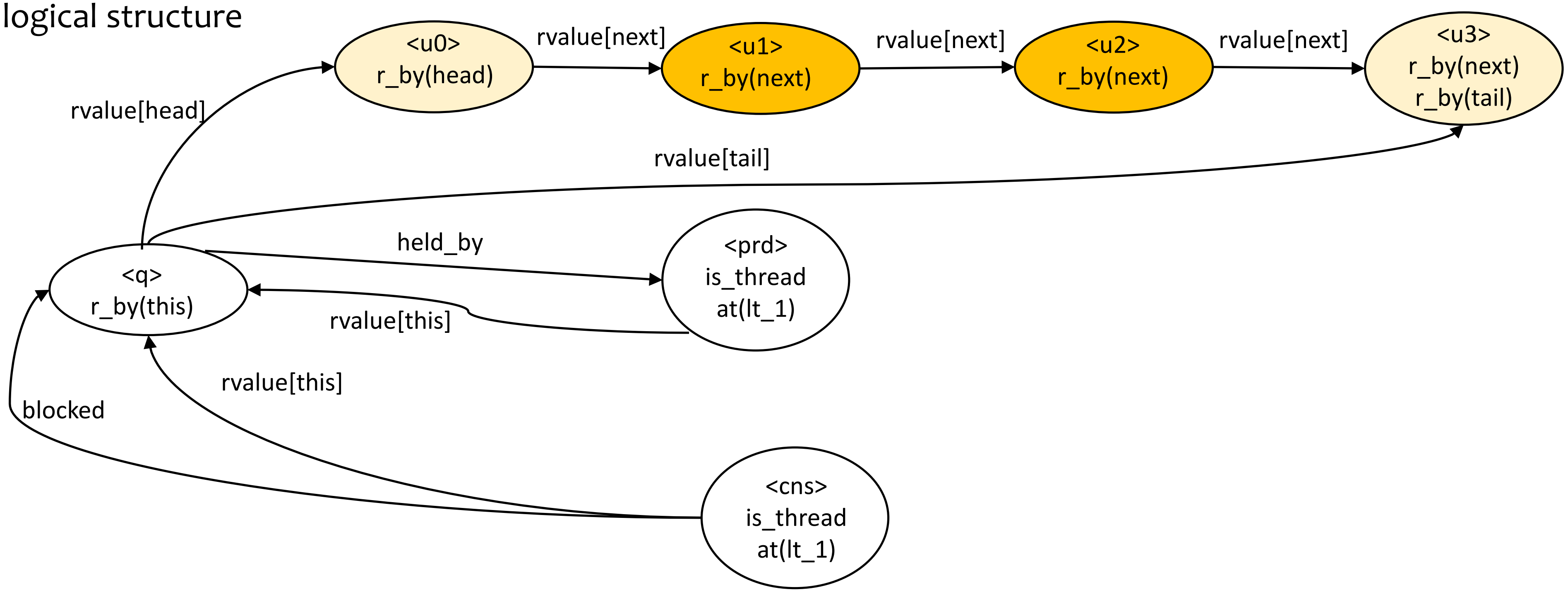
$\exists t\colon is\_thread(t) \land held\_by(l,t)$

lock l is acquired by some thread

Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

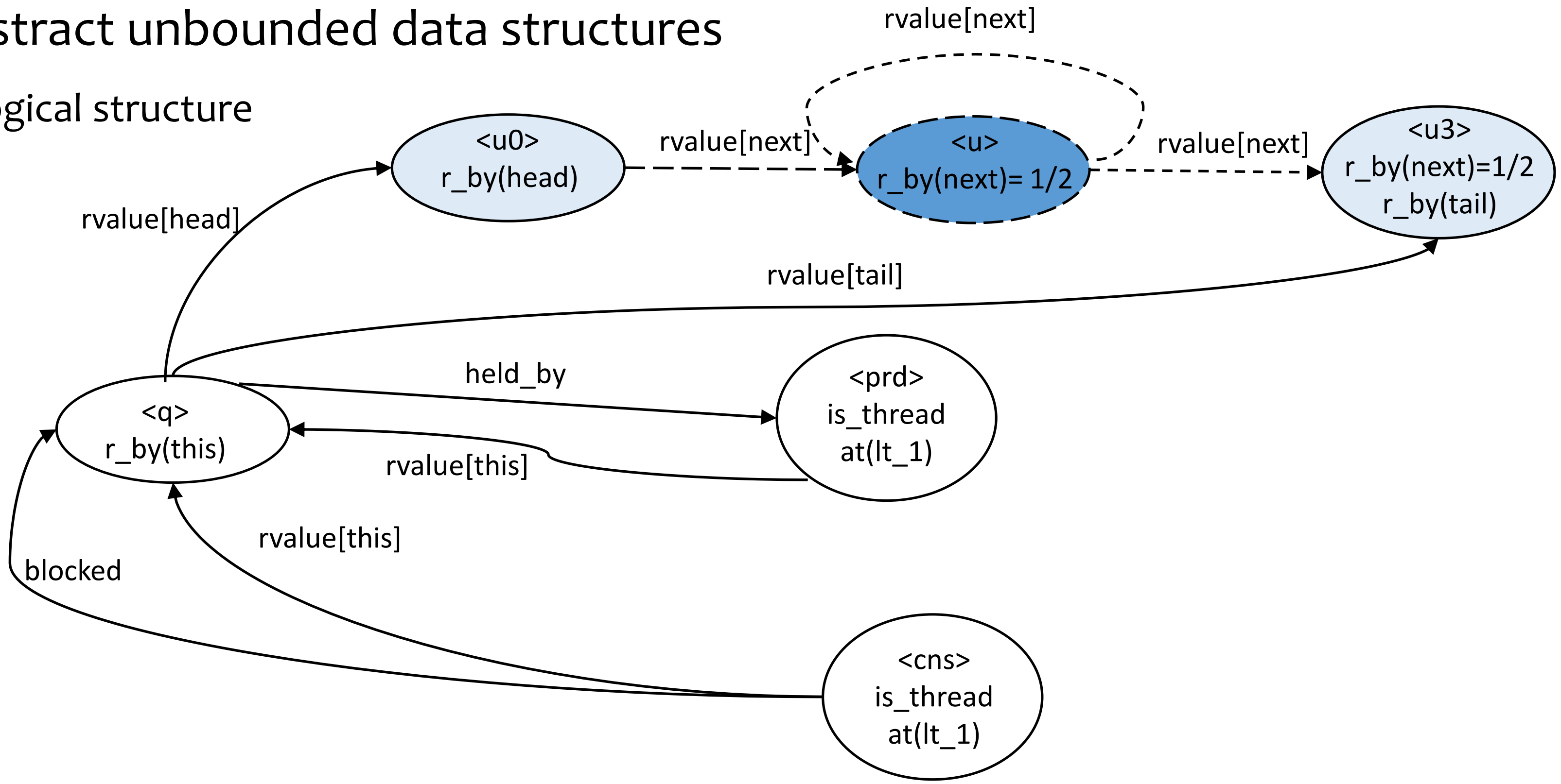
- Example: abstract unbounded data structures
  - 2-valued logical structure



Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

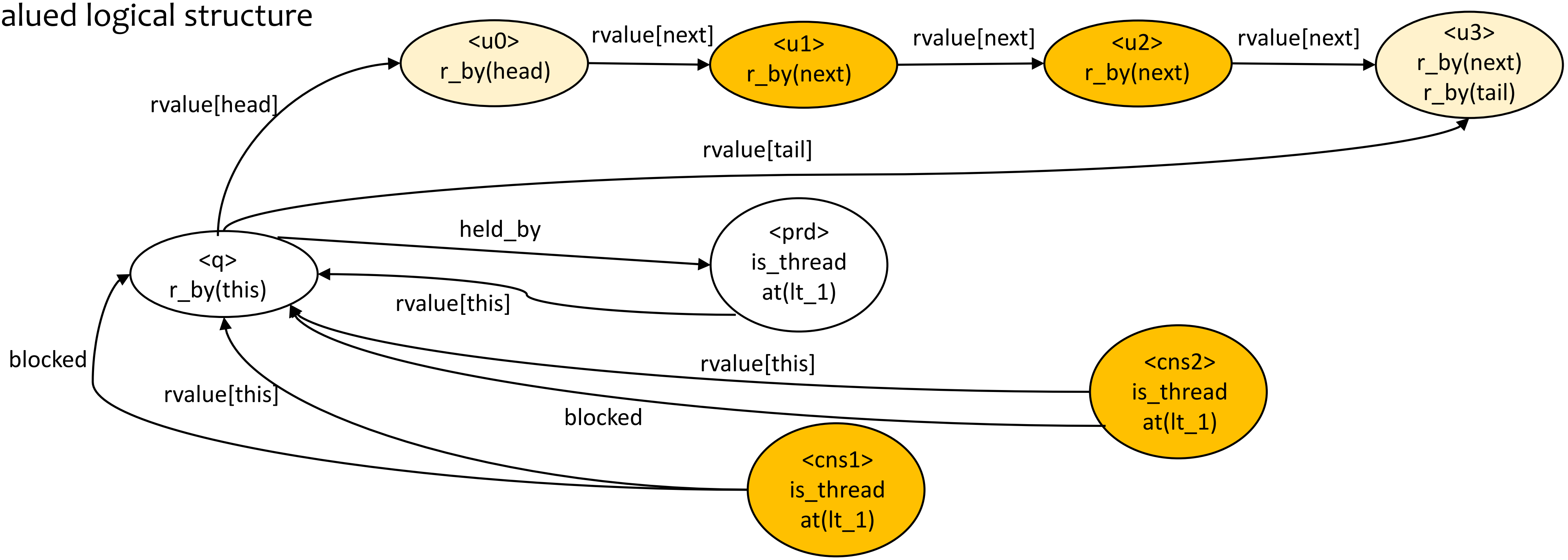
- Example: abstract unbounded data structures
  - 3-valued logical structure



Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

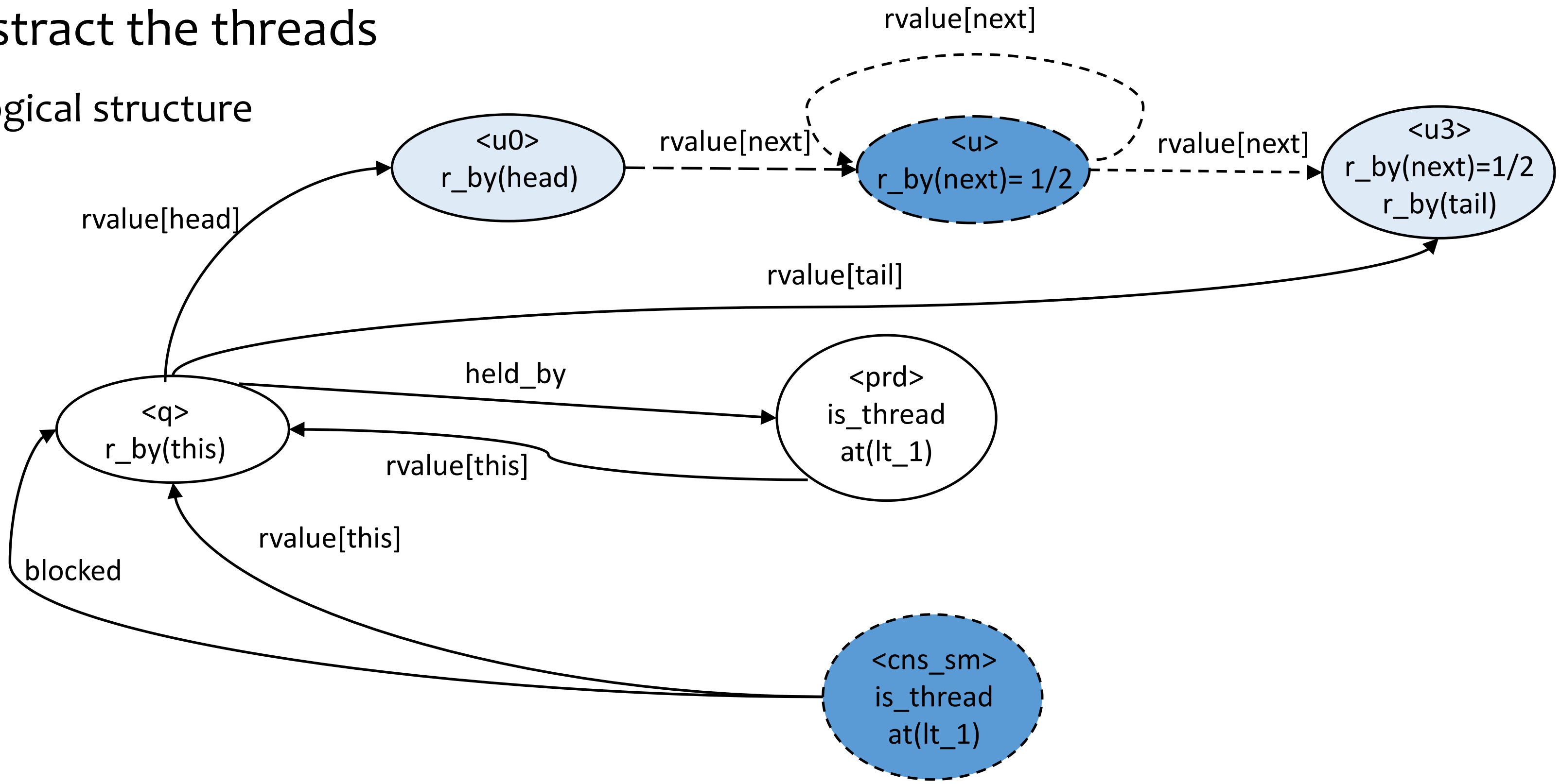
- Example: abstract the threads
  - 2-valued logical structure



Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

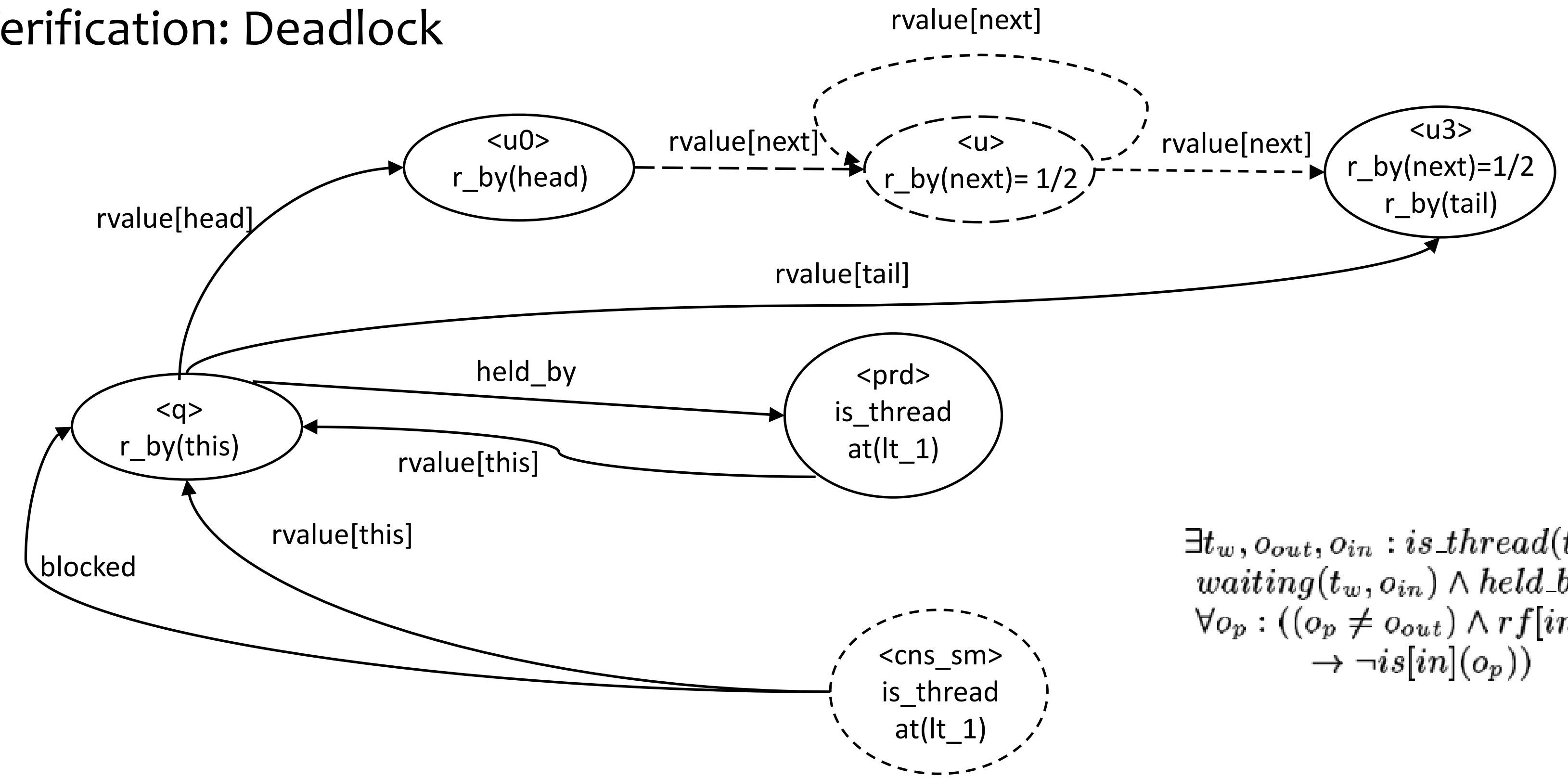
- Example: abstract the threads
  - 3-valued logical structure



Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

- Verification: Deadlock

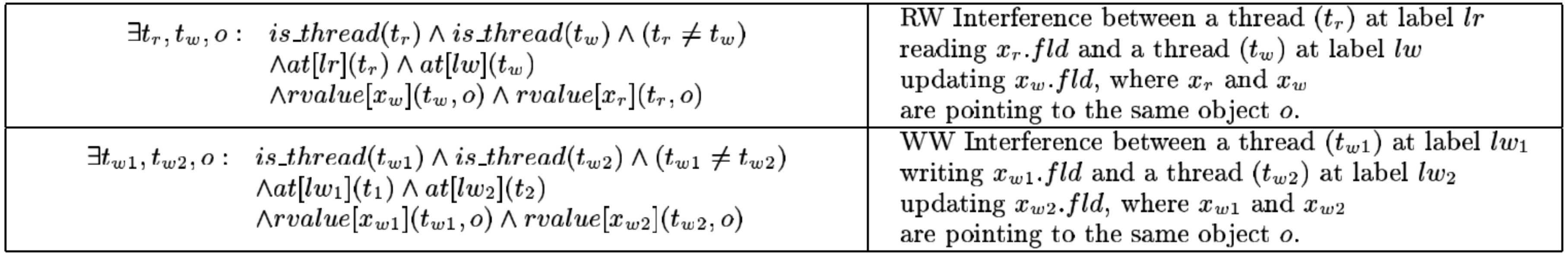


$$\begin{aligned}&\exists t_w, o_{out}, o_{in} : is\_thread(t_w)\wedge\\ &waiting(t_w, o_{in}) \wedge held\_by(o_{out}, t_w) \wedge rf[in](o_{out}, o_{in})\wedge\\ &\forall o_p : ((o_p \neq o_{out}) \wedge rf[in](o_p, o_{in}) \wedge rf[in](o_{out}, o_p)\\ &\qquad \rightarrow \neg is[in](o_p))\end{aligned}$$

Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.

Concurrent Program Verification

- Verification: Thread state errors
  - Read-Write Race(RW)
  - Write-Write Race(WW)



$\exists t_r, t_w, o: \quad is\_thread(t_r) \wedge is\_thread(t_w) \wedge (t_r \neq t_w)$ $\wedge at[lr](t_r) \wedge at[lw](t_w)$ $\wedge rvalue[x_w](t_w, o) \wedge rvalue[x_r](t_r, o)$	RW Interference between a thread (t_r) at label lr reading $x_r.fld$ and a thread (t_w) at label lw updating $x_w.fld$, where x_r and x_w are pointing to the same object o.
$\exists t_{w1}, t_{w2}, o: \quad is\_thread(t_{w1}) \wedge is\_thread(t_{w2}) \wedge (t_{w1} \neq t_{w2})$ $\wedge at[lw_1](t_1) \wedge at[lw_2](t_2)$ $\wedge rvalue[x_{w1}](t_{w1}, o) \wedge rvalue[x_{w2}](t_{w2}, o)$	WW Interference between a thread (t_{w1}) at label lw_1 writing $x_{w1}.fld$ and a thread (t_{w2}) at label lw_2 updating $x_{w2}.fld$, where x_{w1} and x_{w2} are pointing to the same object o.

  - For more details, refer to the paper *Verifying safety properties of concurrent Java program using 3-valued logic*

Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J] POPL, 2001, 36(3): 27-40.

List Manipulation Bug Detection

- Cleanness Checking
  - Pointer Dereference
    - x and x->n are not NULL in the statement x->n
  - Memory Leakage
    - X is uninitialized or
    - X is pointing to NULL
    - X is pointing to a heap cell which is also reachable from a different stack variable y

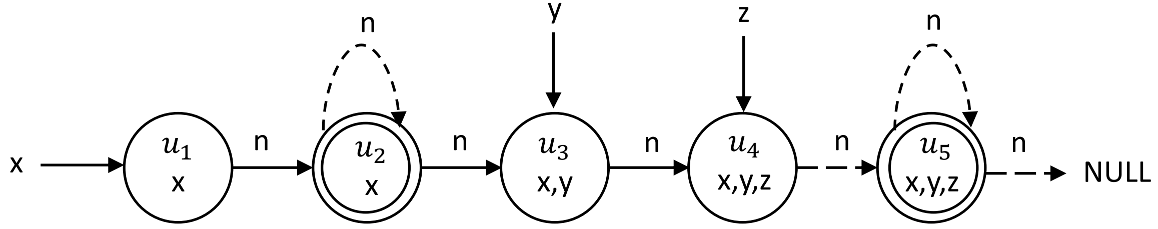
```
typedef struct node {
    struct node *n;
    int data;
} *List;


List fun(List x, int d) {
    t = malloc();
    t->data = d;
    e = x->n;
    x->n = t;
    t->n = e;
    return x;
}
```

Dor N, Rodeh M, Sagiv M. **Checking cleanness in linked lists**, ISSTA 2000: 115-134.

List Manipulation Bug Detection

- Core Predicates
  - Same as TVLA



- Instrumentation Predicates
  - Alloc(n): do not point to NULL
  - Unique(n): represent exactly one object
  - Unshared(n): be pointed by more than two objects

- Cleanness Checking
  - Pointer Dereference
    - x and x->n are not NULL in the statement x->n
  - Memory Leakage
    - X is uninitialized or
    - X is pointing to NULL
    - X is pointing to a heap cell which is also reachable from a different stack variable y

Convert to Constraint Solving Problem

Dor N, Rodeh M, Sagiv M. **Checking cleanness in linked lists**, ISSTA 2000: 115-134.

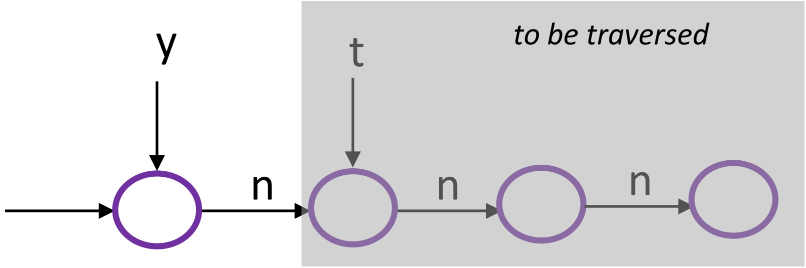
Memory Management

- Memory Management
  - Free analysis
    - Is it safe to insert a free statement in order to deallocate a garbage element?
  - Assign-null analysis
    - Is it safe to assign null to heap references that are not used further in the run?

Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

Memory Management

- A motivating example
  - Two operations
    - List creation
    - List traversal
  - Free analysis
    - It is safe to free the node pointed by y after line 10
    - It is safe to assign null to y.n after line 10



```
public static void main(String args[]) {
  L x, y, t;
  x = null;
  while (...) {
    y = new L();
    y.val = ...;
    y.n = y;
    x = y;
  }
  y = x;
  while (y != null) {
    System.out.println(y.val);
    t = y.n;
    free y? / set y.n to null?
    y = t;
  }
}
```

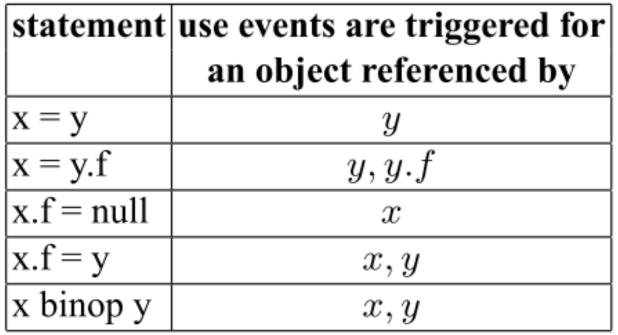
Line 10

Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

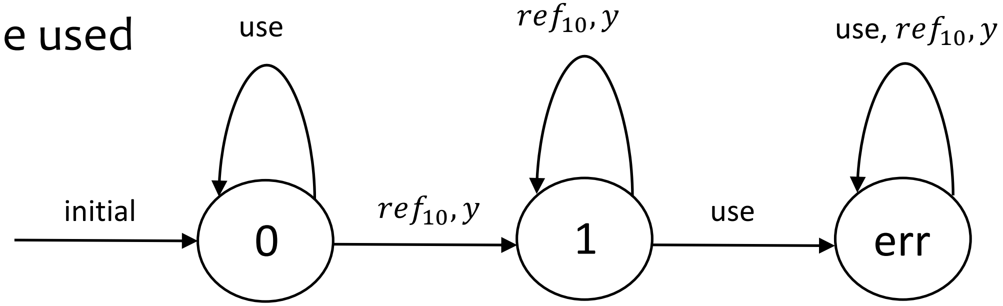
Memory Management

- Heap Safety Automaton(HSA)
  - For free analysis
  - Track the usage properties of each object at each program location
    - Use: triggered by a use a reference to the object
    - $ref_{plc,y}$: triggered when program execution is immediately after *plc* and use are triggered
  - The object referred by y can be freed at plc iff it will not be used

    (It can not have the err state)
  - For each program location and object, there is a corresponding HSA
    - Given an object, HSAs at all the program locations are the same.



statement	**use events are triggered for an object referenced by**
x = y	y
x = y.f	$y, y.f$
x.f = null	x
x.f = y	x, y
x binop y	x, y



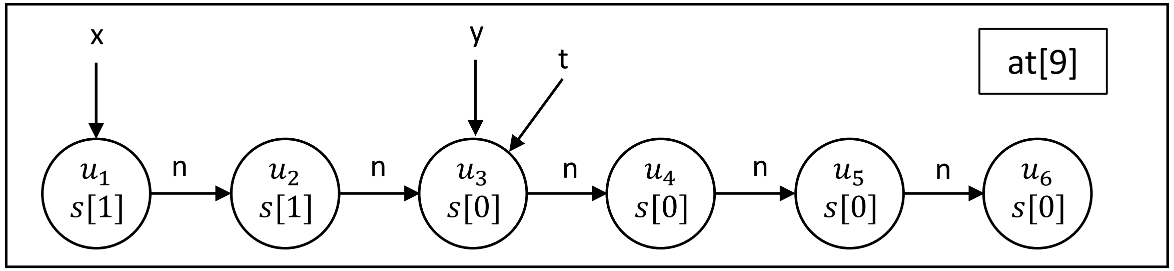
Example: the HSA of y at line 10

Accepting state: {0,1}

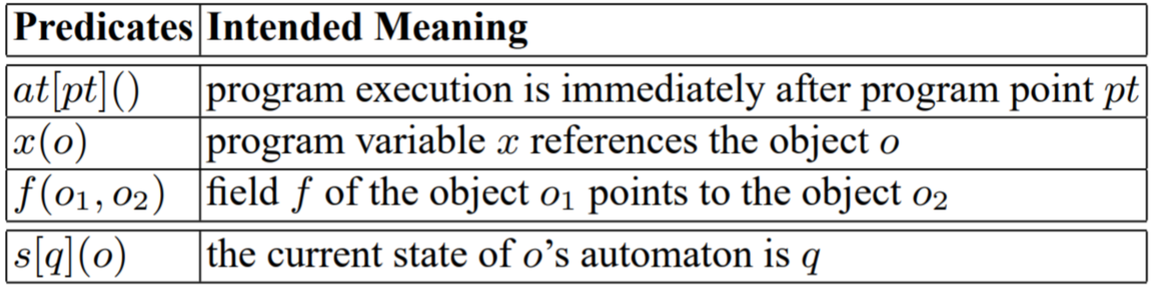
Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

Memory Management

- Shape graph



  - Encode shape graph with predicates



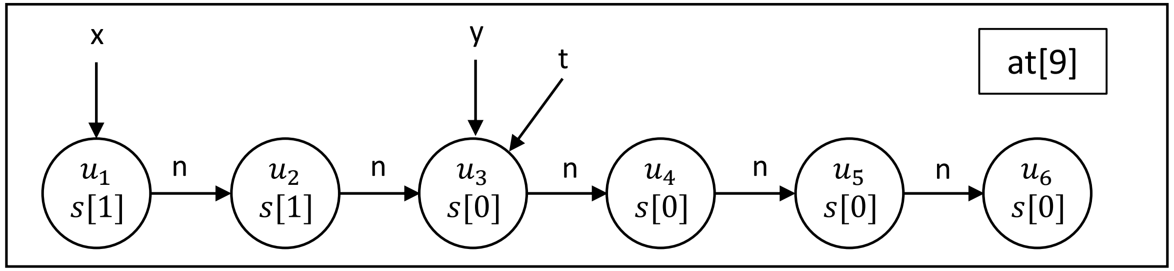
Predicates	**Intended Meaning**
$at[pt]()$	program execution is immediately after program point pt
$x(o)$	program variable x references the object o
$f(o_1, o_2)$	field f of the object o_1 points to the object o_2
$s[q](o)$	the current state of o's automaton is q

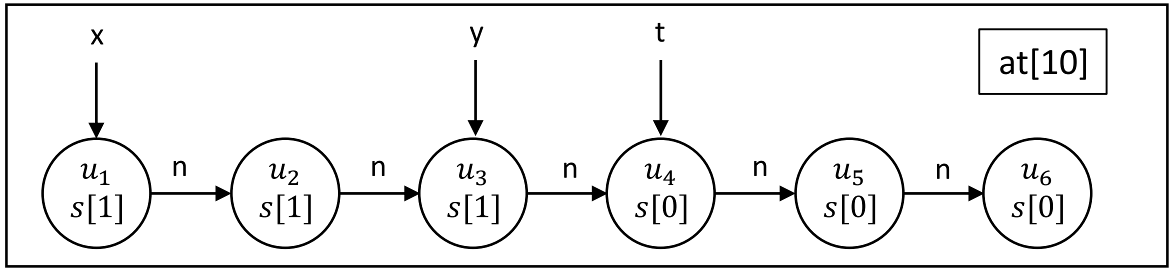
```
         public static void main(String args[]) {
           ...
           while (y != null) {
Line 9 ⇨     System.out.println(y.val);
             t = y.n;
             free y? / set y.n to null?
             y = t;
           }
         }
```

Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

Memory Management

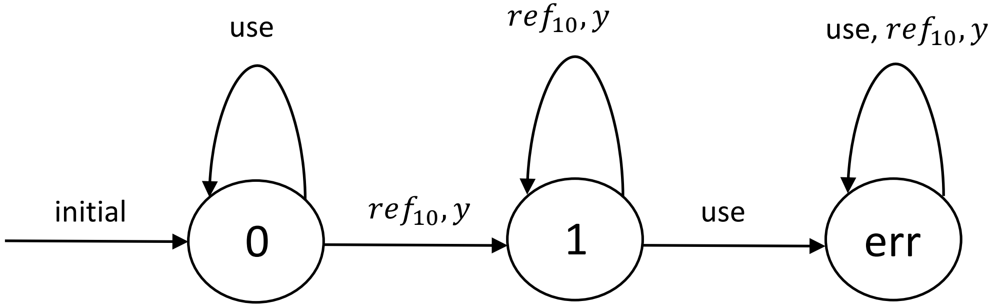
- Update shape graph based on HSA





Line 10 ⇨

```
public static void main(String args[]) {
    ...
    while (y != null) {
        System.out.println(y.val);
        t = y.n;
        free y? / set y.n to null?
        y = t;
    }
}
```

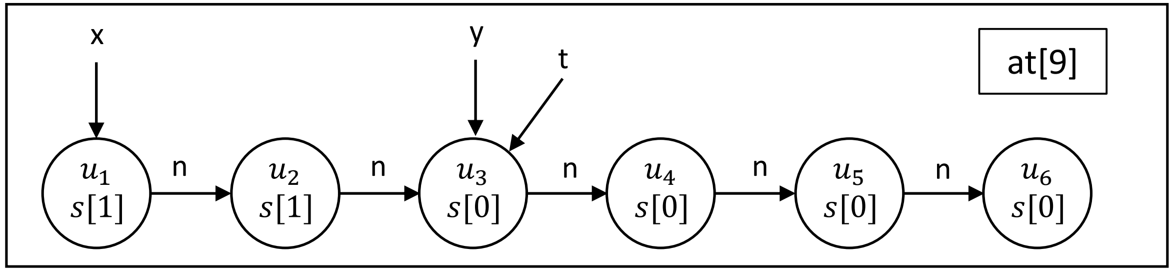


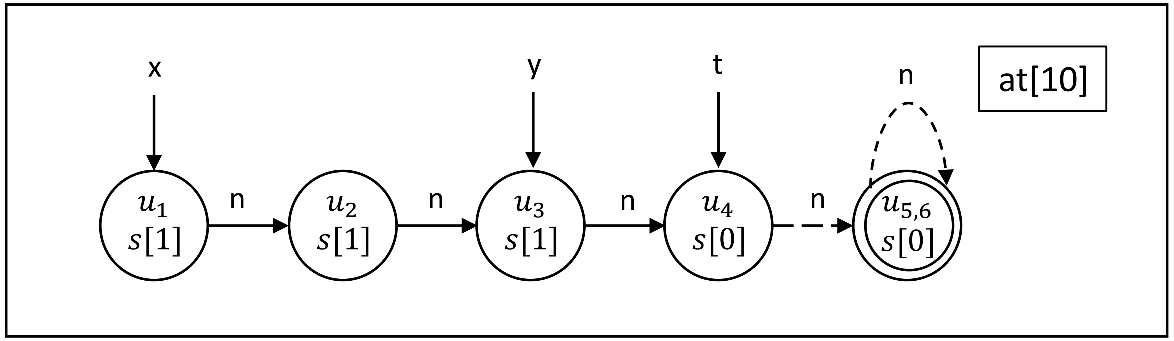
Example: the HSA of y at line 10

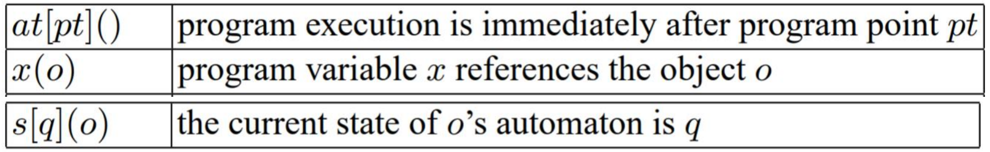
Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

Memory Management

- Canonical Embedding to 3-valued logical strutures







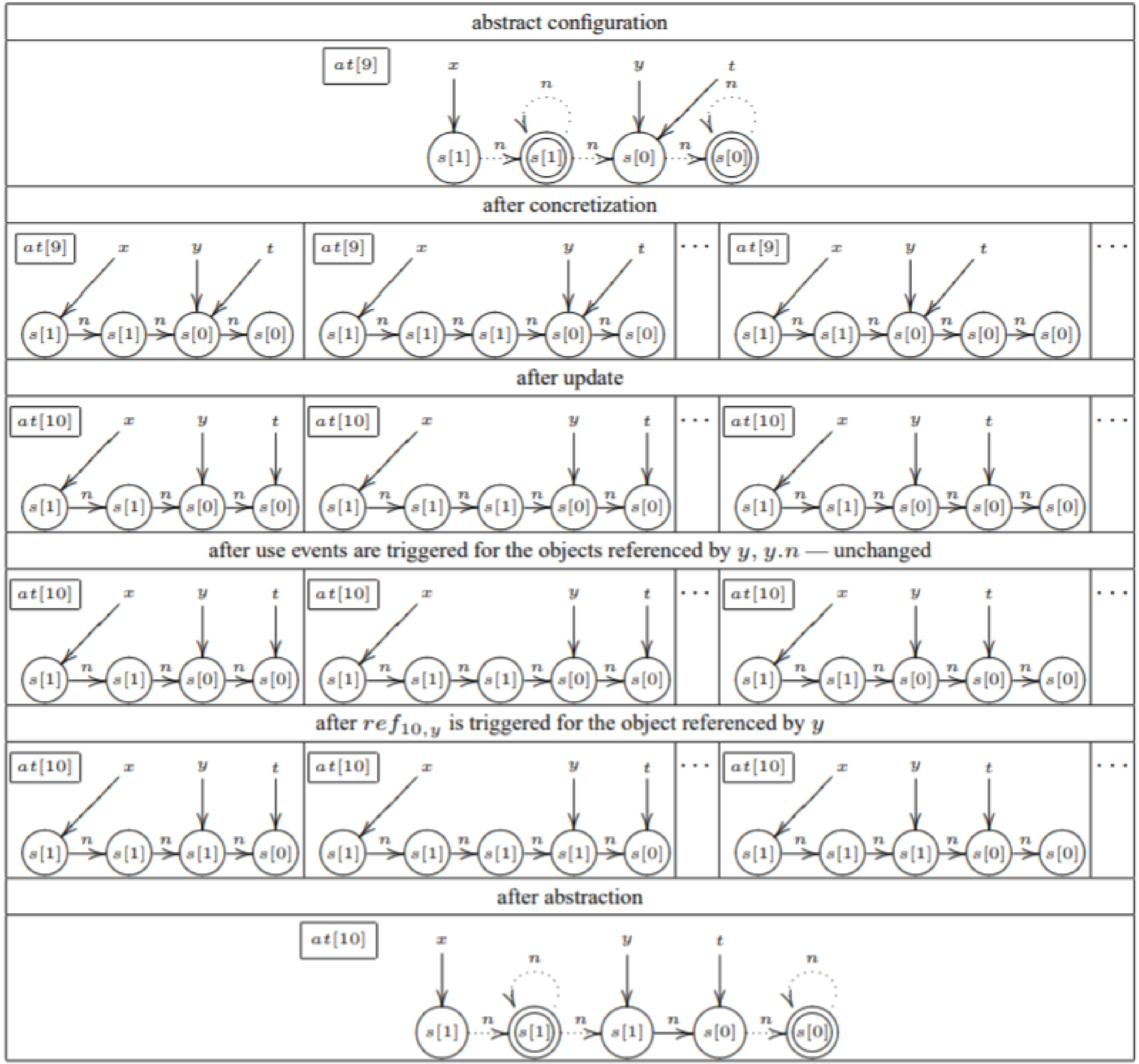
$at[pt]()$	program execution is immediately after program point pt
$x(o)$	program variable x references the object o
$s[q](o)$	the current state of o's automaton is q

Abstraction predicates

Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

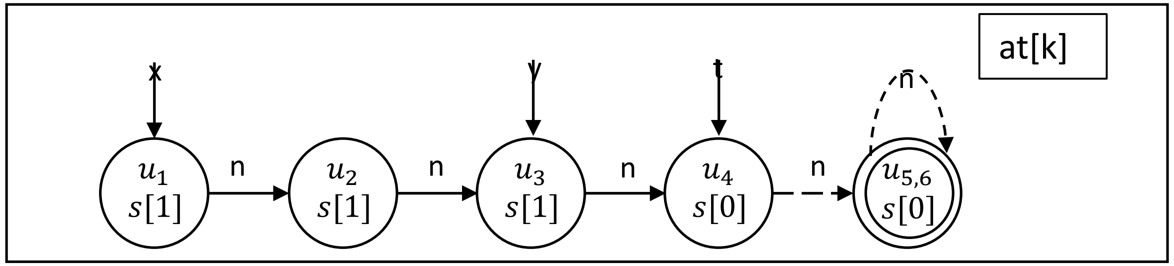
Memory Management

- Thanks for TVLA Embedding Theorem
- It is sound to update shape graphs in 3-valued logic domain
  - Focus(partial concretization)
  - Transfer(based on HSAs)
  - Coerce(remove inconsistencies)
  - Canonical abstraction


abstract configuration
at[9]
x
y
t
n
s[1]
s[0]
after concretization
...
after update
at[10]
after use events are triggered for the objects referenced by y, y.n — unchanged
after ref_{10,y} is triggered for the object referenced by y
after abstraction

Memory Management

- Model the original problem as a verification problem
  - Free analysis: verify in all abstract configurations, all individual are at the state 0 or 1.



  - Assign-null analysis: Discussed as an extension in Section 5 in the paper
- Related follow-on work
  - Effective typestate verification in the presence of aliasing

Shaham R, Yahav E, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C], SAS 2003: 483-503.

Fink S J, Yahav E, Dor N, et al. **Effective typestate verification in the presence of aliasing**[C]. ISSTA 2006

Testcase Generation

- Problem: Constraint solving based bug detection and safety verification are sound but incomplete, and cause false positives
- Motivation: Remove these false positives
- Insight
  - Testing is complete but unsound
  - The results of program analysis and verification can guide testcase generation
- Approach
  - Calculate the weakest precondition backward from the program location reporting errors
  - Solve the precondition at program entries to generate the testcases

Erez G, Yahav E, Sagiv M. **Generating concrete counterexamples for sound abstract interpretation**[M]. Tel Aviv University, 2004.

Testcase Generation

- Related works
  - Symbolic abstraction
  - Directed symbolic exection
  - Directly fuzzing

Erez G, Yahav E, Sagiv M. **Generating concrete counterexamples for sound abstract interpretation**[M]. Tel Aviv University, 2004.

TVLA: Summary

- The limitations of analogue pointer analysis in stage 1
  - The expressivity is limited
  - The shape properties are different in previous works(lack of general approach)
- TVLA
  - Abstract memory configuration(shape graph) and transformer(function summary) **in logical structures**
  - Encode and abstract memory configuration(shape graph) **by predicates** canonical abstraction
  - Encode the semantics **by logic formula** predicate update formula
  - Statement guides the pointwise state transformation focus operation

Strong expressivity
General framework

TVLA: Summary

- TVLA is rigorous and elegant
  - Encode memory configuration and transformer in **3-valued first-order logic**
  - Encode the semantics by **constraints**
  - Get precise abstractions(memory configuration/transformer) by **constraint solving**
  - Perform more **strong updates** by **symbolic abstraction**
- Application
  - Safe Property Verification: concurrency…
  - Memory Management
  - Not scalable in program analysis
    - first-order logic constraint solving in formula update, focus and coerce

The Future of TVLA

- Survive or die?
  - His spirit is always with us
- Inspiration
  - Strong update(3-valued logic)
  - Symbolic abstraction(Focus operation)
- How to improve its scalability
  - Other kinds of logic
  - Hybrid approach
  - …

Thanks
Q&A

TVLA Paper List

- Theoretical Work
- Numerical Domain Summarization
- Safety Property Verification
- List Manipulation Bug Detection
- Other Applications
  - Memory Management
  - Testcase Generation
- Strong Updates

Paper List: Theoretical Work

- Sagiv M, Reps T, Wilhelm R. **Parametric shape analysis via 3-valued logic**[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2002, 24(3): 217-298.
- Reps T, Loginov A, Sagiv M. **Semantic minimization of 3-valued propositional formulae**[C]//Proceedings 17th Annual IEEE Symposium on Logic in Computer Science. IEEE, 2002: 40-51.
- Reps T, Sagiv M, Loginov A. **Finite differencing of logical formulas for static analysis**[C]//European Symposium on Programming. Springer, Berlin, Heidelberg, 2003: 380-398.
- Yorsh G, Reps T, Sagiv M. **Symbolically computing most-precise abstract operations for shape analysis**[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2004: 530-545.

Paper List: Theoretical Work(cond)

- Loginov A, Reps T, Sagiv M. **Abstraction Refinement for 3-Valued Logic Analysis**[R]. University of Wisconsin-Madison Department of Computer Sciences, 2004.
- Reps T W, Sagiv M, Wilhelm R. **Static program analysis via 3-valued logic**[C]//International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2004: 15-30.
- Jeannet B, Loginov A, Reps T, et al. **A relational approach to interprocedural shape analysis**[C]//ACM Transactions on Programming Languages and Systems (TOPLAS), 2004
- Reps T, Sagiv M, Yorsh G**. Symbolic implementation of the best transformer**[C]// International Workshop on Verification, Model Checking, and Abstract Interpretation. Springer, Berlin, Heidelberg, 2004: 252-266.

Paper List: Numerical Domain Summarization

- Gopan D, DiMaio F, Dor N, et al. **Numeric domains with summarized dimensions**[C]. International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2004: 512-529.
- Gopan D, Reps T, Sagiv M. **A framework for numeric analysis of array operations**[J]. ACM SIGPLAN Notices, 2005, 40(1): 338-350.
- Jiang J, Chen L, Wu X, et al. **Block-Wise Abstract Interpretation by Combining Abstract Domains with SMT**[C]//International Conference on Verification, Model Checking, and Abstract Interpretation. Springer, Cham, 2017: 310-329.

Paper List: Safety Property Verification

- Yahav E. **Verifying safety properties of concurrent Java programs using 3-valued logic**[J]. POPL, 2001, 36(3): 27-40.
- Yahav E, Ramalingam G. **Verifying safety properties using separation and heterogeneous abstractions**[C]//ACM SIGPLAN Notices. ACM, 2004, 39(6): 25-34.

Paper List: List Manipulation Bug Detection

- Dor N, Rodeh M, Sagiv M. **Checking cleanness in linked lists**[C]//International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2000: 115-134.
- Lev-Ami T, Reps T, Sagiv M, et al. **Putting static analysis to work for verification: A case study**[C]//ACM SIGSOFT Software Engineering Notes. ACM, 2000, 25(5): 26-38.

Paper List: Other Applications

Memory Management

- Shaham R, Yahav E, Kolodner E K, et al. **Establishing local temporal heap safety properties with applications to compile-time memory management**[C]//International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2003: 483-503.

Testcase Generation

- Erez G, Yahav E, Sagiv M. **Generating concrete counterexamples for sound abstract interpretation**[M]. Tel Aviv University, 2004.

Paper List: Strong Updates

- Fink S J, Yahav E, Dor N, et al. **Effective typestate verification in the presence of aliasing**[C]. ISSTA 2006
- Dillig I, Dillig T, Aiken A. **Fluid updates: Beyond strong vs. weak updates**[C]//European Symposium on Programming. Springer, Berlin, Heidelberg, 2010: 246-266.