Abstract Interpretation and the Heap

Computer Science and Artificial Intelligence Laboratory
MIT

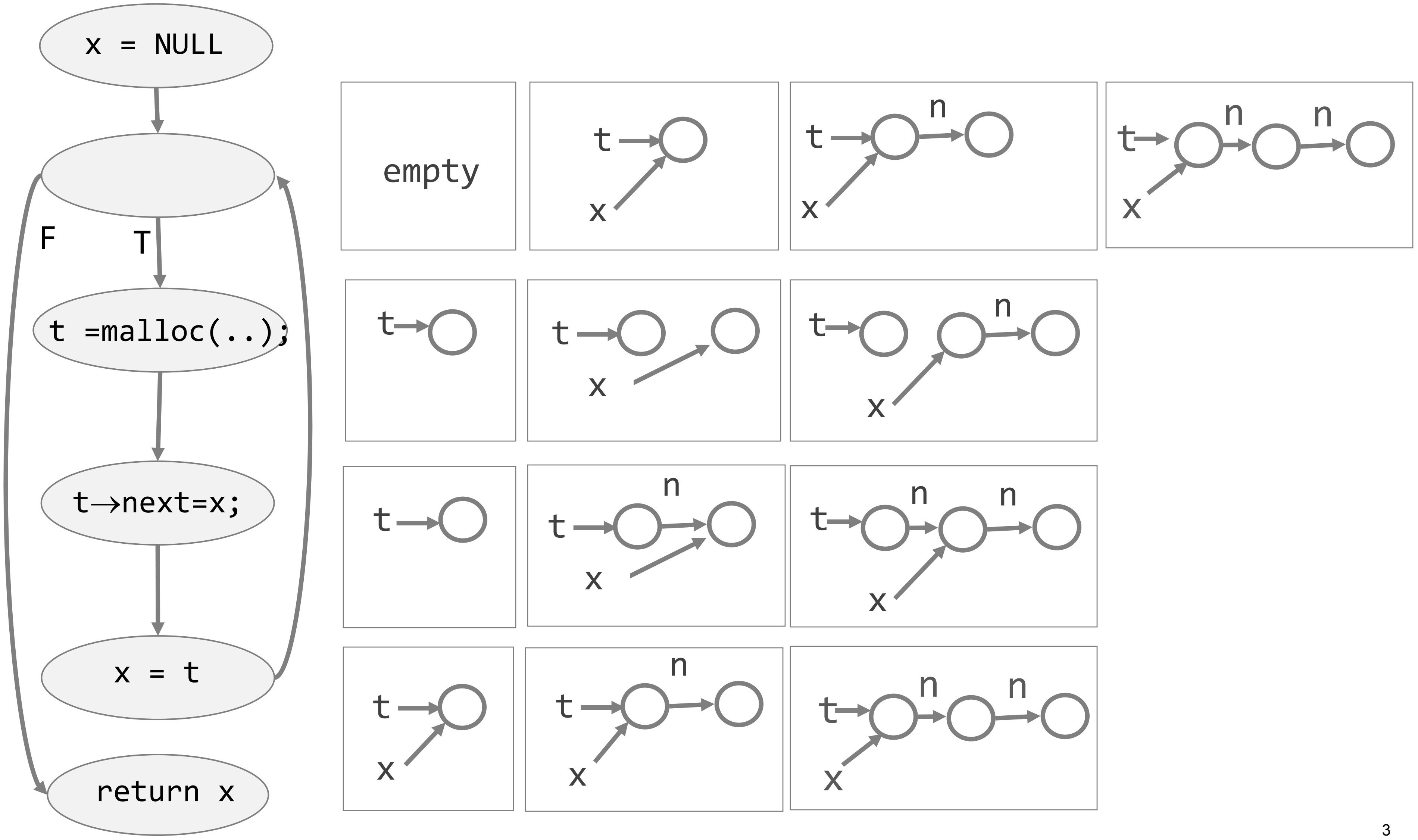
With slides and examples by Mooly Sagiv.
Used with permission.

Nov 18, 2015

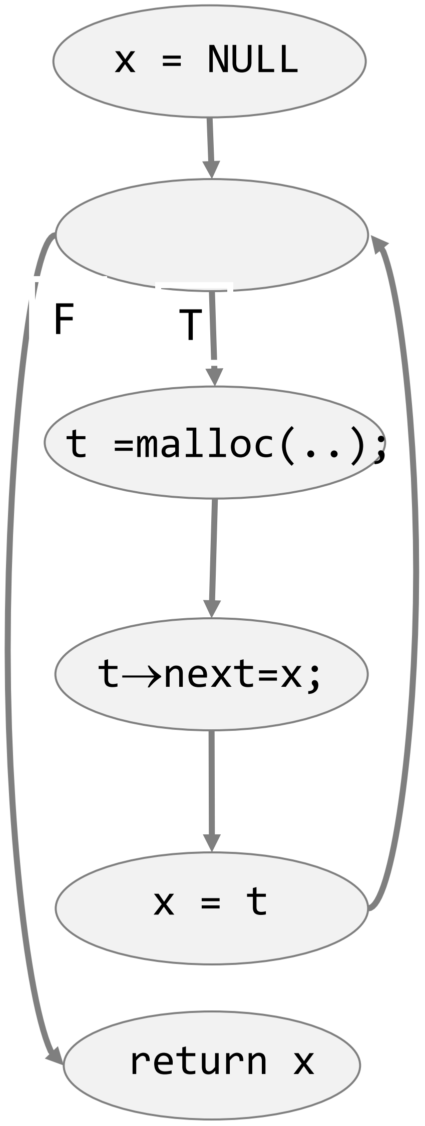
Recap: Collecting Semantics

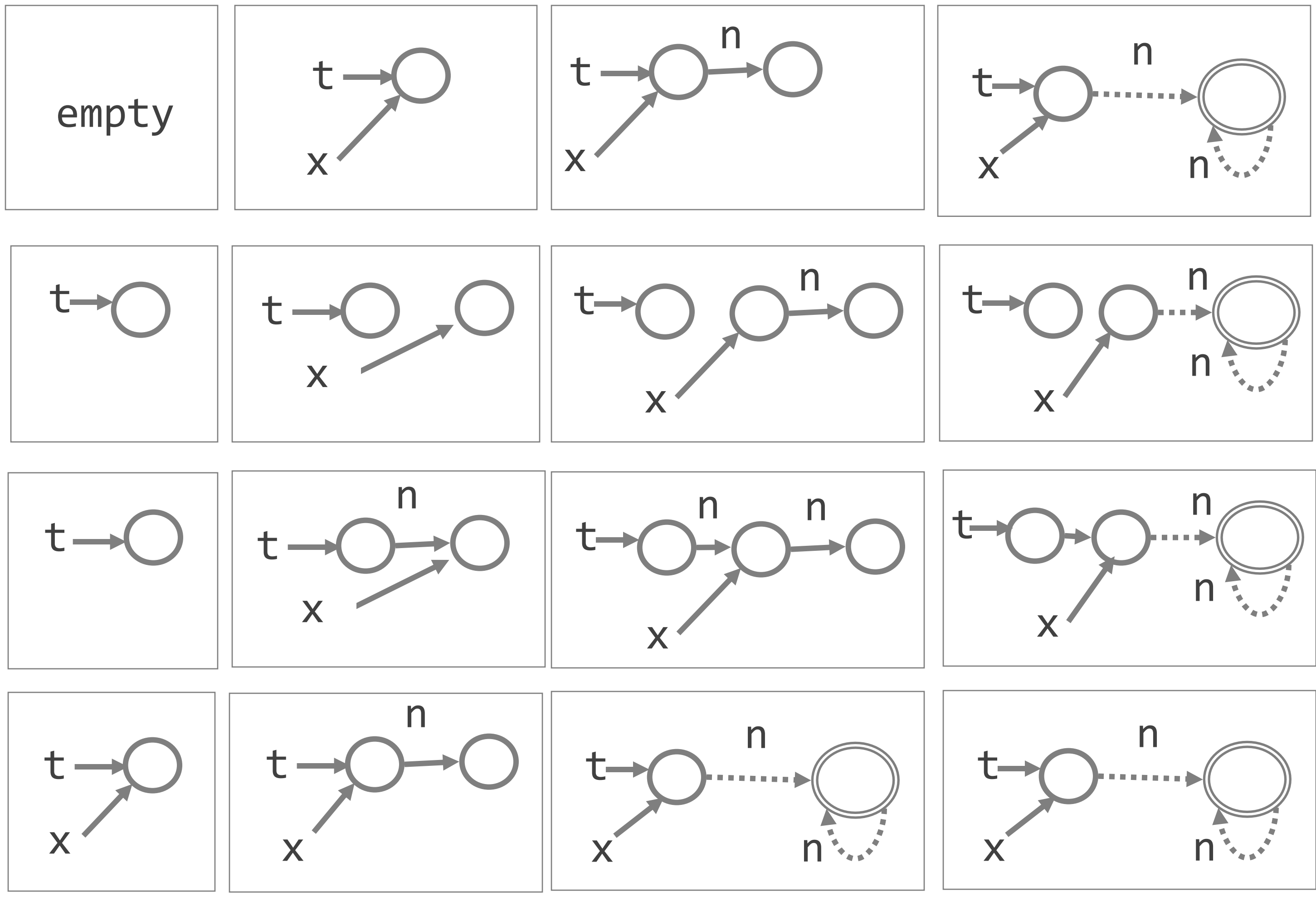
Compute for each program the true set of states that can occur at each point

Example: Collecting Interpretation



Example: Abstract Interpretation





Concrete Interpretation

A slightly different view of the state

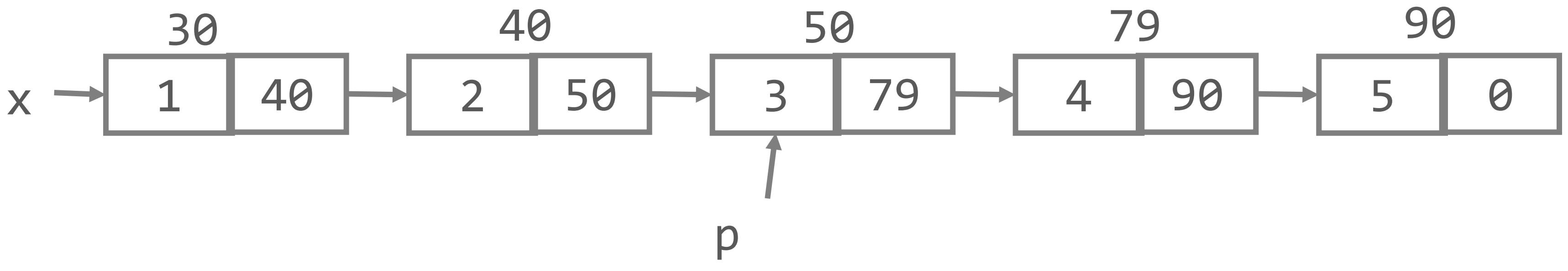
- Env: $Var \rightarrow Values$
- One map per field
- Field: $Loc \rightarrow Values$
- $Values = Loc \cup Atoms$

Example

- Env $=[x \rightarrow 30, p \rightarrow 79]$
- Fields:

  next $= [30 \rightarrow 40, 40 \rightarrow 50, 50 \rightarrow 79, 79 \rightarrow 90]$

  val$=[30 \rightarrow 1, 40 \rightarrow 2, 50 \rightarrow 3, 79 \rightarrow 4, 90 \rightarrow 5]$



The TVLA Approach

Represent the store with logical predicates

- Then do abstraction on these predicates
- An approach to building abstractions instead of a single one

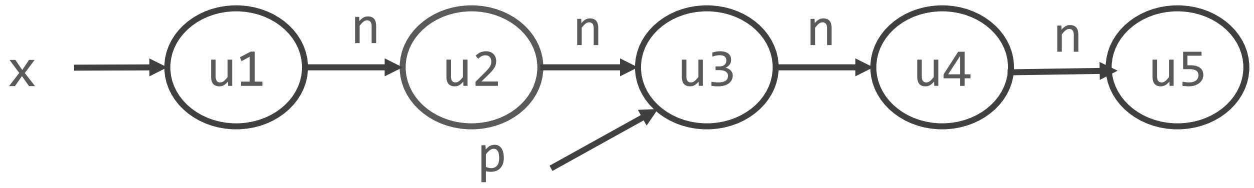
Locations ≈ Individuals

Program variables ≈ Unary predicates

Fields ≈ Binary predicates

Example

- U = {u1, u2, u3, u4, u5}
- x = {u1}, p = {u3}
- n = {<u1, u2>, <u2, u3>, <u3, u4>, <u4, u5>}



Important notation

Transitive closure of a binary predicate $n(u, v)$

- $n^*(u, v) \coloneqq u = v \lor (\exists w.\ n(u, w) \land n^*(w, v))$
- $n^+(u, v) \coloneqq (\exists w.\ n(u, w) \land n^*(w, v))$

Concrete Interpretation

State:

- x : predicate for variable x.
- n : predicate for next field

Rules $[\![s]\!](x, n) = (x', n')$

- $[\![x = null]\!] \quad n' = n \quad \forall v.\ x'(v) = 0$
- $[\![x = malloc()]\!] \quad n' = n \quad \forall v.\ x'(v) = IsNew(v)$
- $[\![x = y]\!] \quad n' = n \quad \forall v.\ x'(v) = y(v)$
- $[\![x = y.next]\!] \quad n' = n \quad \forall v.\ x'(v) = \exists w. y(w) \land n(w, v)$
- $[\![x.next = y]\!] \quad x' = x \quad \forall v\, w.\ \ n'(v, w) = \big(\neg x(v) \land n(v, w)\big) \lor (x(v) \land y(w))$

Stating program properties

x points to an acyclic list

- $\forall\, v\, w.\ x(v) \land n^*(v,w) \to \neg n^+(w,v)$

The heap n' reverses the list pointed at by x in n

- $\forall v\, w\, r.\, x(v) \land n^*(v,w) \to (n(w,r) \leftrightarrow n'(r,w))$

Canonical Abstraction

Convert logical structures of unbounded size into bounded size

Guarantees that number of logical structures in every program is finite

Every first-order formula can be conservatively interpreted

Same idea we explored last time, but revisited in Three Valued Logic

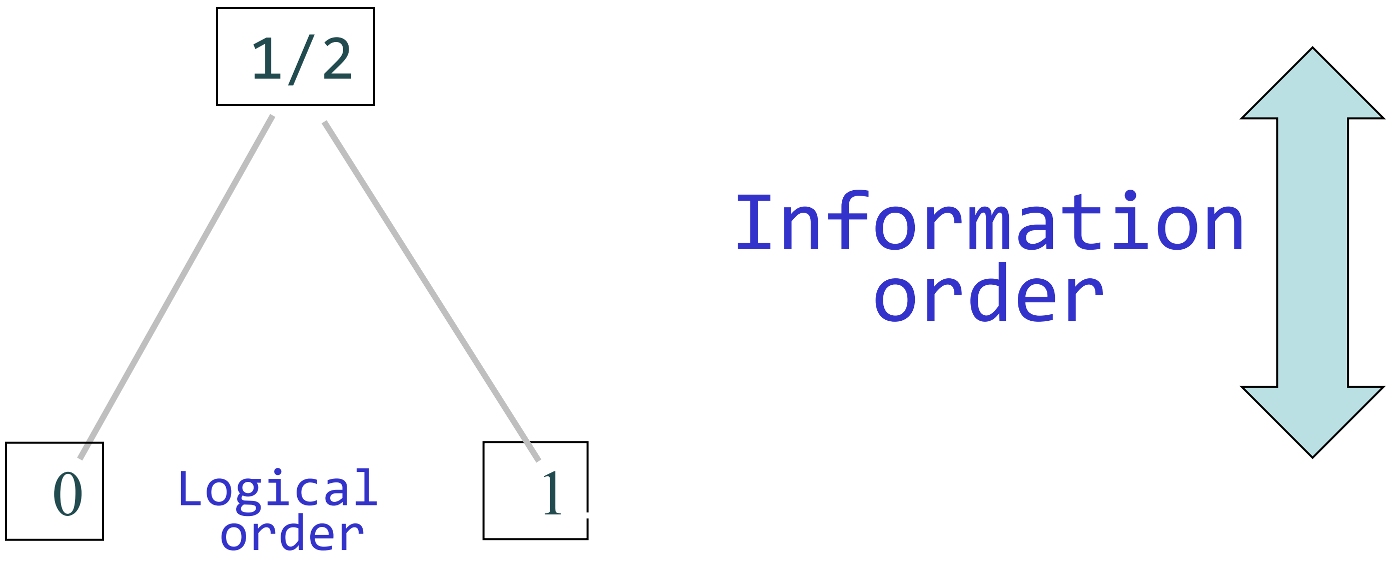
Kleene Three-Valued Logic

1: True

0: False

1/2: Unknown

A join semi-lattice: $0 \sqcup 1 = 1/2$



Boolean Connectives [Kleene]

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

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

Key idea

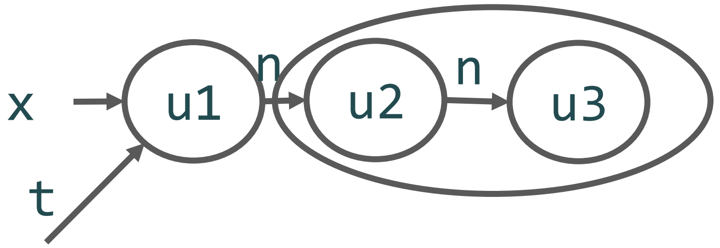
Predicates describing program state are now predicates in 3-Valued Logic

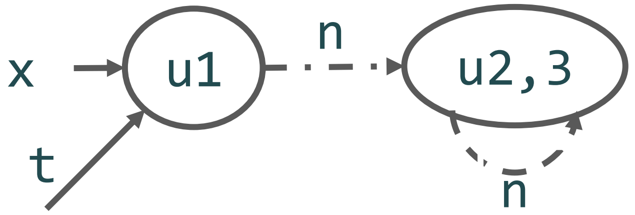
- Let U be the set of individuals in the concrete domain (potentially infinite)
- Let U’ be the set of individuals in the abstract domain (finite)
- Let $f: U \to U'$
- Then a predicate p^B over U can be abstracted to p^S over U' as follows

  $$p^S(u'_1, \dots u'_k) = \sqcup \{p^B(u_1, \dots, u_k) \mid f(u_1) = u'_1, \dots, f(u_k) = u'_k\}$$
- Since U' is bounded, p^S can be represented with a table

Canonical Abstraction

```
x = NULL;
while (…) do {
      t = malloc();
      t →next=x;
     x = t
}
```





```
n(u1,u2)=1
n(u1,u3)=0
n(u2,u3)=1
n(u3,u3)=0
…
```

```
ns(u1,u23)=1/2
n(u23,u23)=1/2
```

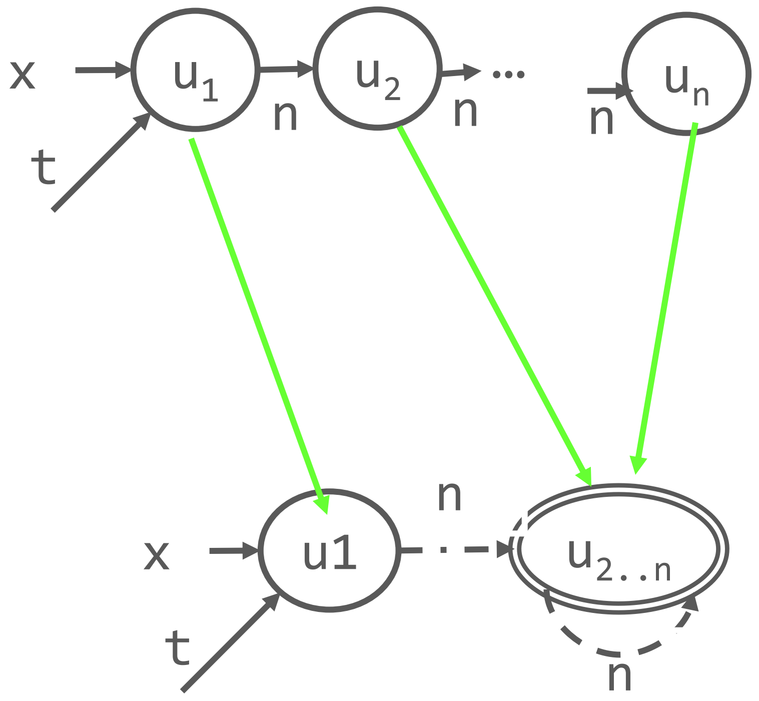
Big Idea

You can increase precision by tracking additional predicates

Cyclicity predicate

$$c[x]() = \exists v_1, v_2 : x(v_1) \wedge n^*(v_1, v_2) \wedge n^+(v_2, v_2)$$

c[x]()=0



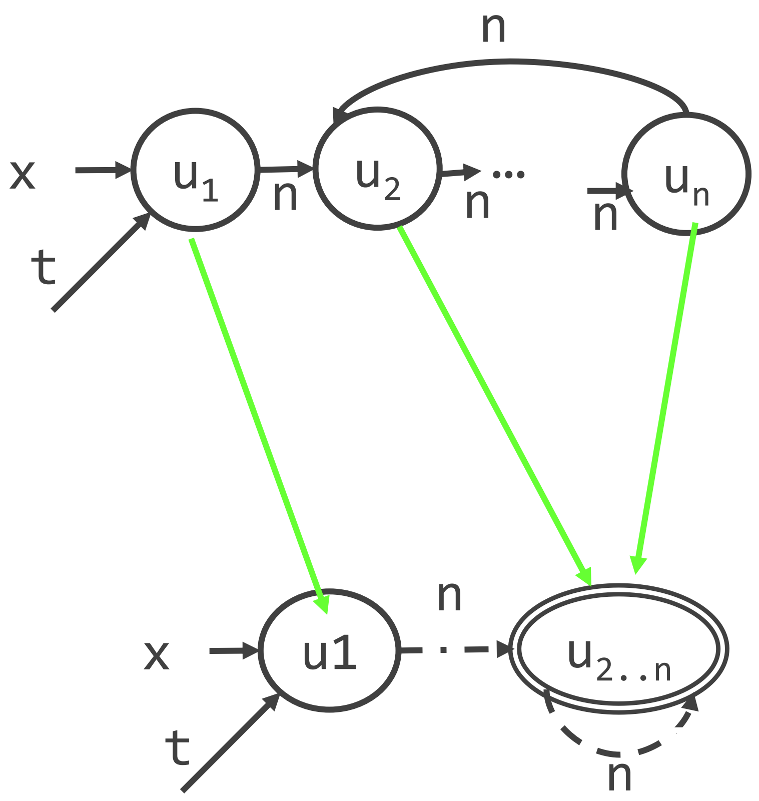
c[x]()=0

From the abstract graph alone we cannot tell there are no cycles, but the predicate tells us this is the case.

Cyclicity predicate

$c[x]() = \exists v_1, v_2 : x(v_1) \wedge n^*(v_1, v_2) \wedge n^+(v_2, v_2)$

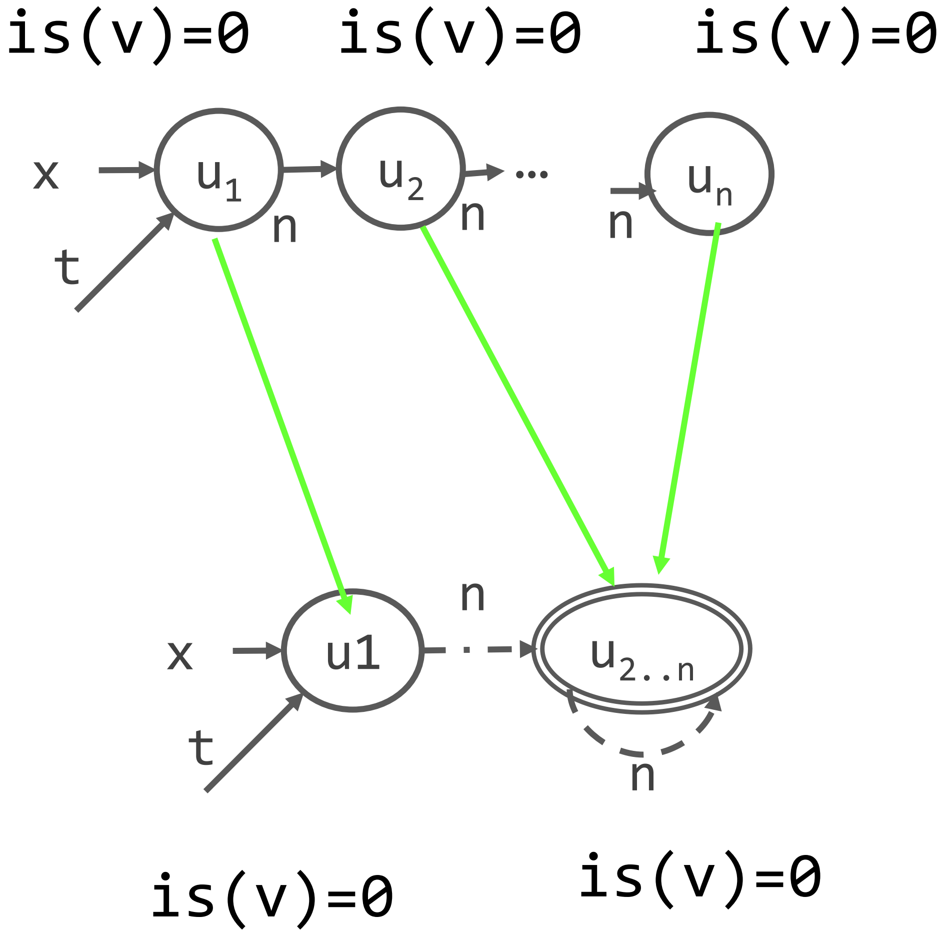
c[x]()=1



c[x]()=1

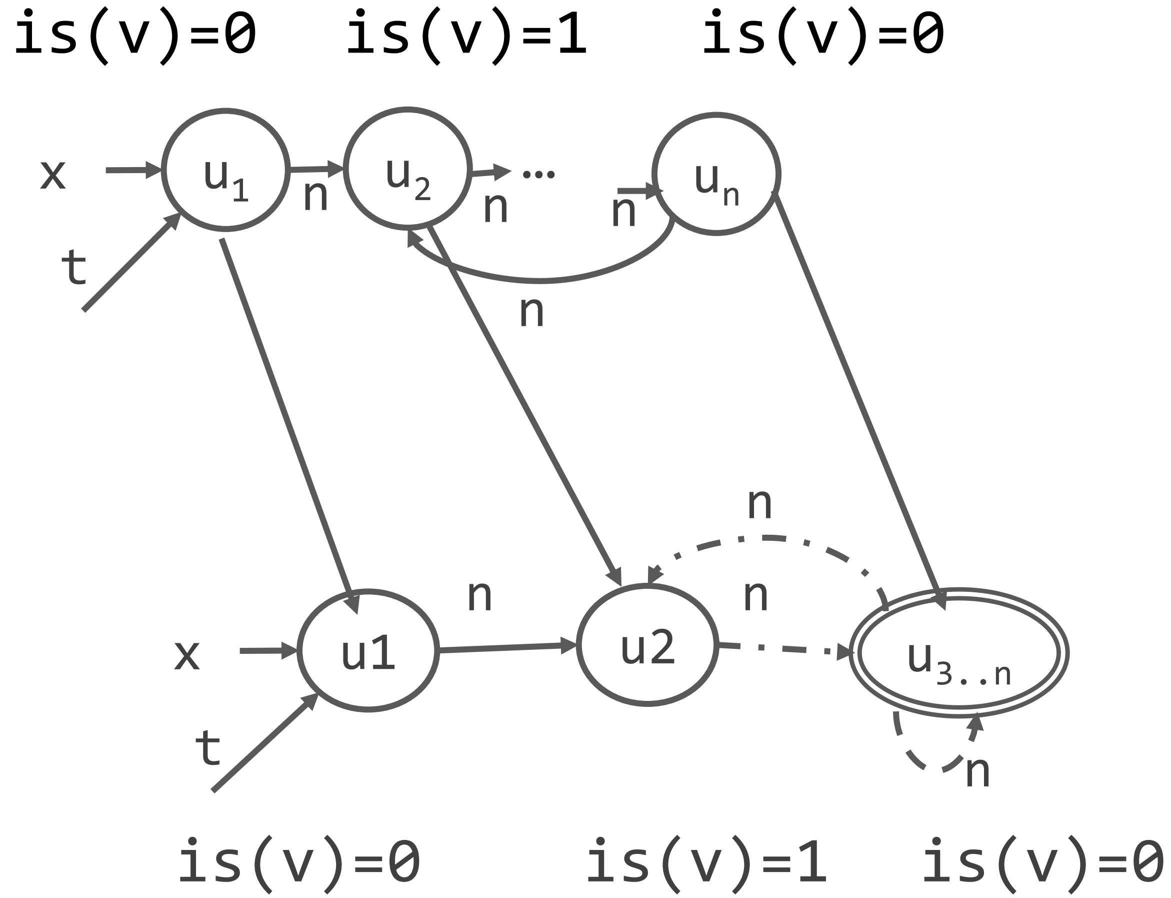
Heap Sharing predicate

$$is(v) = \exists v_1, v_2 : n(v_1, v) \land n(v_2, v) \land v_1 \neq v_2$$



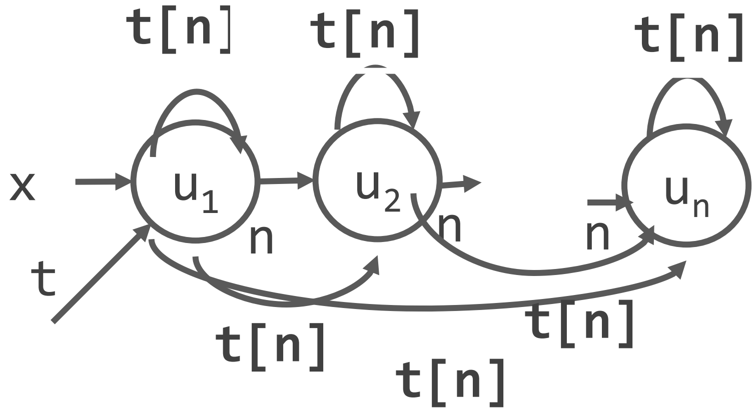
Heap Sharing predicate

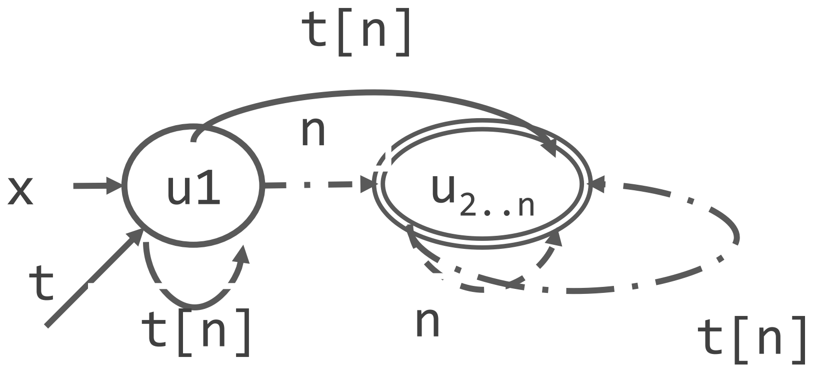
$$is(v) = \exists v_1,v_2: n(v_1,v) \wedge n(v_2,v) \wedge v_1 \neq v_2$$



Reachability predicate

`t[n](v1, v2) = n*(v1,v2)`





Additional Instrumentation predicates

reachable-from-variable-$x(v)$

$c_{fb}(v) = \forall v_1\text{: } f(v, v_1) \; \underline{\Omega}\, b(v_1, v)$

tree(v)

dag(v)

inOrder(v) =

$\forall v_1\text{: } n(v, v_1) \rightarrow dle(v,v_1)$

Instrumentation (Summary)

Refines the abstraction

Adds global invariants

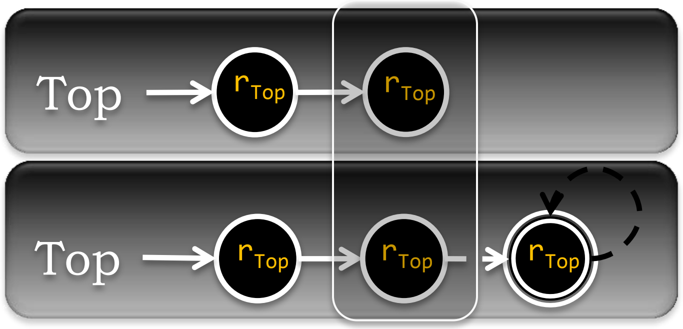
But requires update-formulas
(generated automatically in TVLA2)

Partial Concretization (focus)

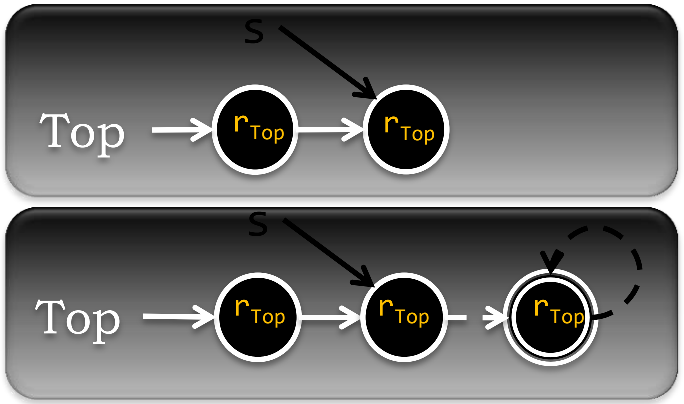
Helpful in making transfer functions more precise

Expand an abstract heap into a collection of more concrete

Partial Concretization Based on Transformer ($s = Top \rightarrow n$)



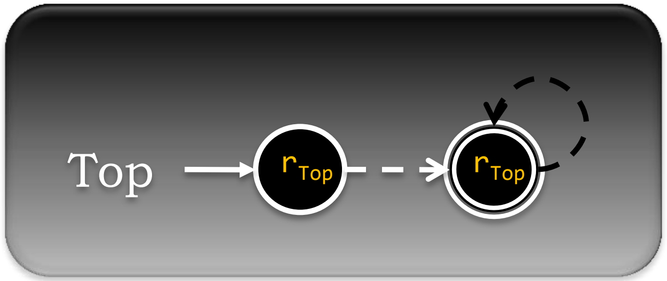
Abstract Semantics



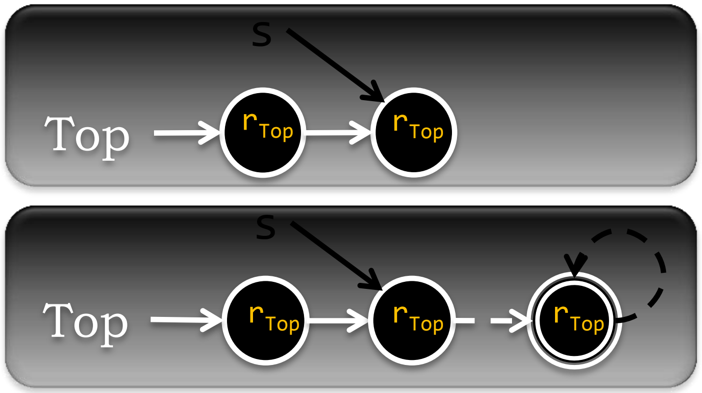
$s'(v) = \exists v1 : Top(v1) \wedge n(v1, v)$

Partial Concretization

Canonical Abstraction



Abstract Semantics


s
Top
r Top
r Top
s
Top
r Top
r Top
r Top