

Experience with Generic Shape Analysis

Mooly Sagiv
Tel Aviv University
(visiting Stanford University)

Denis Gopan, Akash Lal, Alexey Loginov, Tom Reps (University of
Madison-Wisconsin)

Igor Bogodlov, Tal Lev-Ami, Nurit Dor, Roman Manevich,
Noam Rinetzky, Ran Shacham, Eran Yahav, Greta Yorsh (TAU)

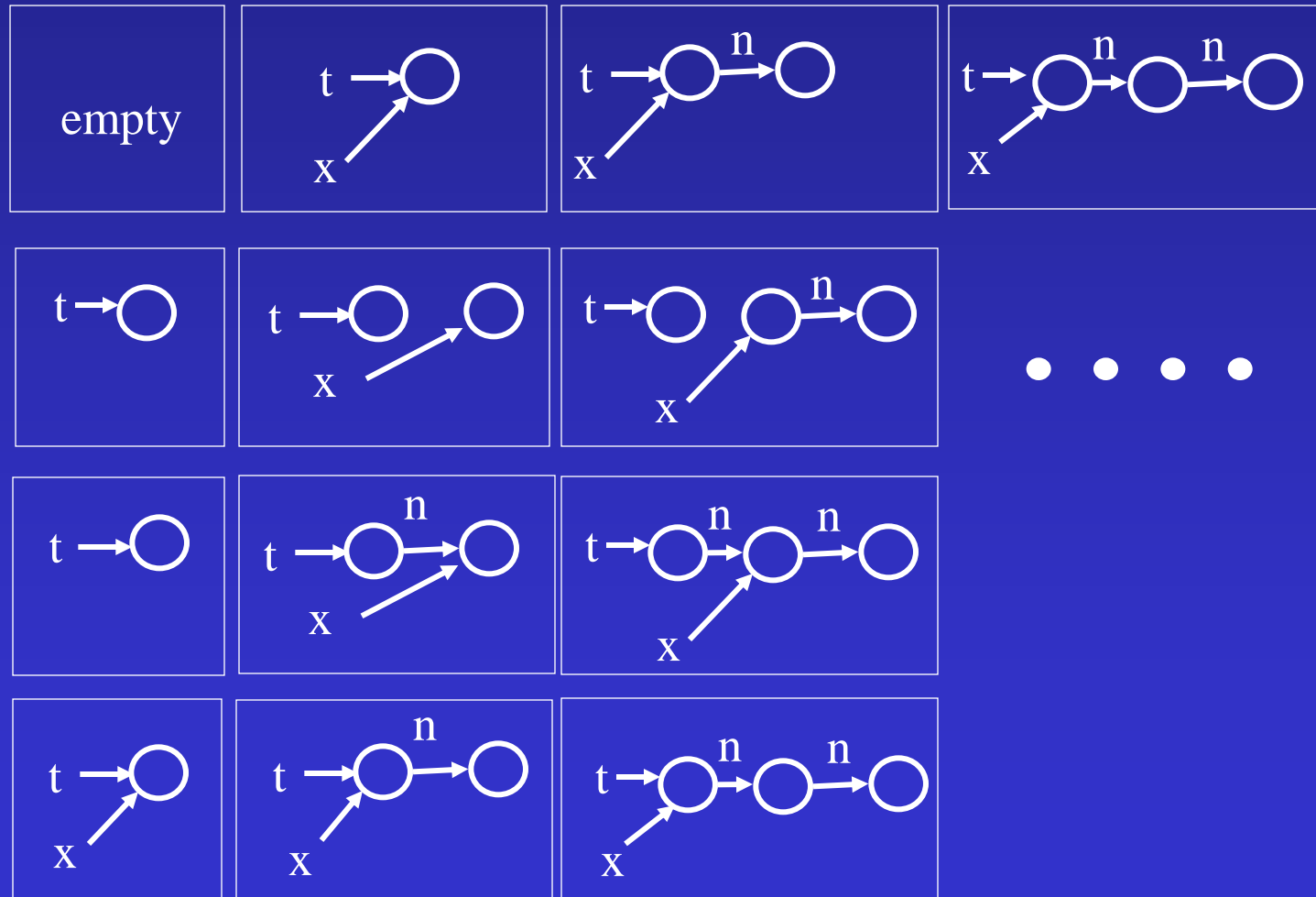
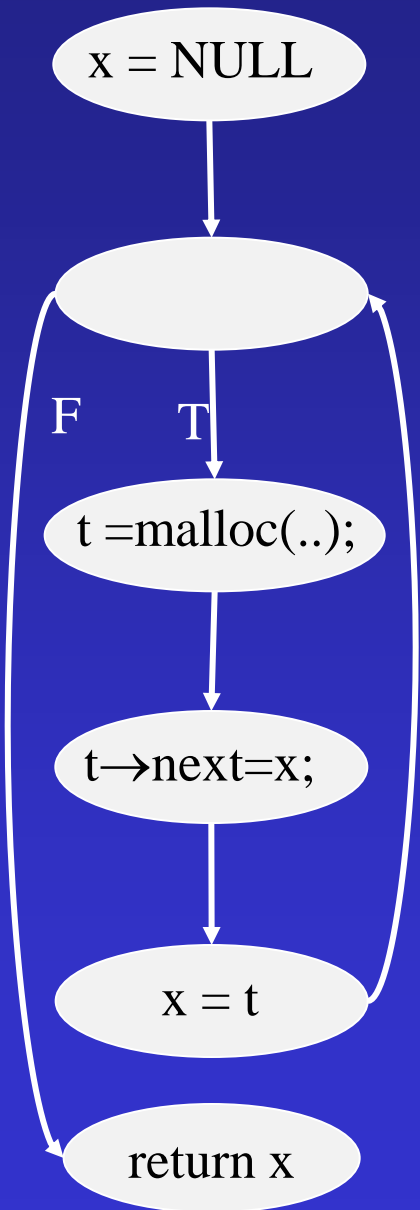
John Field (IBM)

Gilad Arnold, Bill McCloskey (UCB)

G. Ramalingam (MSR)

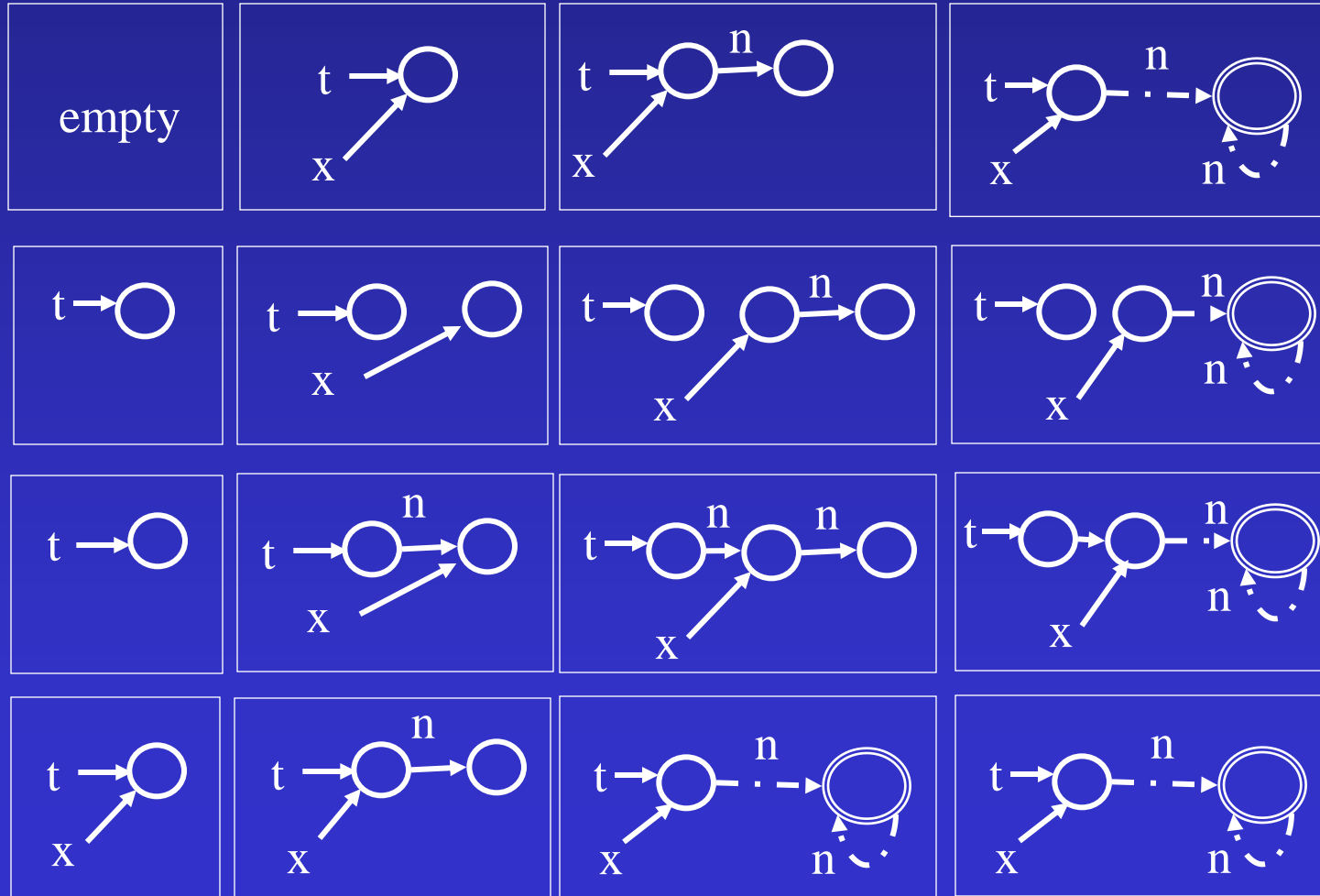
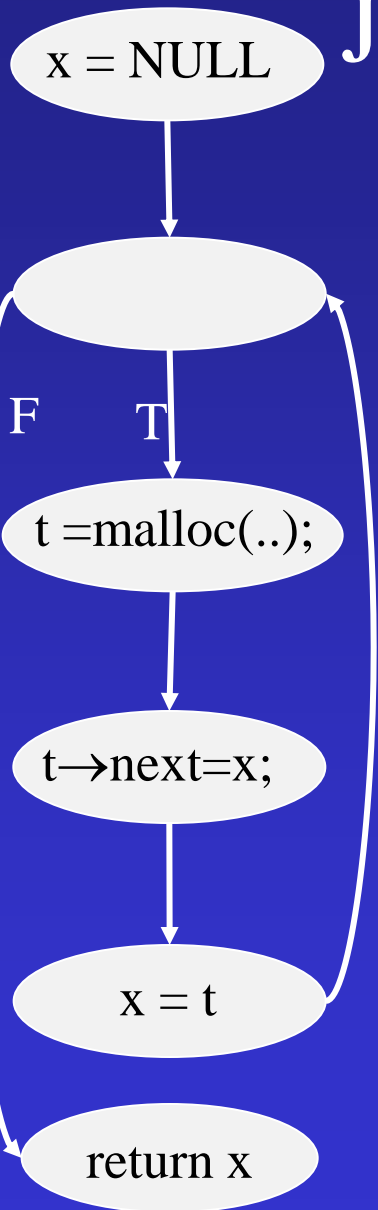
Reinhard Wilhelm (Universität des Saarlandes)

Example: Concrete Interpretation



Example: Shape Analysis

Jones and Muchnick 1981



TVLA: A parametric system for Shape Analysis

- A research tool
- Parameters
 - Concrete Semantics
 - States
 - Interpretation Rules
 - Abstraction
 - Transformers
- Iteratively compute a probably sound solution
- Specialize the shape analysis algorithm to class of programs

Partial Correctness

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

```
List InsertSort(List x) {
    List r, pr, rn, l, pl; r = x; pr = NULL;
    while (r != NULL) {
        l = x; rn = r → n; pl = NULL;
        while (l != r) {
            if (l → data > r → data) {
                pr → n = rn; r → n = l;
                if (pl == NULL) x = r;
                else pl → n = r;
                r = pr;
                break;
            }
            pl = l; l = l → n;
        }
        pr = r; r = rn;
    }
    assert sorted[x,n];
    //assert perm[x, n, x, n];
    return x;
}
```

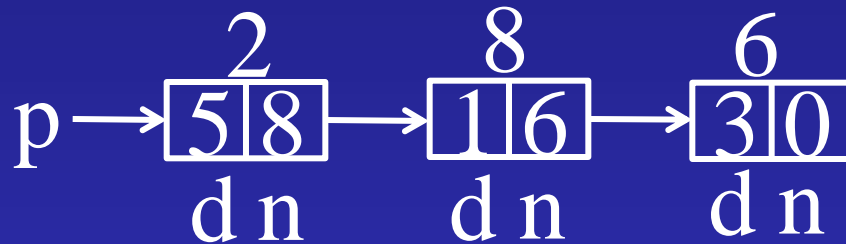
Partial Correctness

```
List quickSort(List p, List q) {  
    if(p==q || q == NULL)  
        return p;  
    List h = partition(p,q);  
    List x = p→n;  
    p →n = NULL;  
    List low = quickSort(h, p);  
    List high = quickSort(x, NULL);  
    p→n = high;  
    assert sorted[low, n];  
    return low;  
}
```

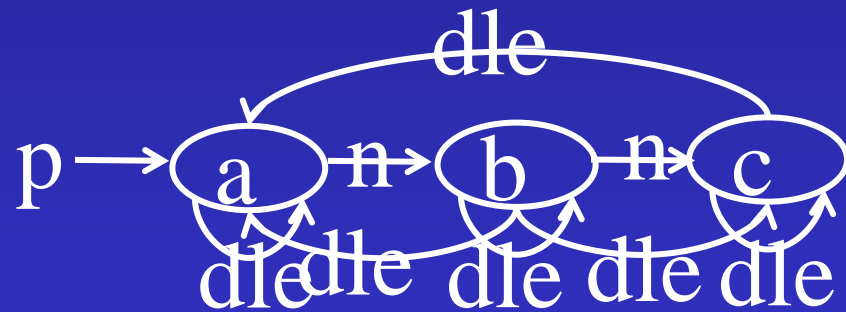
Concrete States in TVLA

- Nodes represent memory locations
- Unary relations represent:
 - Pointer variables
 - Boolean properties of memory locations
- Binary relations on nodes represent:
 - Pointer fields
 - Relations between locations

Insertion Sort



Concrete



TVLA

p	
a	1
b	0
c	0

n	a	b	c
a	0	1	0
b	0	0	0
c	0	0	0

dle	a	b	c
a	1	1	0
b	1	1	1
c	1	0	1

TVLA

Vanilla Concrete Interpretation Rules

Statement	Precondition	Update formula
$x = \text{NULL}$	true	$x'(v) := 0$
$x = y$	true	$x'(v) := y(v)$
$x = y \rightarrow f$	$\exists v: y(v)$	$x'(v) := \exists w: y(w) \wedge f(w, v)$
$x \rightarrow f = y$	$\exists v: x(v)$	$f'(v, w) :=$ $\begin{cases} y(w) & x(v) \\ f(v, w) & \neg x(v) \end{cases}$

Concrete Interpretation Rules for dle

Condition	Precondition
$x \rightarrow d \leq y \rightarrow d$	$\exists v, w: x(v) \wedge y(w) \wedge dle(v, w)$
sorted [x,n]	$\forall u, v, w: x(u) \wedge n^*(u, v) \wedge n(v, w) \rightarrow dle(v, w)$

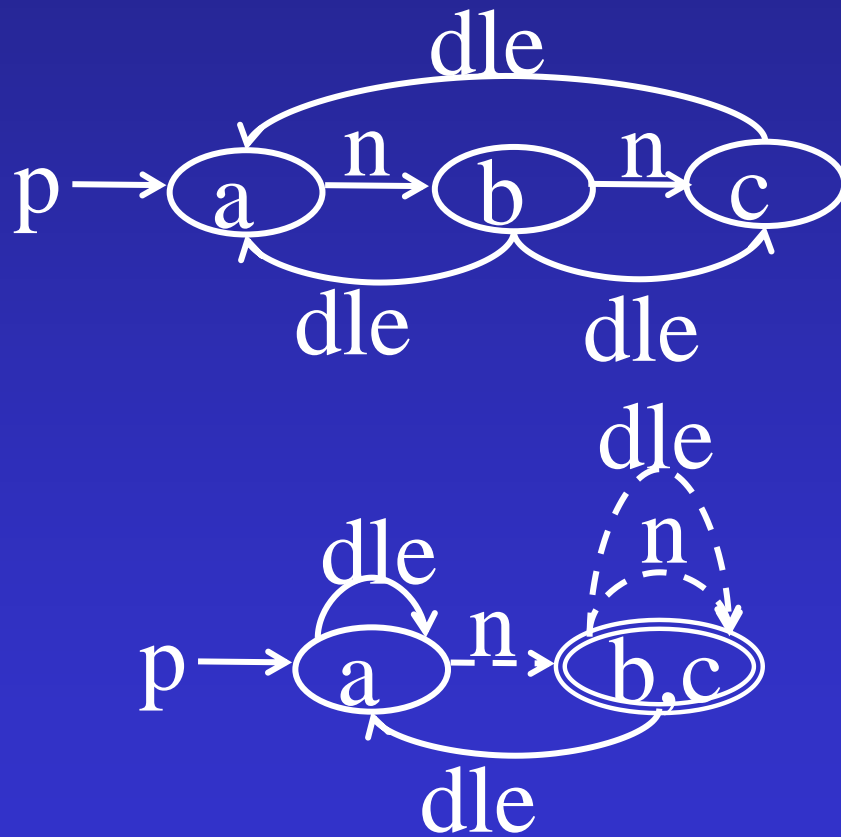
3-Valued Logical Structures

- A set of individuals (nodes) U
- Relation meaning
 - Interpretation of relation symbols in P
 - $p^0() \rightarrow \{0, 1, 1/2\}$
 - $p^1(v) \rightarrow \{0, 1, 1/2\}$
 - $p^2(u, v) \rightarrow \{0, 1, 1/2\}$
- A join semi-lattice: $0 \sqcup 1 = 1/2$

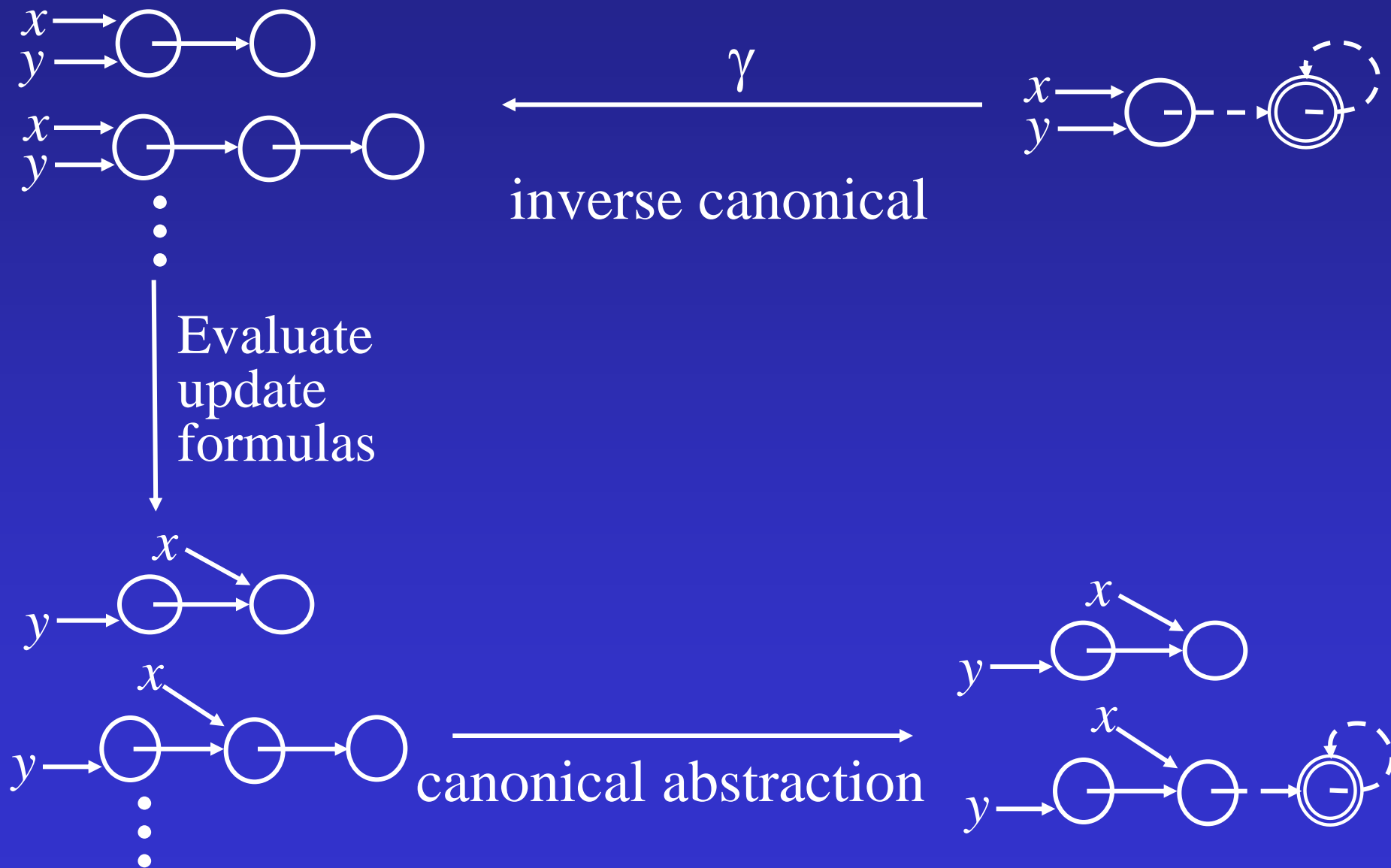
Canonical Abstraction (β)

- Partition the individuals into **equivalence classes** based on the values of their unary relations
 - Every individual is mapped into its equivalence class
- Collapse relations via \sqcup
 - $p^S(u'_1, \dots, u'_k) = \sqcup \{p^B(u_1, \dots, u_k) \mid f(u_1)=u'_1, \dots, f(u_k)=u'_k\}$
- At most 2^A abstract individuals

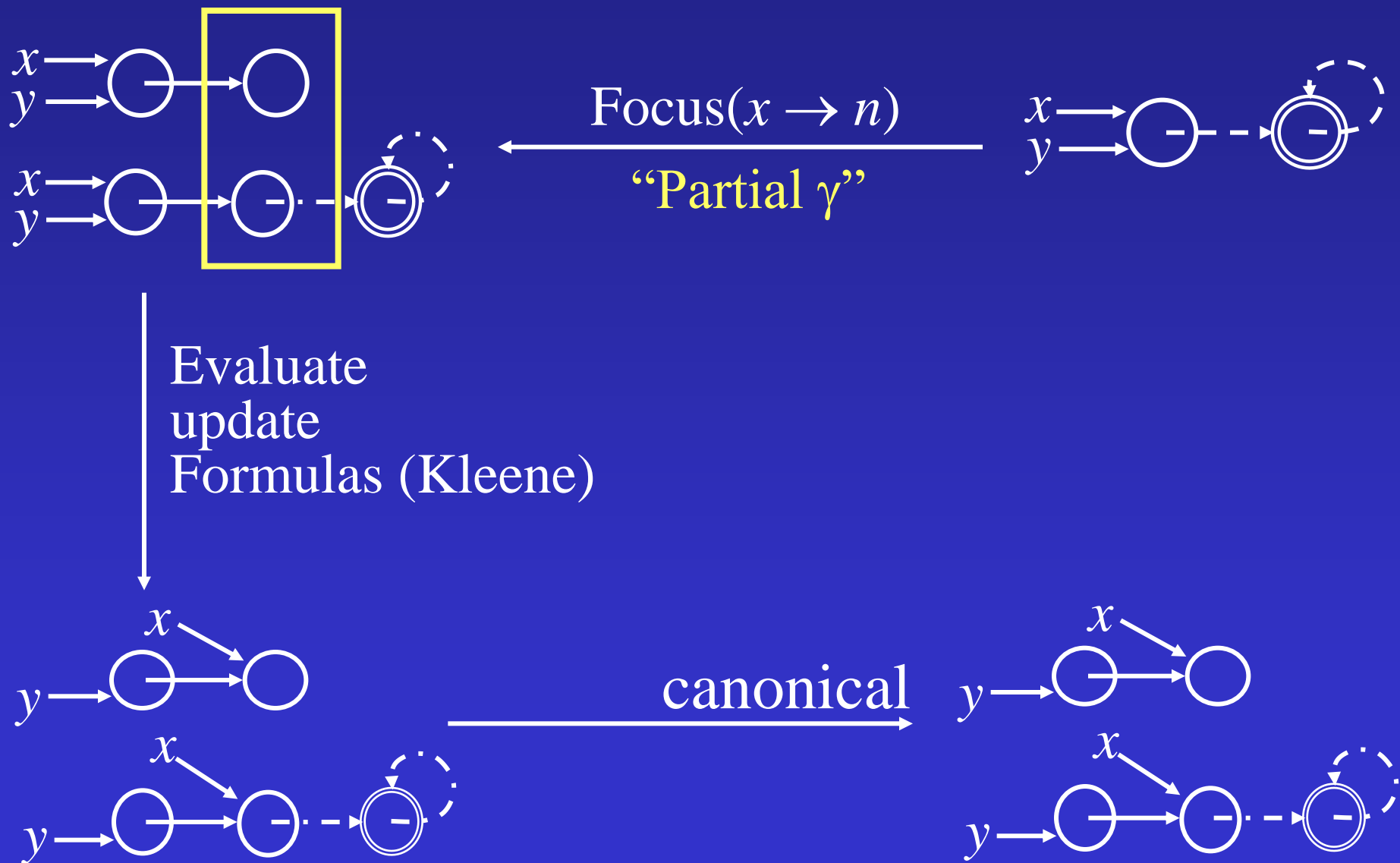
Sorting



Best Abstract Transformer ($x = x \rightarrow n$)



“Focus”-Based Transformer ($x = x \rightarrow n$)



Increasing Precision

- Global invariants
 - User-supplied, or consequence of the semantics of the programming language
 - Naturally expressed in FOTC
- Record extra information in the concrete interpretation
 - Tunes the abstraction
 - Refines the concretization

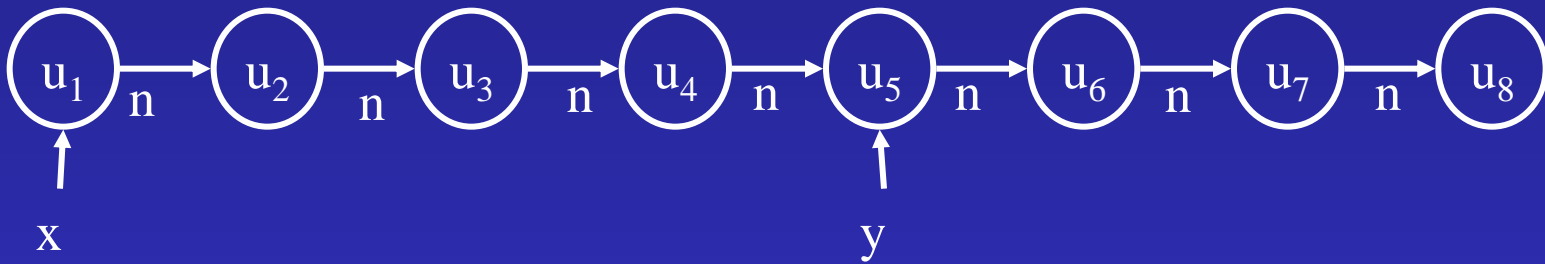
Global Invariants (Sorting)

- For all pointer variables x :
 $\forall v, w: x(v) \wedge x(w) \rightarrow v = w$
- For all pointer fields:
 $\forall u, v, w: f(u, v) \wedge f(u, w) \rightarrow v = w$
- $\forall u: dle(u, u)$
- $\forall u, v: dle(u, v) \wedge dle(v, w) \rightarrow dle(u, w)$
- $\forall u, v: dle(u, v) \vee \neg dle(v, u)$

Instrumentation Principle

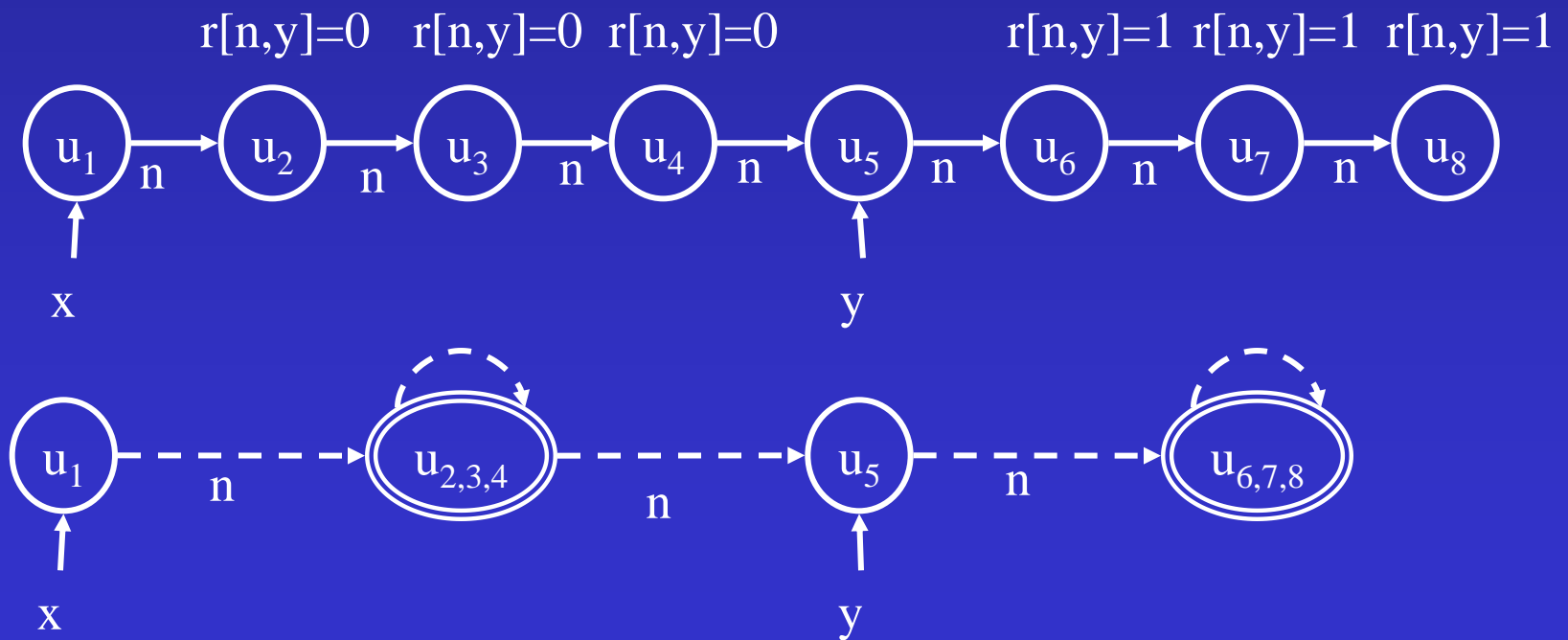
- Refine the abstraction by explicitly recording information lost by the abstraction
- TVLA users define extra information
- Automatically maintained by the system

List Segments

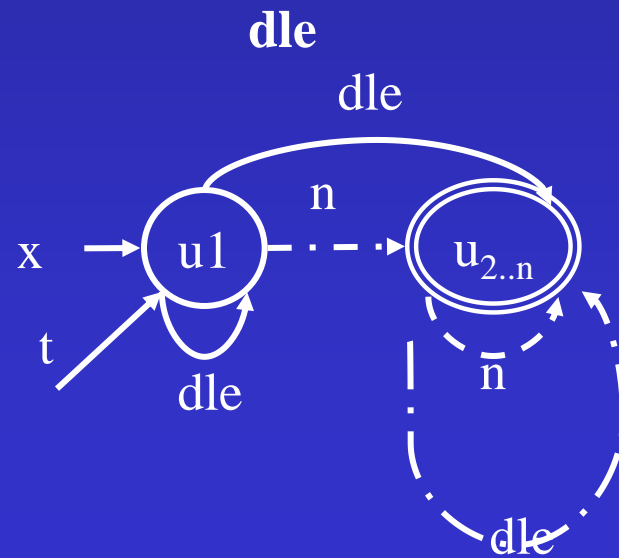
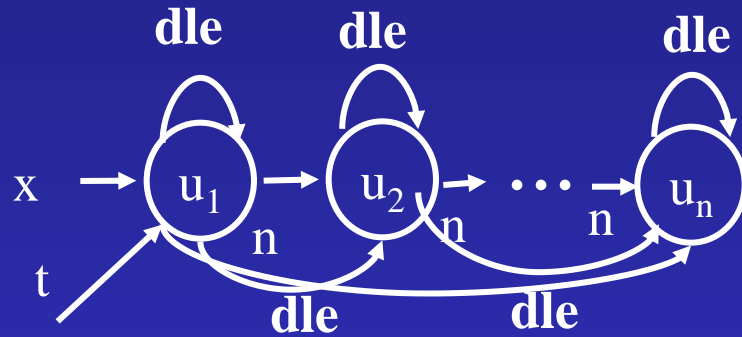


Reachability from a Variable

- $r[n,y](v) = \exists w: y(w) \wedge n^*(w, v)$



Sortedness



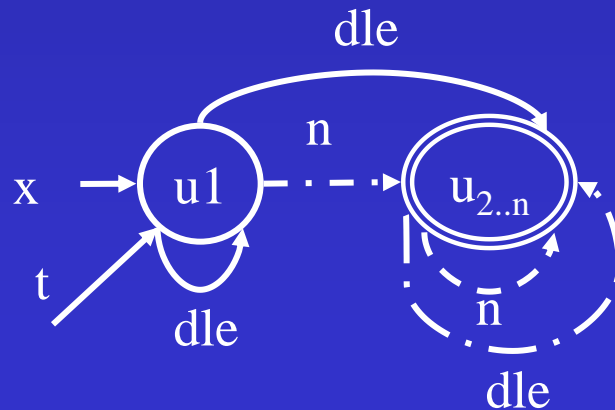
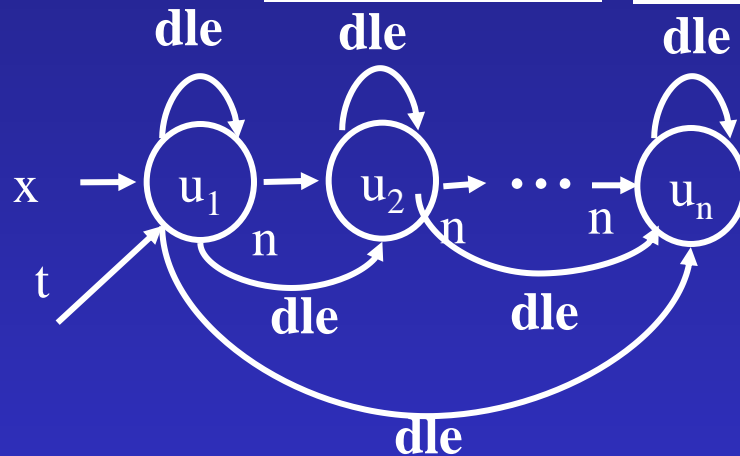
Example: Sortedness

$$\text{inOrder}(v) = \forall v1: n(v, v1) \rightarrow \text{dle}(v, v1)$$

inOrder = 1

inOrder = 1

inOrder = 1



inOrder = 1

inOrder = 1

Additional Instrumentation Relations

- $\text{is}[f](v) \exists w1, w2: w1 \neq w2 \wedge f(w1, v) \wedge f(w2, v)$
- $\text{c}[f,b](v) = \forall w: f(v, w) \rightarrow b(w, v)$
- $\text{tree}(v)$
- $\text{dag}(v)$
- Weakest Precondition
[Ramalingam, PLDI'02]
- Learned via Inductive Logic Programming
[Loginov, CAV'05]
- Relevance heuristics

Automatically Maintaining Instrumentation

$$\text{inOrder}(v) = \forall v1: n(v, v1) \rightarrow \text{dle}(v, v1)$$

Statement	Precondition	Update formula
$x \rightarrow n = \text{NULL}$	$\exists v: x(v)$	$n'(v, w) := \begin{cases} 0 & x(v) \\ n(v, w) & \neg x(v) \end{cases}$ $\text{InOrder}'(v) := \begin{cases} 1 & x(v) \\ \text{inOrder}(v) & \neg x(v) \end{cases}$
$x \rightarrow n = y$	$\exists v: x(v) \wedge \forall w: \neg n(v, w)$	$n'(v, w) := \begin{cases} y(w) & x(v) \\ n(v, w) & \neg x(v) \end{cases}$ $\text{InOrder}'(v) := \begin{cases} \exists w: y(w) \wedge \text{dle}(v, w) & x(v) \\ \text{inOrder}(v) & \neg x(v) \end{cases}$

Instrumentation (Summary)

- Refines the abstraction

$$r[n, y](v) = \exists w: y(w) \wedge n^*(w, v)$$

- Adds global invariants

$$\forall v: r[n, y](v) \leftrightarrow \exists w: y(w) \wedge n^*(w, v)$$

$$\gamma(S^\#) = \{S : S \models \Sigma, \beta(S) = S^\#\}$$

Scaling

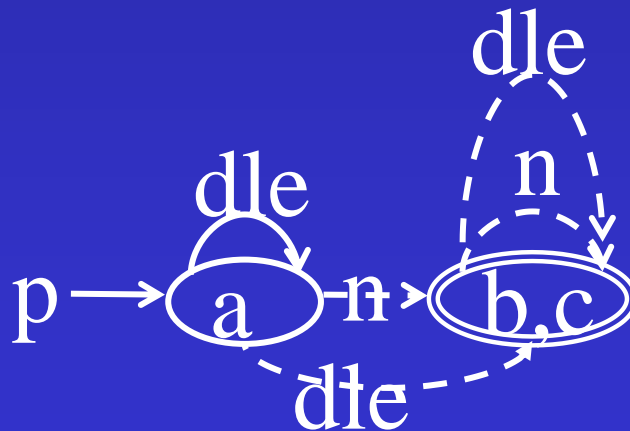
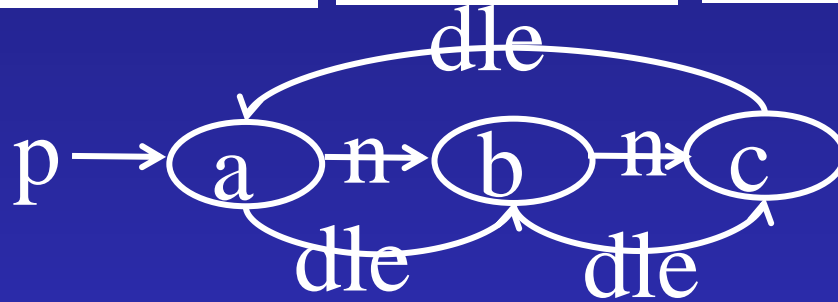
- Reducing the number of shape nodes
 - Abstraction vs. Non-Abstraction Predicates
- Reducing the number of shape graphs
 - Partial Join [Manevich, SAS'04]
 - Heap Decomposition [Manevich, SAS'08]
- Handling procedures [Rintezkey, POPL'05, Jeanneat, TOPLAS'09]
 - Relational analysis with only local pointers
- Handling concurrency

InOrder as distinction

inOrder = 1

inOrder = 0

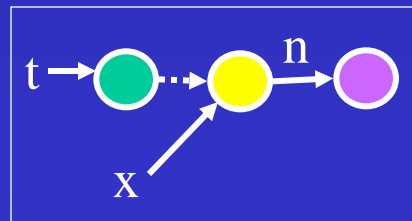
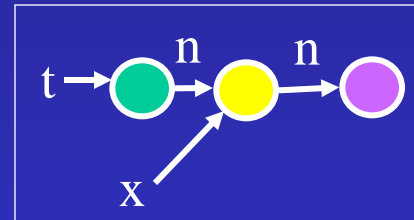
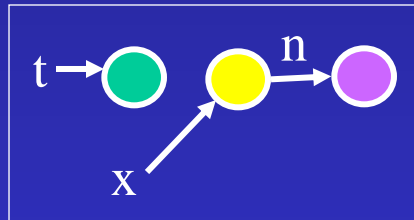
inOrder = 1



inOrder = 1

inOrder = 1/2

Partially Disjunctive vs. Powerset Abstraction



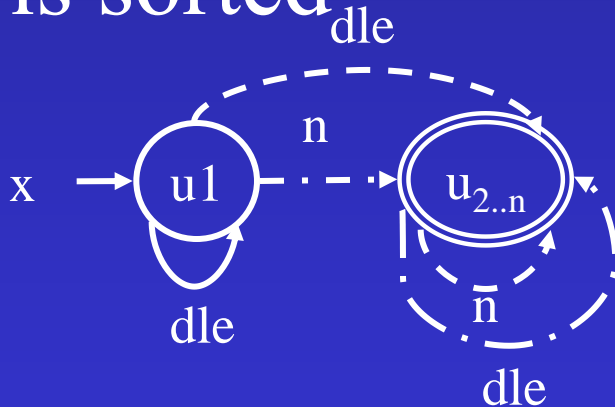
Example: InsertSort

Abstraction	Time sec.	#Structures
\cup /abs inorder	>1800	> 166,000
\cup /nonabs inorder	12.5	6,938
\sqcup /abs inorder	72.6	16,612
\sqcup /nonabs inorder	2.2	1,489

```
List InsertSort(List x) {  
  List r, pr, rn, l, pl; r = x; pr = NULL;  
  while (r != NULL) {  
    l = x; rn = r → n; pl = NULL;  
    while (l != r) {  
      if (l → data > r → data) {  
        pr → n = rn; r → n = l;  
        if (pl == NULL) x = r;  
        else pl → n = r;  
        r = pr;  
        break;  
      }  
      pl = l; l = l → n;  
    }  
    pr = r; r = rn;  
  }  
  assert(sorted[n,x]);  
  return x;  
}
```

Example: InsertSort (with bug)

“Unable to prove
that the list pointed
by x is sorted”



inOrder = 1/2

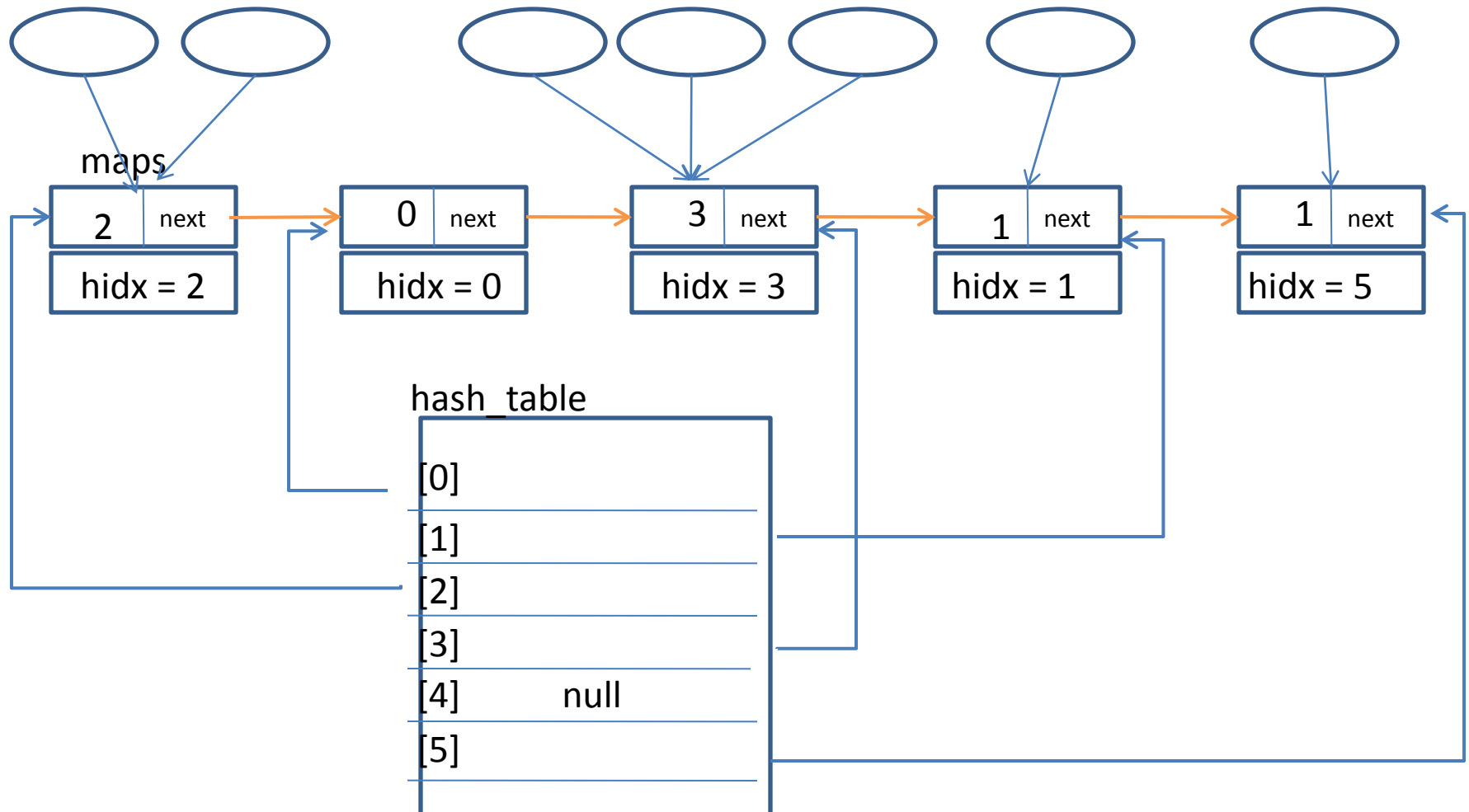
inOrder = 1

```
List InsertSort(List x) {
  if (x == NULL) return NULL
  pr = x; r = x->n;
  while (r != NULL) {
    pl = x; rn = r->n; l = x->n;
    while (l != r) {
      pr->n = rn ;
      r->n = l;
      pl->n = r;
      r = pr;
      break;
    }
    pl = l;
    l = l->n;
  }
  pr = r;
  r = rn;
  assert(sorted[n,x]);
}
```

DESKCHECK: Bill McCloskey

- Inspired by TVLA
- Combines Canonic Abstraction with Numeric Analysis
- Richer modeling language
- Nelson & Oppen like architecture
- More automatic
- Used to prove memory safety of real system's code

tthttpd: Web Server



$\forall n: \text{maps}. \forall v: \mathbb{Z}. \text{hash_table}[v] == n \rightarrow n \rightarrow \text{hash_idx} == v$

$\forall n: \text{maps}. n \rightarrow \text{rc} = | \{ n' : \text{http_conn} . n' \rightarrow \text{file_data} == n \} |$

thttpd:add_hash

```
static int
add_hash( Map* m )
{
    unsigned int h, he, i;

    h = hash( m->ino, m->dev, m->size, m->ctime );
    he = ( h + hash_size - 1 ) & hash_mask;
    for ( i = h; ; i = ( i + 1 ) & hash_mask )
    {
        if ( hash_table[i] == (Map*) 0 )
        {
            hash_table[i] = m;
            m->hash = h;
            m->hash_idx = i;
            return 0;
        }
        if ( i == he )
            break;
    }
    return -1;
}
```

Analysis of tthttpd web server

Function	Analysis time (s)
mmc_map	28.23
mmc_unmap	9.08
mmc_cleanup	76.81
mmc_destroy	5.80
Total	123.47

Other Applications

- Absence of Concurrent Modification Exceptions [Ramalingam, PLDI'02]
- Linearizability of lock-free concurrent linked-list implementations [Amit CAV'07, Berdine CAV'08, Manevich SAS'08]

Some Mistakes

- Not thinking of intended users
- The modeling language (TVP) is terrible
 - Transition System
 - Combines UI with semantics
 - Two low leveled
 - Uses CPP
 - Untyped
 - ...
- Insufficient documentation
- Interacting with other systems
- Java frontend came out too late
 - No user interaction
- Not going open source

Questions

- Can programmers specialize program analysis?
- Interactive program analysis?

Lessons Learned

- TVLA can successfully model complex system code
- Prove interesting properties of small intricate programs
- Useful research tool
- Specialize the analysis to a class of programs
- Repeatable
 - A single TVP suffices for all the linked list sorting routines