Lecture 08(a) – Shape Analysis – continued
Lecture 08(b) – Typestate Verification
Lecture 08(c) – Predicate Abstraction

PROGRAM ANALYSIS & SYNTHESIS

Eran Yahav

Previously

- Shape Analysis

Today

- Shape Analysis – continued
- Concurrent Shape Analysis
- Typestate Verification
- Predicate Abstraction (optimistically!)

Shape Analysis

Automatically verify properties of programs manipulating dynamically allocated storage

Identify all possible shapes (layout) of the heap

Shape Analysis via 3-valued Logic

1) Abstraction
- 3-valued logical structure
- canonical abstraction

2) Transformers
- via logical formulae
- soundness by construction
  - embedding theorem, [SRW02]

Collecting State Semantics

$$
CSS[v] = \begin{cases}
\{<\emptyset,\emptyset>\} & \text{if } v = \text{entry} \\[2ex]
\bigcup\limits_{\substack{(w,v) \in E(G), \\ w \in Assignments(G)}} \{ [\![st(w)]\!](S) \mid S \in CSS[w] \} \cup \\
\bigcup\limits_{\substack{(w,v) \in E(G), \\ w \in Skip(G)}} \{ S \mid S \in CSS[w] \} \cup \\
\bigcup\limits_{(w,v) \in True\text{-}Branches(G)} \{ S \mid S \in CSS[w] \text{ and } S \vDash cond(w) \} \cup & \text{othrewise} \\
\bigcup\limits_{(w,v) \in False\text{-}Branches(G)} \{ S \mid S \in CSS[w] \text{ and } S \vDash \neg cond(w) \}
\end{cases}
$$

empty universe

empty interpretation

all predecessors of v in the CFG

set of states at node w

effect of the statement at node w

set of states at node v

Collecting Semantics

- At every program point – a potentially infinite set of two-valued logical structures
- Representing (at least) all possible heaps that can arise at the program point

- Next step:
  find a bounded abstract representation

3-Valued Logical Structures

- A set of individuals (nodes) *U*
- Relation meaning
  - Interpretation of relation symbols in *P*

    $p^0() \rightarrow \{0,1, 1/2\}$

    $p^1(v) \rightarrow \{0,1, 1/2\}$

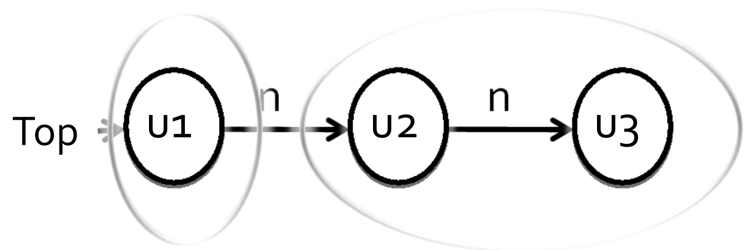
    $p^2(u,v) \rightarrow \{0,1, 1/2\}$
- A join semi-lattice: $0 \sqcup 1 = 1/2$

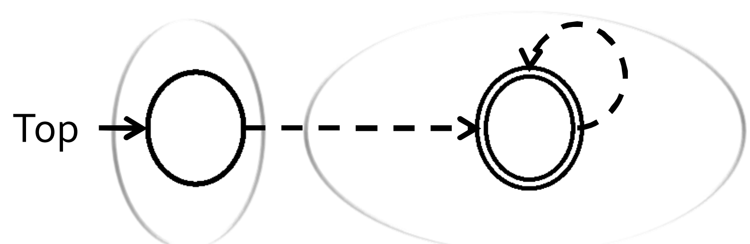
Property Space

- 3-struct[P] = the set of 3-valued logical structures over a vocabulary (set of predicates) P

- Abstract domain
  - $\wp$(3-Struct[P])
  - $\sqsubseteq$ is $\subseteq$
    - We will see alternatives later (maybe)

Canonical Abstraction





Canonical Abstraction (β)

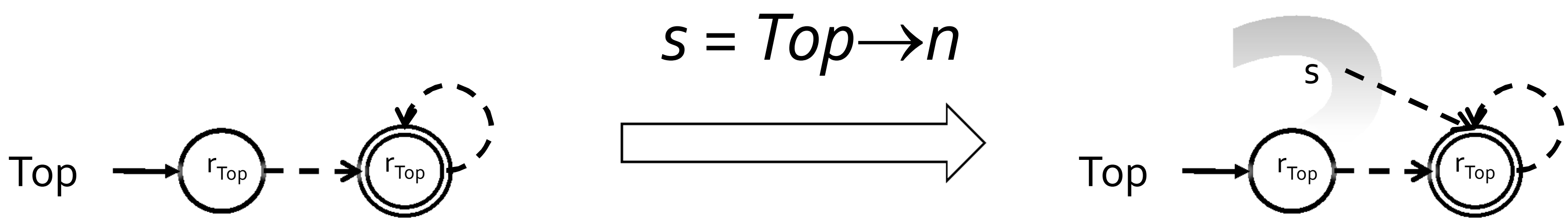
- Merge all nodes with the same unary predicate values into a single summary node
- Join predicate values

$$\iota'(u'_1,\ldots,u'_k) = \sqcup \{\iota(u_1,\ldots,u_k) \mid f(u_1)=u'_1,\ldots,f(u_k)=u'_k\}$$

- Converts a state of arbitrary size into a 3-valued abstract state of bounded size

- $\alpha(C) = \sqcup \{\beta(c) \mid c \in C\}$

Abstract Semantics

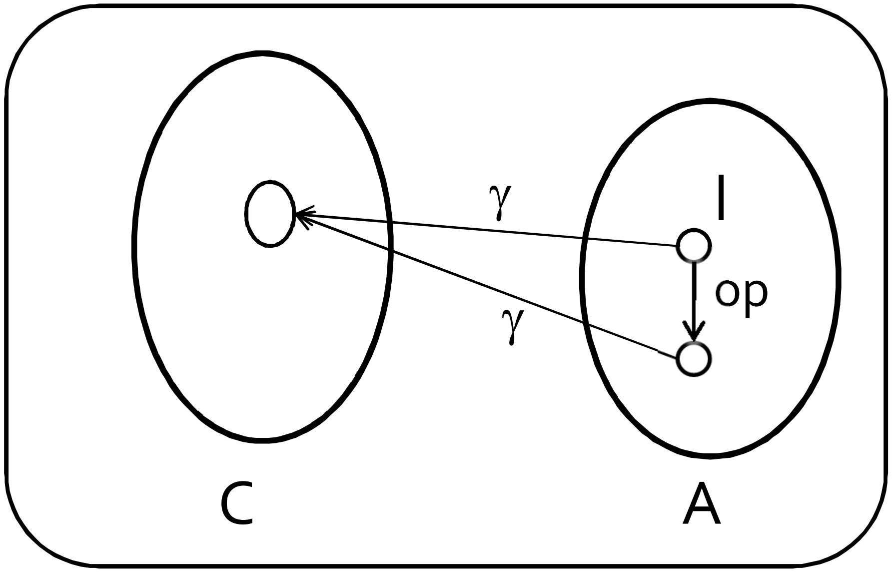


⟦`s = Top->n`⟧

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

Semantic Reduction

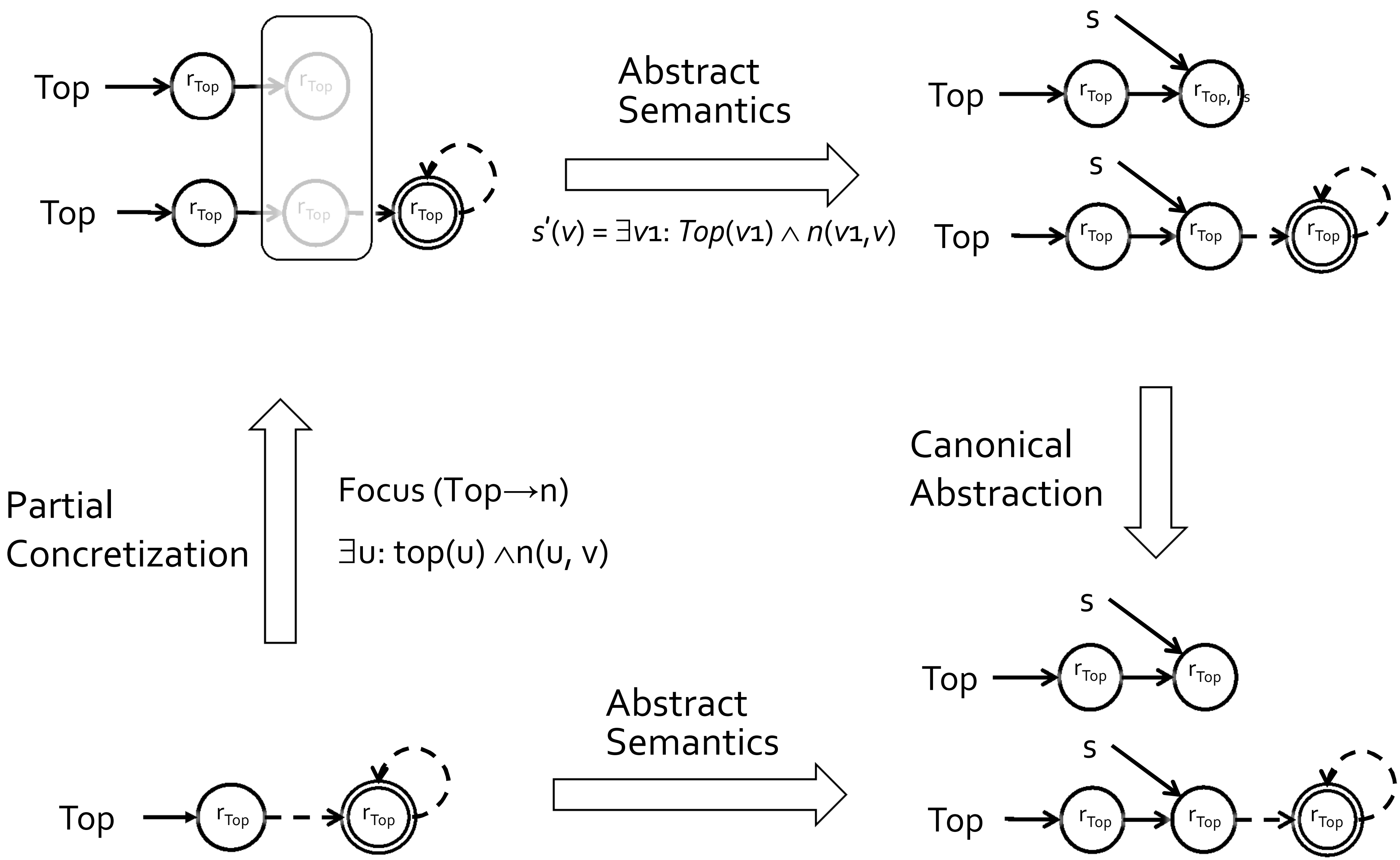
- Improve the precision of the analysis by recovering properties of the program semantics
- A Galois connection (C, α, γ, A)
- An operation $op: A \rightarrow A$ is a semantic reduction when
  - $\forall l \in L_2 \; op(l) \sqsubseteq l$ and
  - $\gamma(op(l)) = \gamma(l)$



The Focus Operation

- Focus: Formula→($\wp$(3-Struct) $\hookrightarrow$ $\wp$(3-Struct))
- Generalizes materialization
- For every formula φ
  - Focus(φ)(X) yields structure in which φ evaluates to a definite values in all assignments
  - Only maximal in terms of embedding
  - Focus(φ) is a semantic reduction
  - But Focus(φ)(X) may be undefined for some X

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



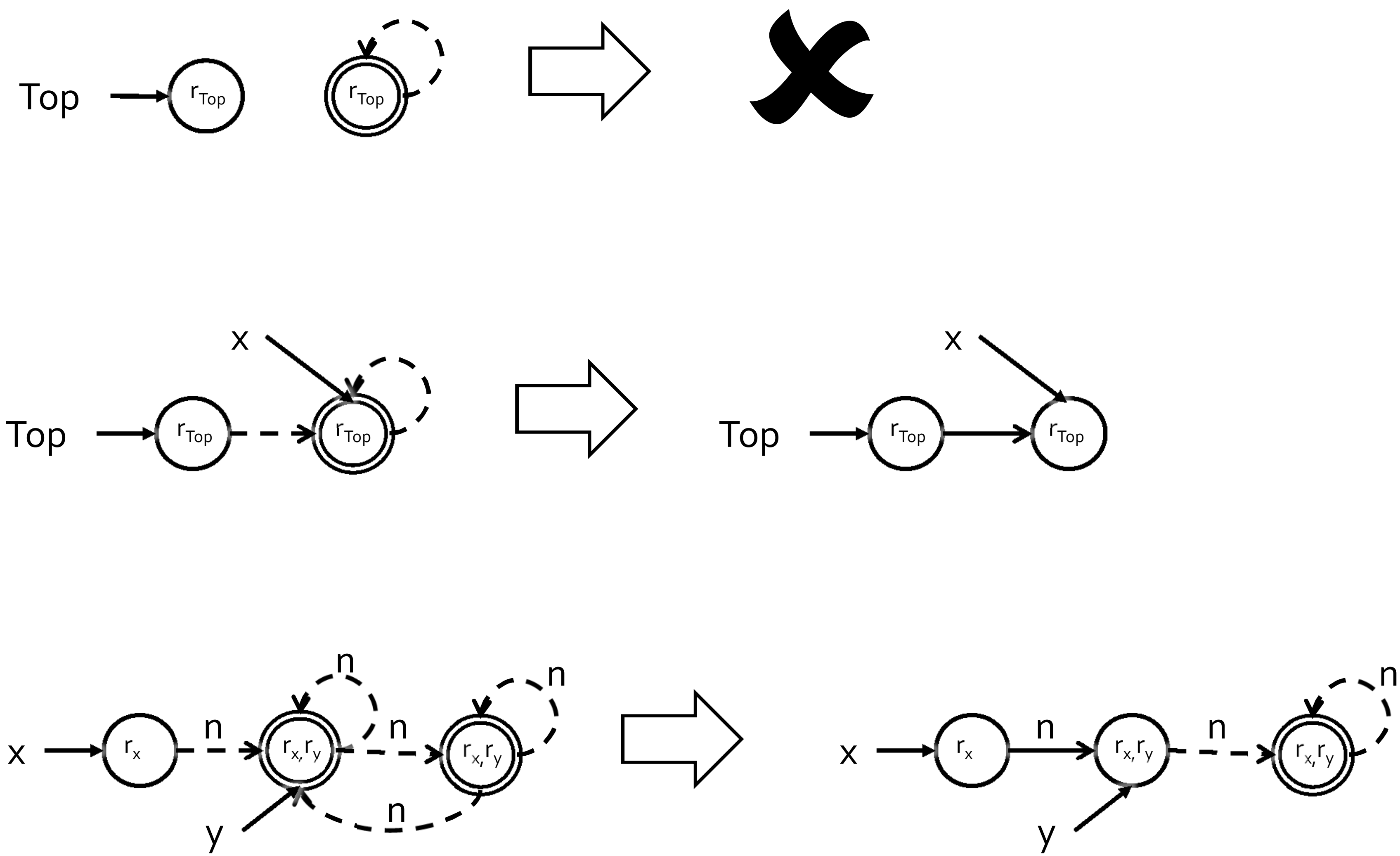
Partial Concretization

- Locally refine the abstract domain per statement
- Soundness is immediate
- Employed in other shape analysis algorithms [Distefano et.al., TACAS'06, Evan et.al., SAS'07, POPL'08]
- Emplyed in other analysis algorithms [Typestate verification, ISSTA'06]

The Coercion Principle

- Another Semantic Reduction
- Can be applied after Focus or after Update or both
- Increase precision by exploiting structural properties possessed by all stores (Global invariants)
- Structural properties captured by constraints
- Apply a constraint solver

Apply Constraint Solver



Sources of Constraints

- Properties of the operational semantics
- Domain specific knowledge
  - Instrumentation predicates
- User supplied

Example Constraints

$x(v_1) \wedge x(v_2) \rightarrow eq(v_1, v_2)$

$n(v, v_1) \wedge n(v,v_2) \rightarrow eq(v_1, v_2)$

$n(v_1, v) \wedge n(v_2,v) \wedge \neg eq(v_1, v_2) \leftrightarrow is(v)$

$n^*(v_3, v_4) \leftrightarrow t[n](v_1, v_2)$

Abstract Transformers: Summary

- Kleene evaluation yields sound solution
- Focus is a statement-specific partial concretization
- Coerce applies global constraints

Abstract Semantics

$$
SS[v] = \begin{cases}
\{ \langle \emptyset, \emptyset \rangle \} & \text{if } v = \text{entry} \\[1em]
\bigcup\limits_{\substack{(w,v) \in E(G), \\ w \in Assignments(G)}} \{ t\_embed(coerce(\llbracket st(w) \rrbracket 3(focus_{F(w)}(SS[w])))) \cup \\[1em]
\bigcup\limits_{\substack{(w,v) \in E(G), \\ w \in Skip(G)}} \{ S \mid S \in SS[w] \} \cup & \text{othrewise} \\[1em]
\bigcup\limits_{(w,v) \in True\text{-}Branches(G)} \{ t\_embed(S) \mid S \in coerce(\llbracket st(w) \rrbracket 3(focus_{F(w)}(SS[w]))) \text{ and } S \models 3 \; cond(w) \} \cup \\[1em]
\bigcup\limits_{(w,v) \in False\text{-}Branches(G)} \{ t\_embed(S) \mid S \in coerce(\llbracket st(w) \rrbracket 3(focus_{F(w)}(SS[w]))) \text{ and } S \models 3 \; \neg cond(w) \} \cup
\end{cases}
$$

Recap

- Abstraction
  - canonical abstraction
  - recording derived information

- Transformers
  - partial concretization (focus)
  - constraint solver (coerce)
  - sound information extraction

Stack Push

```
void push (int v) {
 Node *x =
   alloc(sizeof(Node));
```

∃v: x(v)

```
 x→d = v;
```

∃v: x(v)

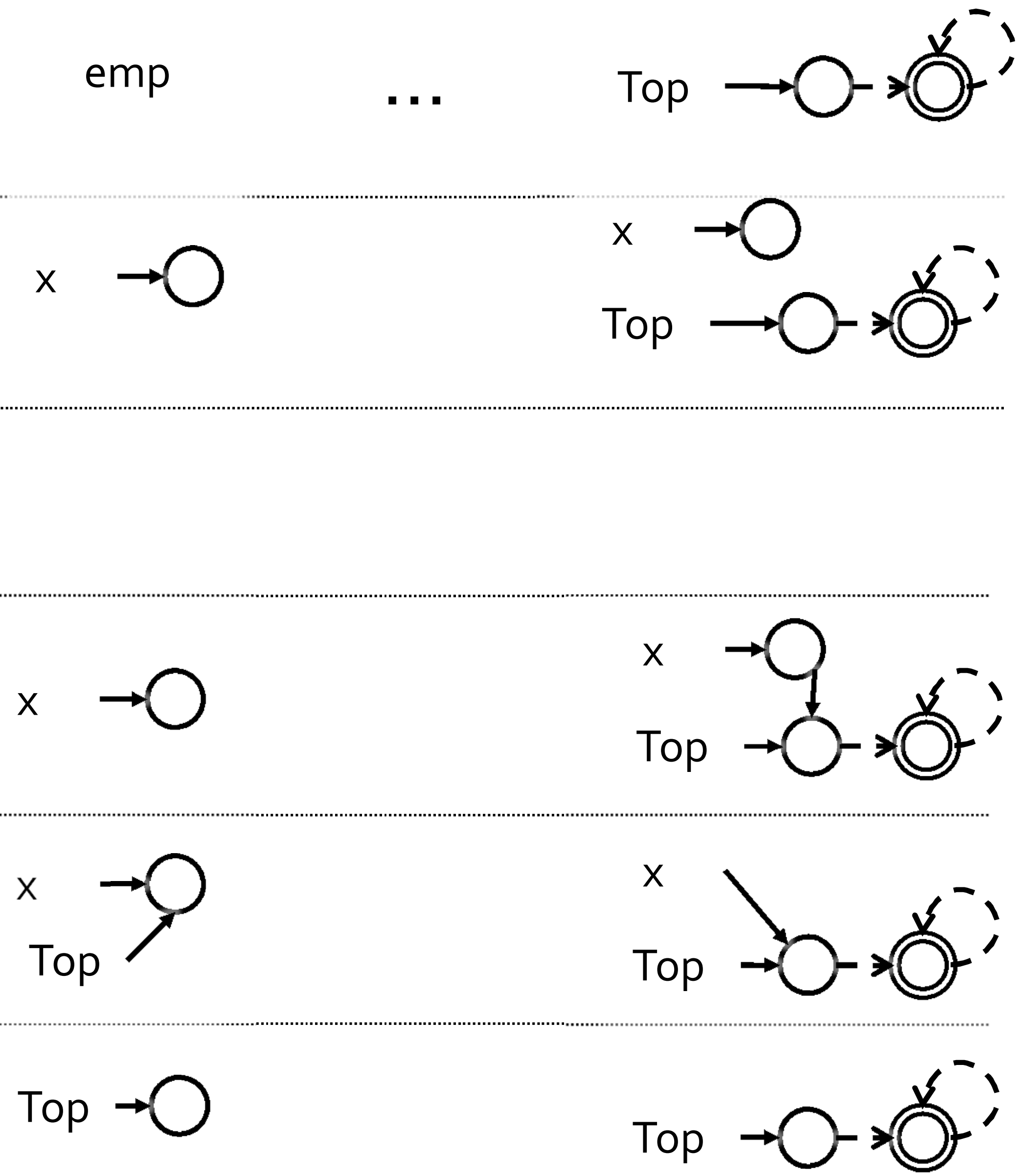
```
 x→n = Top;
```

```
 Top = x;
```

$\neg\exists v_1, v_2 : n(v_1, v_2) \wedge Top(v_2)$

```
}
```

$\forall v : \neg c(v)$



Non-blocking Stack [Treiber 1986]

```
#define EMPTY -1

typedef int data_type;

typedef struct node t {
    data_type d;
    struct node t *n
} Node;

typedef struct stack t {
    struct node t *Top;
} Stack;
```

```
void push(Stack *S, data_type v) {
  Node *x = alloc(sizeof(Node));
  x->d = v;
  do {
    Node *t = S->Top;
    x->n = t;
  } while (!CAS(&S->Top,t,x));
}


data_type pop(Stack *S){
  do {
    Node *t = S->Top;
    if (t == NULL)
      return EMPTY;
    Node *s = t->n;
    data_type r = t->d;
  } while (!CAS(&S->Top,t,s));
  return r;
}
```

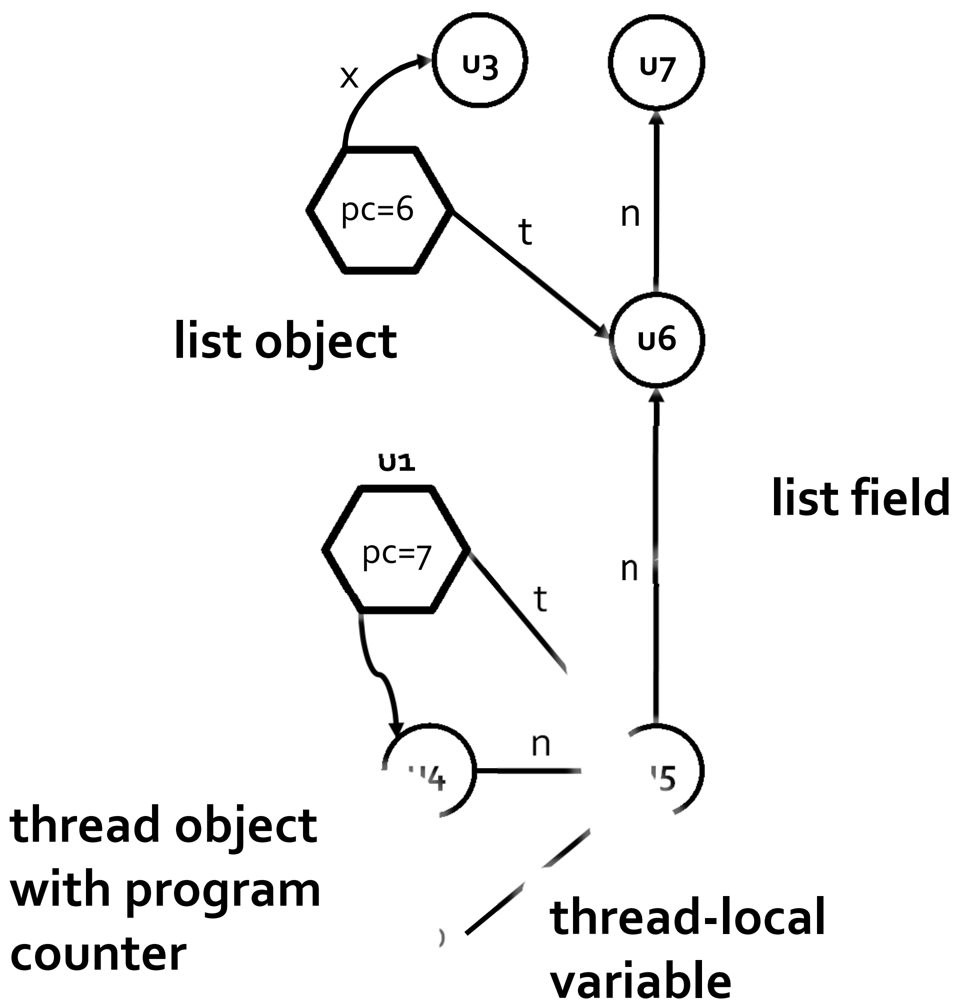
Concurrent Shape Analysis

- a thread is represented as a thread object
- add predicates to vocabulary

- Recipe
  1) abstraction: canonical abstraction
  2) transformers: interleaving + as before

- Bounded threads
  - Static thread names
- Unbounded threads
  - thread objects abstracted via canonical abstraction

Concrete State



U = { u1, u2, u3,…,u7 }

isThread = { u1,u2 }

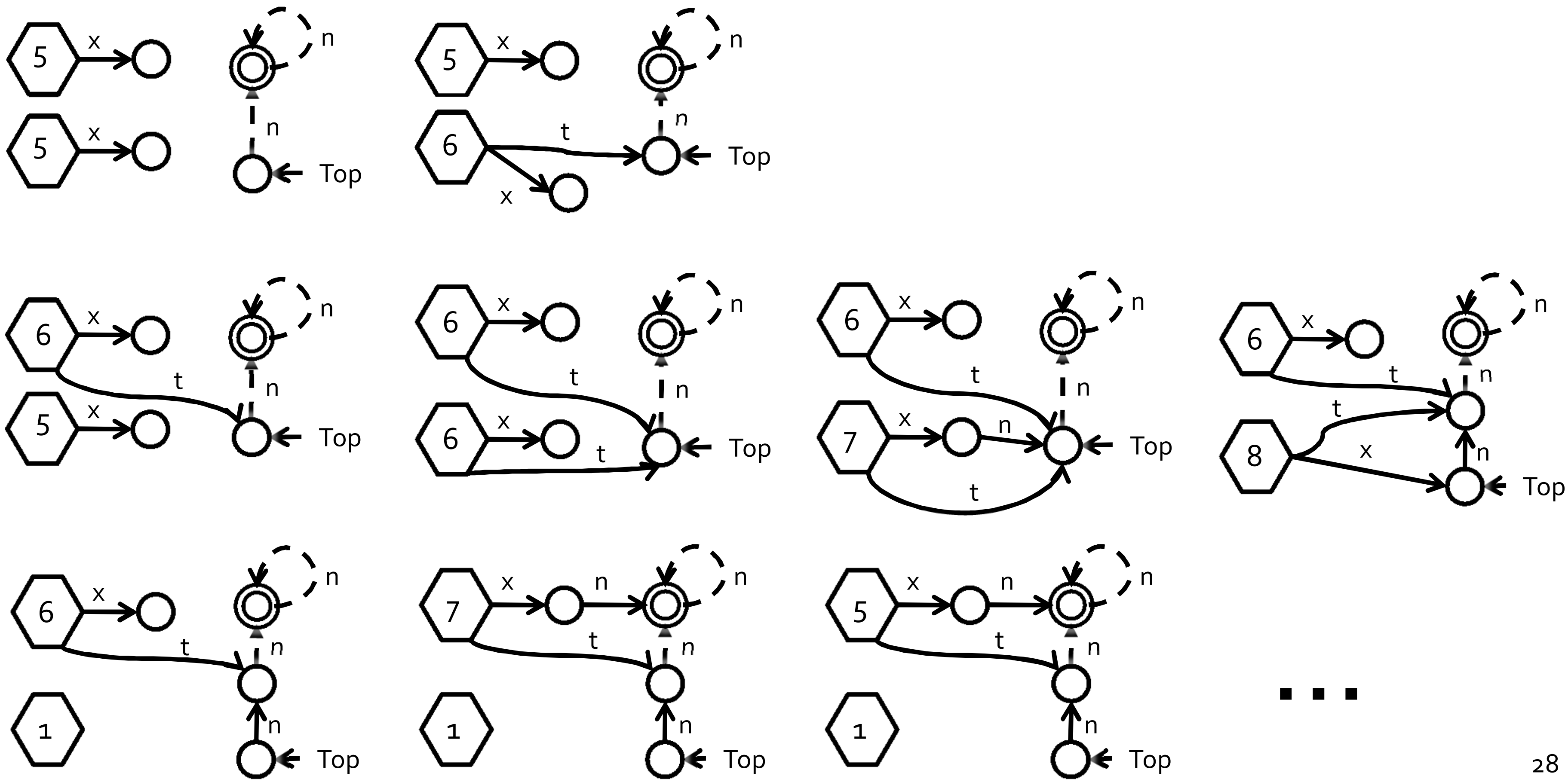
at[pc=1] = { }
…
at[pc=6] = { u3 }
at[pc=7]= { u1 }

Top = { u5 }
…
x = { (u1,u4), (u2,u3)}
t={ (u1,u5),(u2,u6)}
n = { (u5,u6), (u6,u7) }
t 1= { u1 } t2 = { u2 }

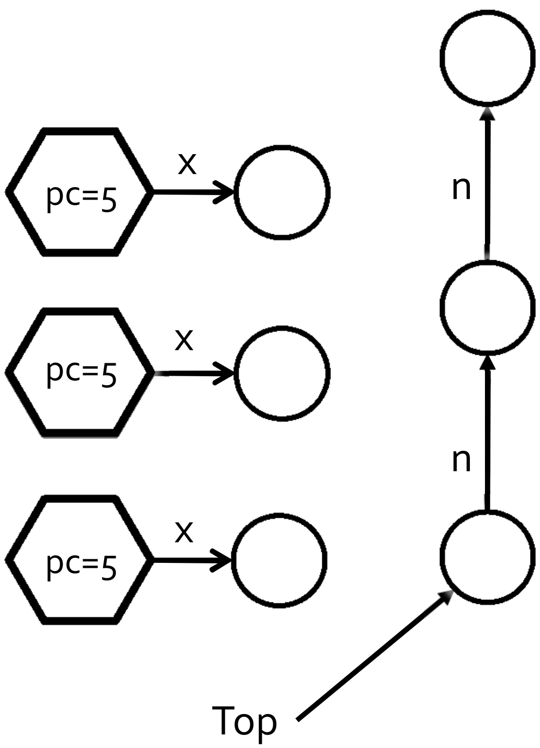
Exploration

```
[1] void push(Stack *S, data_type v) {
[2]   Node *x = alloc(sizeof(Node));
[3]   x->d = v;
[4]   do {
[5]     Node *t = S->Top;
[6]     x->n = t;
[7]   } while (!CAS(&S->Top,t,x));
[8] }
```

$\forall v: \neg c(v)$

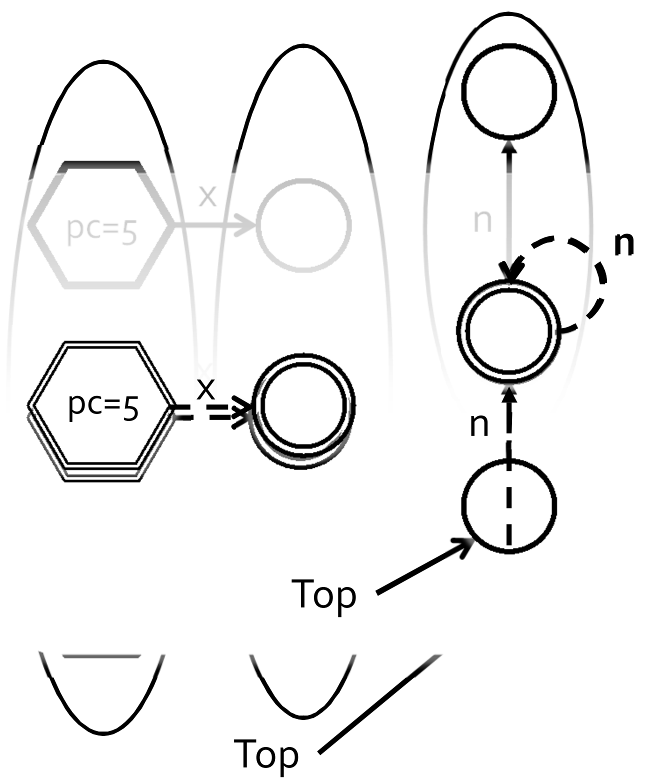


Representing an Unbounded Number of Threads



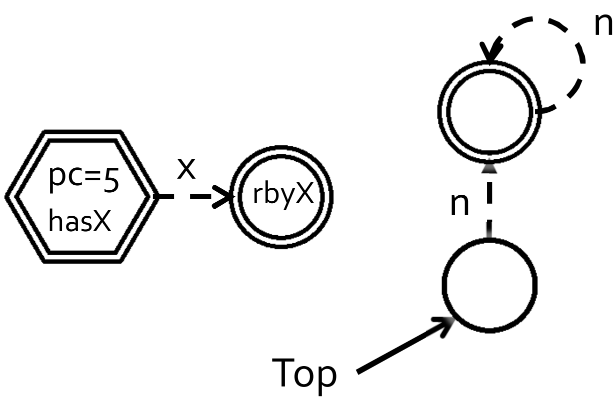

```
void push(Stack *S, data_type v) {
  Node *x = alloc(sizeof(Node));
  x->d = v;
  do {
    Node *t = S->Top;
    x->n = t;
  } while (!CAS(&S->Top,t,x));
}
```

Representing an Unbounded Number of Threads

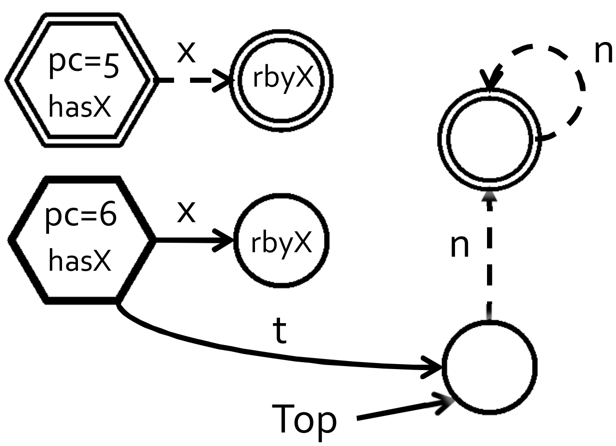



```
void push(Stack *S, data_type v) {
  Node *x = alloc(sizeof(Node));
  x->d = v;
  do {
    Node *t = S->Top;
    x->n = t;
  } while (!CAS(&S->Top,t,x));
}
```

Abstract Semantics



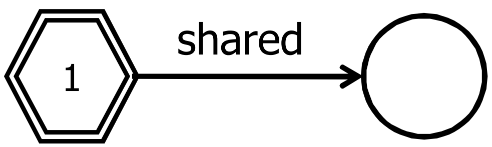

```
[1] void push(Stack *S, data_type v) {
[2]   Node *x = alloc(sizeof(Node));
[3]   x->d = v;
[4]   do {
[5]     Node *t = S->Top;
[6]     x->n = t;
[7]   } while (!CAS(&S->Top,t,x));
[8] }
```



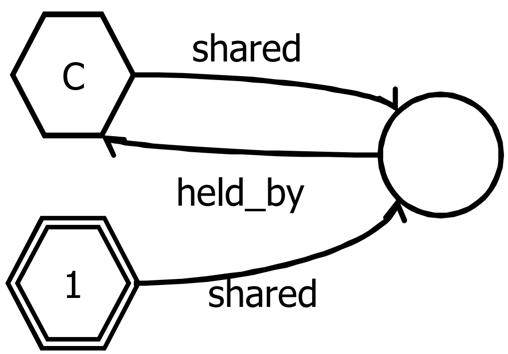
Example - Mutual Exclusion

```
[1] while (true) {
[2]  lock(shared)
[C]  // critical actions
[3]  unlock(shared)
[4] }
```

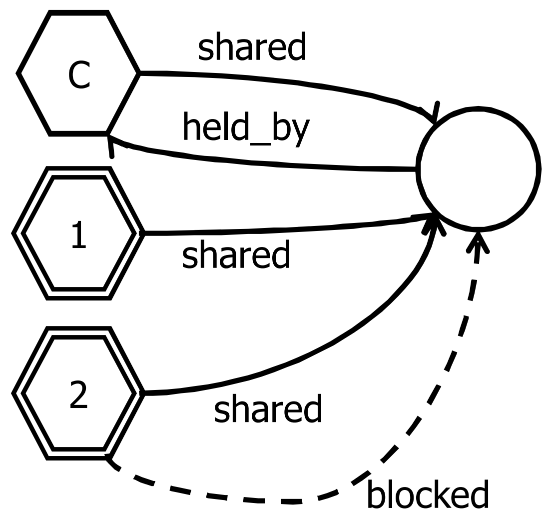
$\forall t_1, t_2: (t_1 \neq t_2) \rightarrow \neg(at[pc=c](t_1) \wedge at[pc=c](t_2))$



Initial configuration



A thread enters the critical section



Other threads may be blocked or just beginning execution

Recap

```
#define EMPTY -1

typedef int data_type;

typ

} N

typ

} Stack;
```

```
[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));

[9]  data_type pop(Stack *S){
[10]     do {
[11]         Node *t = S->Top;
```

✓ No null dereferences

✓ Structural shape invariants

Linearizability

✓ Dynamic Allocation

✓ Destructive Updates

✓ Concurrency