Synthesis with Abstract Interpretation

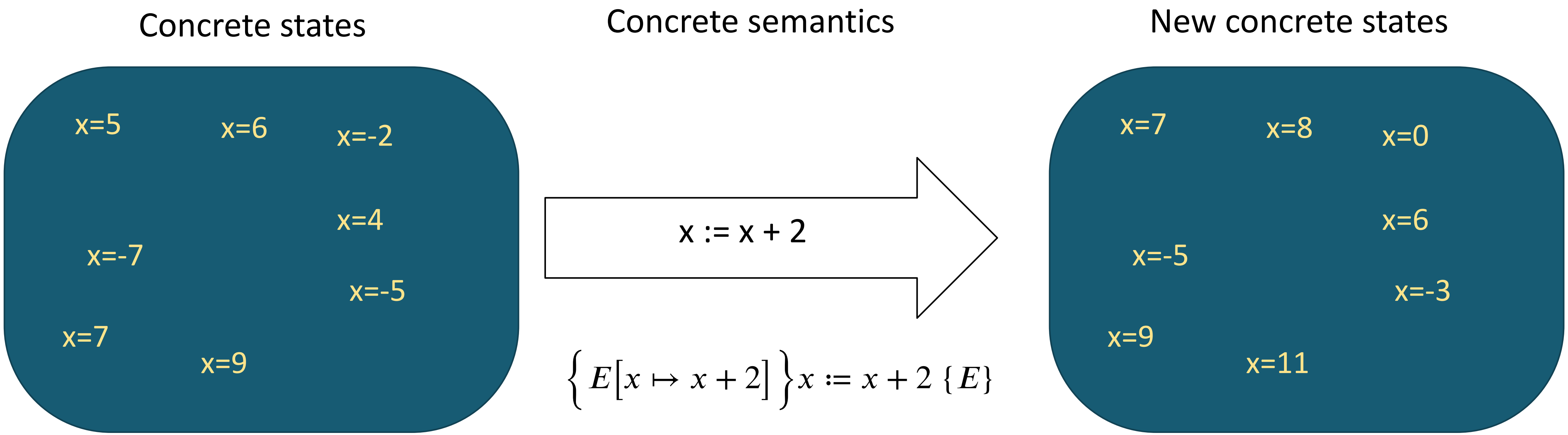
with inputs from Armando Solar-Lezama

History

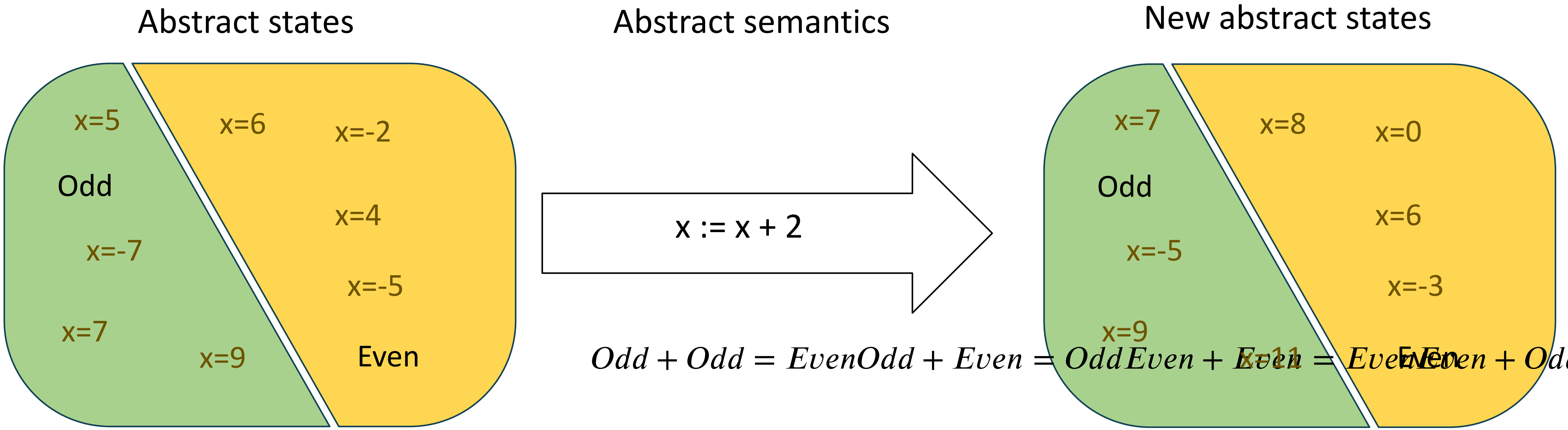
POPL 77 paper by Patrick Cousot and Radhia Cousot

- Brings together ideas from the compiler optimization community with ideas in verification
- Provides a clean and general recipe for building analyses and reasoning about their correctness

Key idea 1: Abstract domain

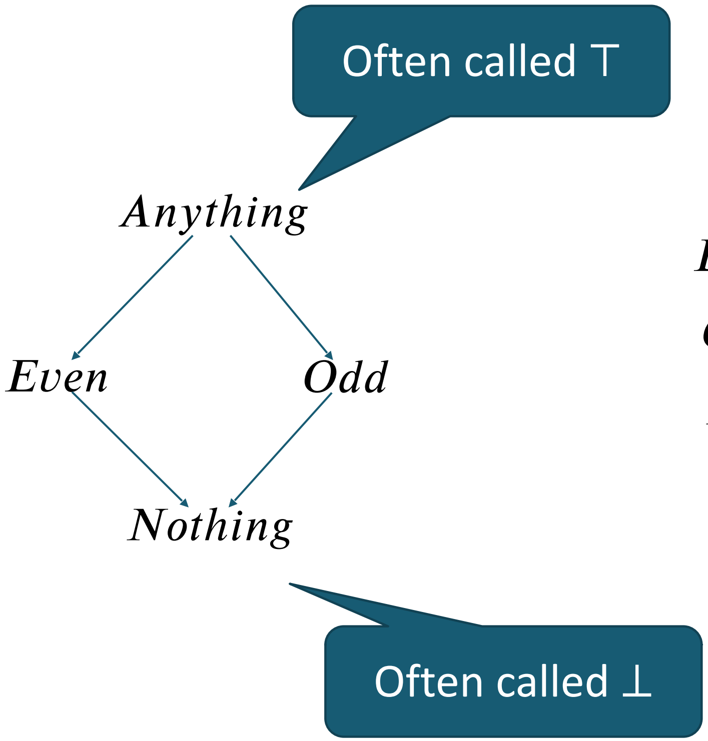


Key idea 1: Abstract domain



$Odd + Odd = Even \quad Odd + Even = Odd \quad Even + Even = Even \quad Even + Odd = O$

Abstract values form a lattice



$$Even \sqsubseteq Anything$$

$$Odd \sqsubseteq Anything$$

$$Nothing \sqsubseteq Even$$

$$Nothing \sqsubseteq Odd$$

Abstract Domain

An abstract domain is a lattice

- Although some analysis relax this restriction.
- Elements in the lattice are called *Abstract Values*

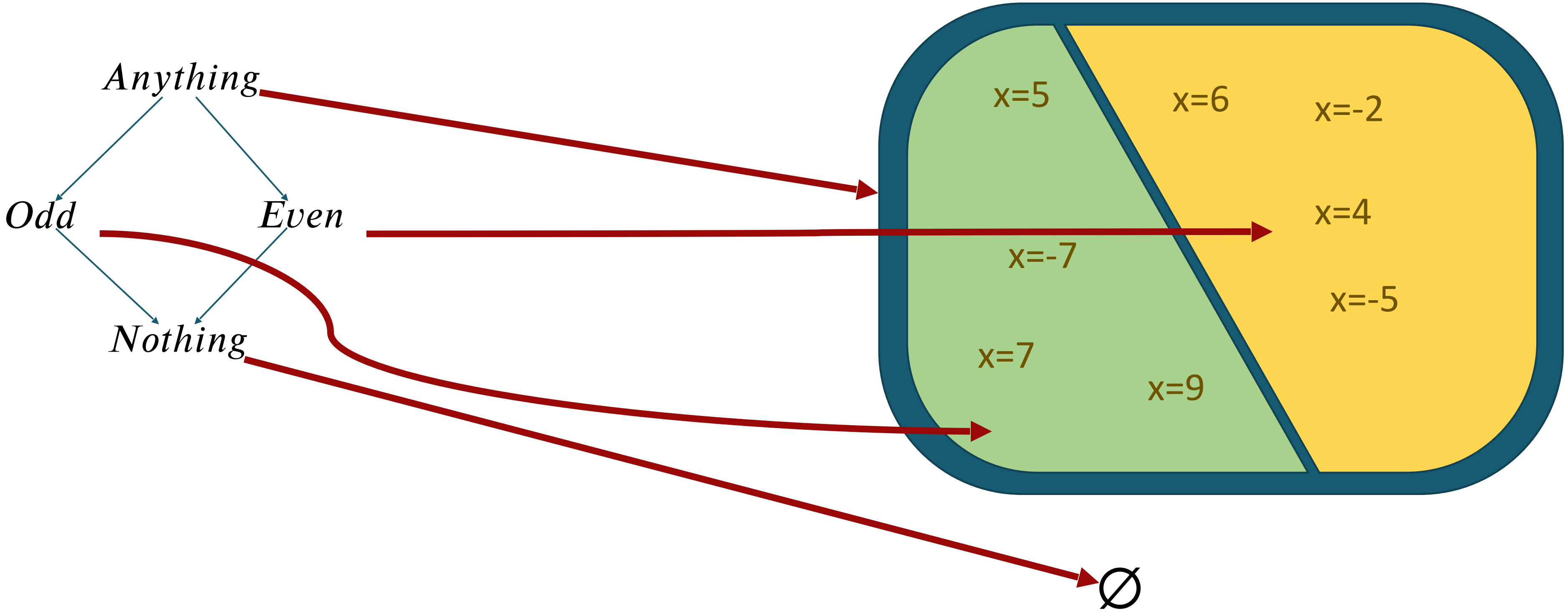
Need to relate elements in the lattice with states in the program

- **Abstraction Function**: $\beta: \mathscr{V} \rightarrow Abs$
  - Maps a value in the program to the "best" abstract value
- **Concretization Function:** $\gamma: Abs \rightarrow \mathscr{P}(\mathscr{V})$
  - Maps an abstract value to a set of values in the program

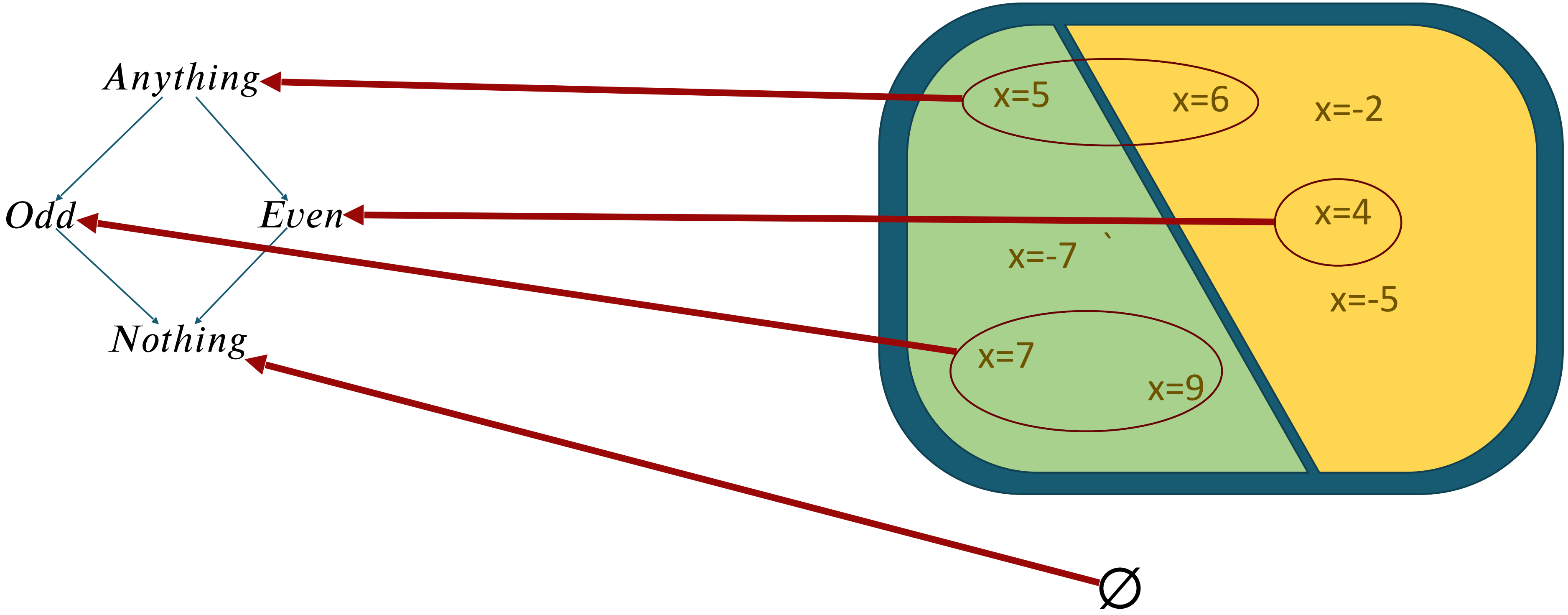
Example:

- Parity Lattice

Concretization



Abstraction



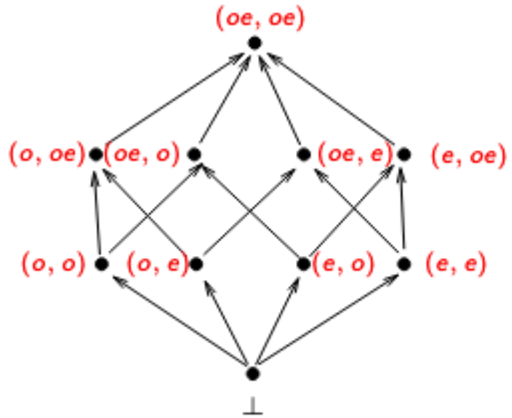
An Abstract Interpretation

Components of an abstract interpretation:

- Set of abstract states D, forming a complete lattice.
- "Concretization" function $\gamma : D \to 2^{State}$, which associates a set of concrete states with each abstract state.
- Transfer function $f_n : D \to D$ for each type of node n, which "interprets" each program statement using the abstract states.

Example: AI

- Abstract lattice D



- Transfer function for an assignment node n: `p := p+q`

$$f_n(s) = \begin{cases} \bot & \text{if } s \text{ is } \bot \\ (o, s[q]) & \text{if } s[\text{p}] \text{ is o and } s[\text{q}] \text{ is e,} \\ & \quad \text{or } s[\text{p}] \text{ is e and } s[\text{q}] \text{ is o} \\ (e, s[q]) & \text{if both } s[\text{p}] \text{ and } s[\text{q}] \text{ are o} \\ & \quad \text{or both } s[\text{p}] \text{ and } s[\text{q}] \text{ are e} \\ (oe, s[q]) & \text{otherwise} \end{cases}$$

- The concretization function γ
  - $\gamma((oe, oe)) = State$, $\gamma(\bot) = \emptyset$, $\gamma((o, oe)) = \{(m, n) \mid m \text{ is odd}\}$
  - $\gamma((o, e)) = \{(m, n) \mid m \text{ is odd and } n \text{ is even}\}, \ldots$

Key idea 2: Abstract Interpretation

Compute an abstract value for every program point

- Abstraction of the set of states possible at that point

Iterate until computation converges

Example

L0

x = _input();

x = x * 2;

y = 0;

L1 while(x < 16){

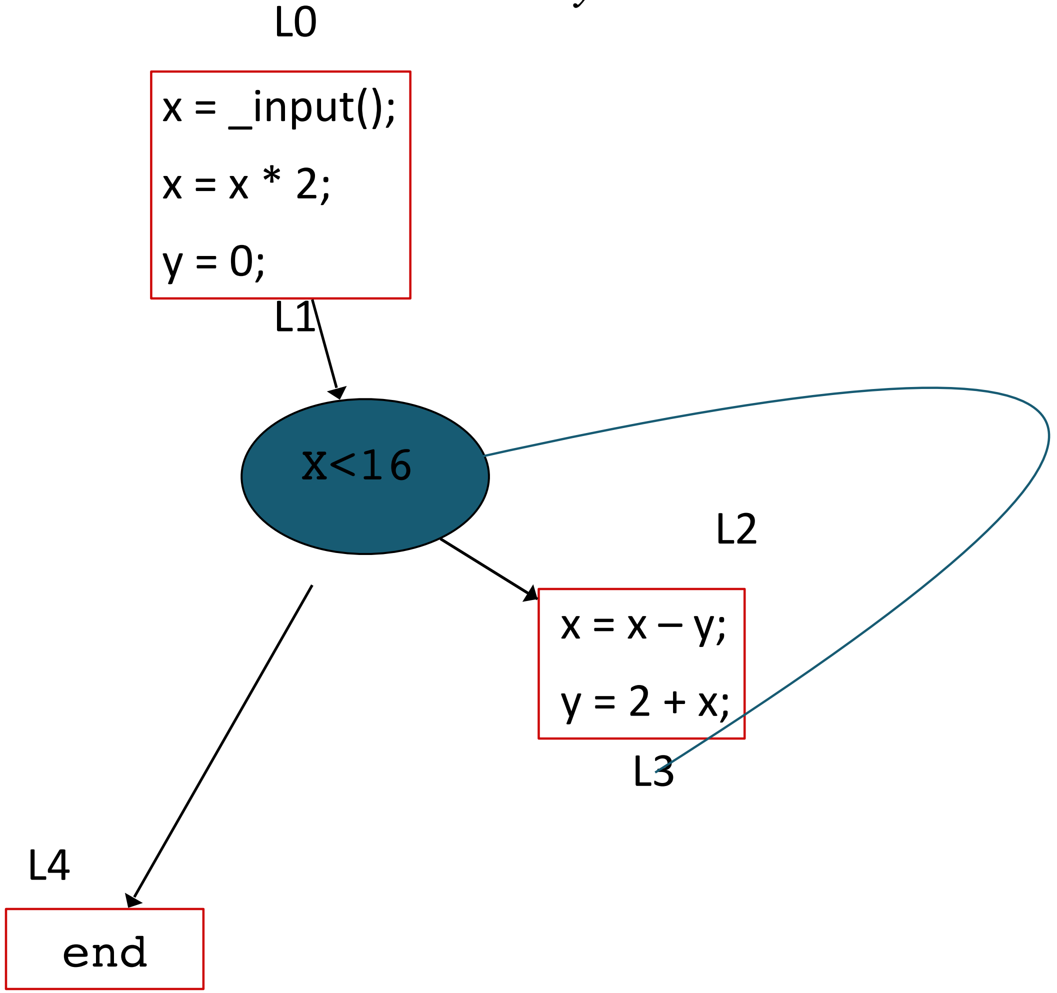
L2 x = x – y;

L3 y = 2 + x;

} assert isEven (y)

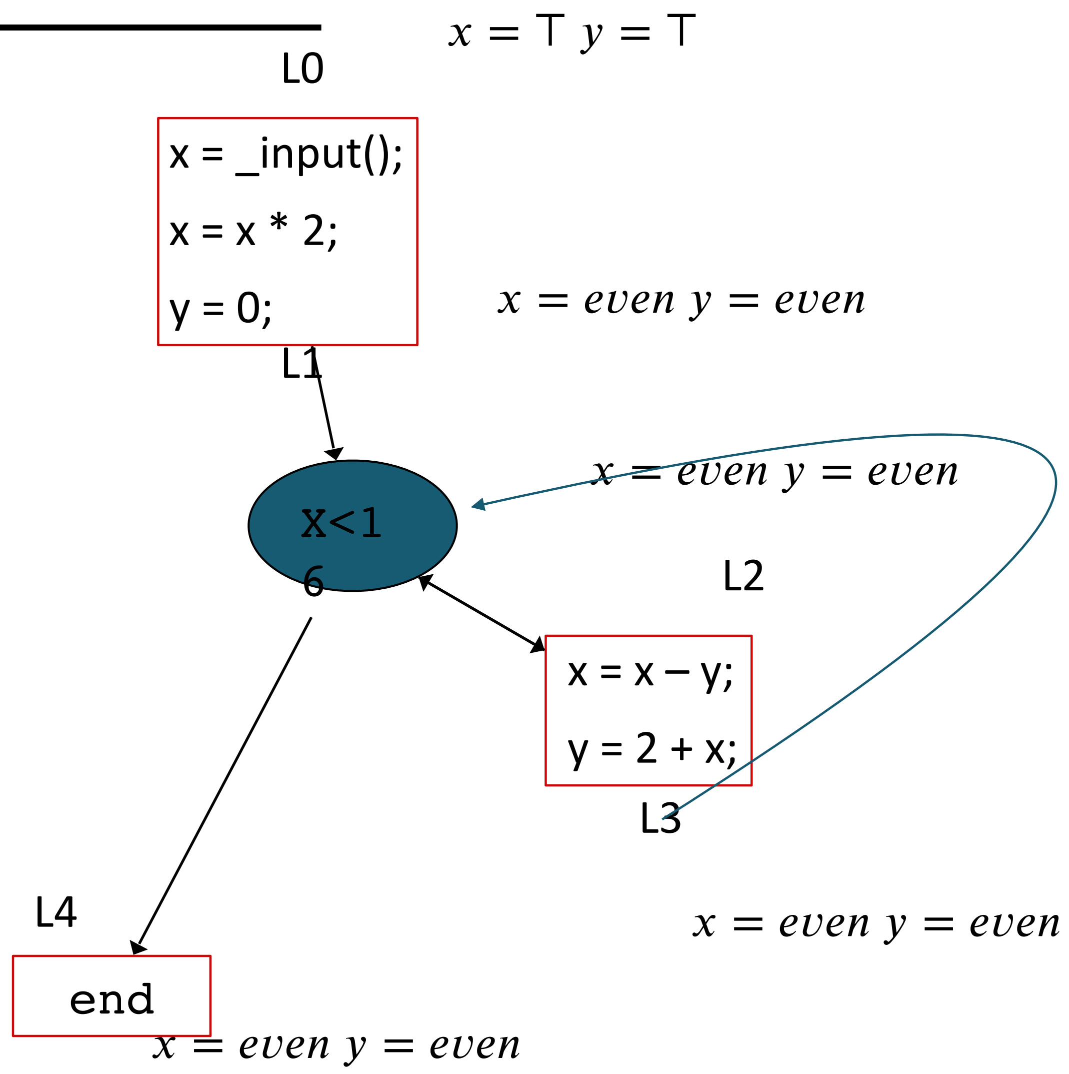
L4

$x = \top \; y = \top$



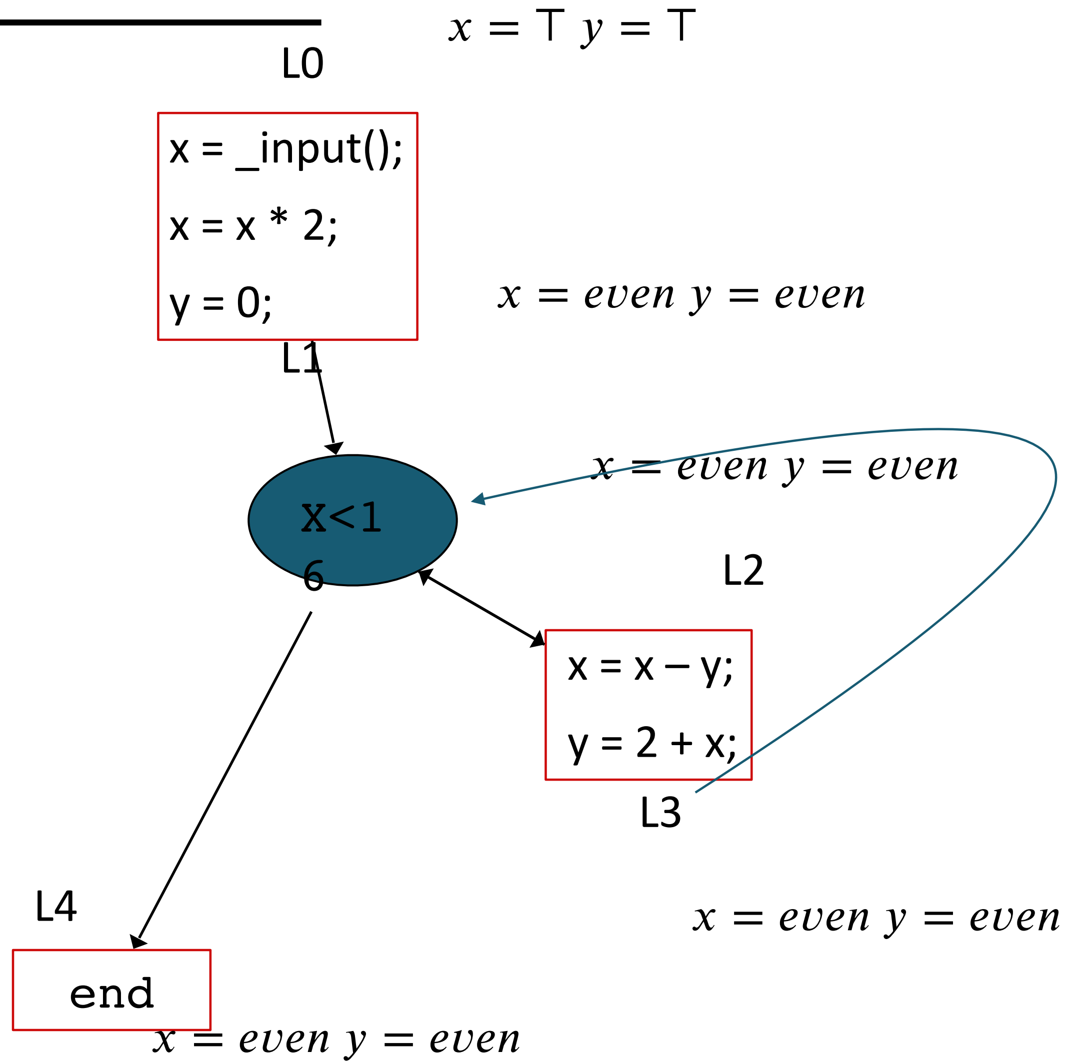
Example

```
L0
        x = _input();
        x = x * 2;
        y = 0;
L1
L2      while(x < 16){
          x = x – y;
L3        y = 2 + x;
        }
L4      assert isEven (y)
```



Example

```
L0
      x = _input();
      x = x * 2;
      y = 0;
L1
L2    while(x < 16){
        x = x – y;
L3      y = 2 + x;
      }
L4    assert isEven (y)
```



Some useful domains

Ranges

- Useful for detecting out-of-bounds errors, potential overflows

Linear relationships between variables

- $a_1 x_1 + a_2 x_2 + \ldots + a_k x_k \geq c$

Problem: Both of these domains have infinite chains!

Widening

Key idea:

- You have been running your analysis for a while
- A value keeps getting “bigger” and “bigger” but refuses to converge
- Just declare it to be $\top$ (or some other big value)

This loses precision

- but it’s always sound

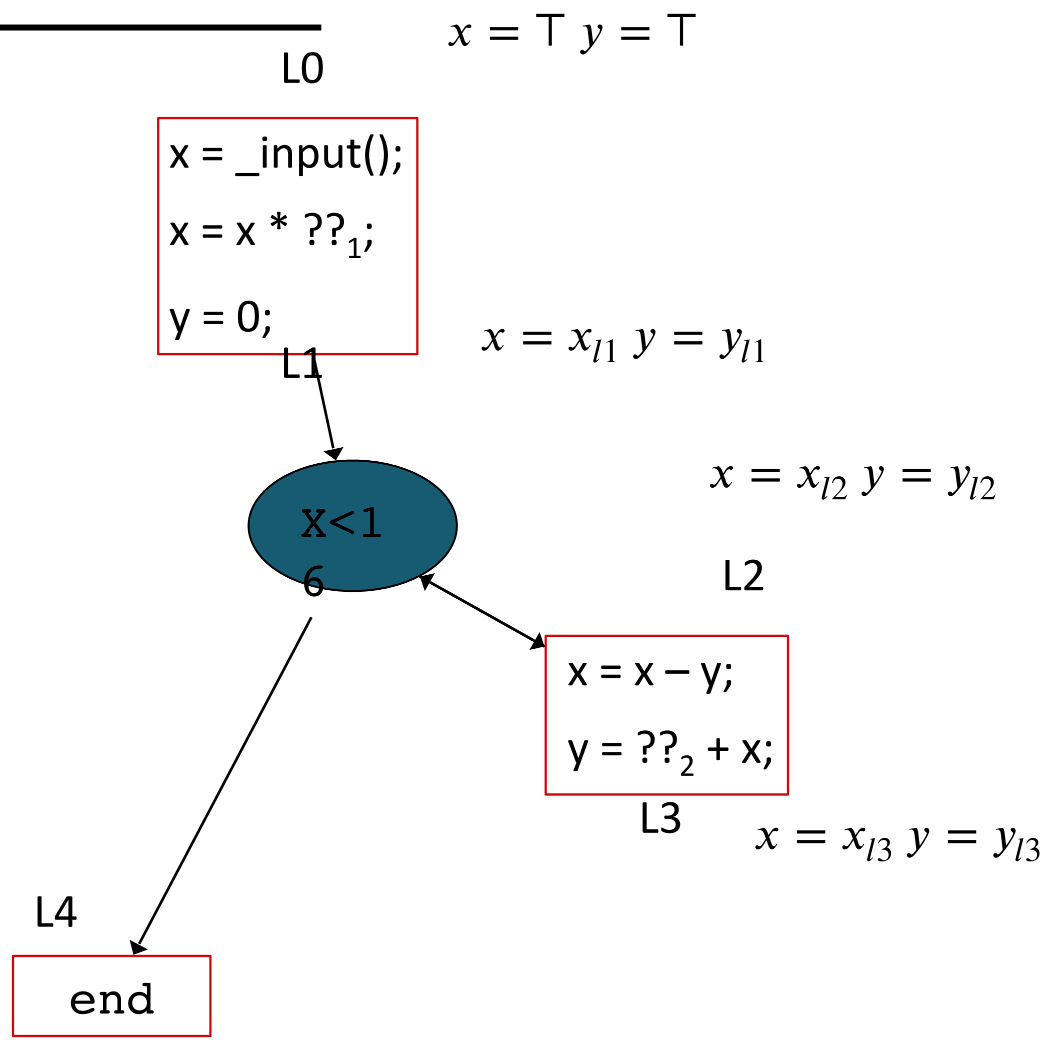
Widening operator: $\nabla : Abs \times Abs \rightarrow Abs$

- $a1 \nabla a2 \sqsupseteq a1, a2$
-

Abstract Interpretation for Synthesis

Example: Simple Case

```
L0
        x = _input();
        x = x * ??1;
L1      y = 0;
L2      while(x < 16){
          x = x – y;
L3        y = ??2 + x;
          assert even(y)
L4      }
```



Synthesis using AI: Simple Case

$$x_{l1} = \top * ??_1 \qquad y_{l1} = even$$
$$x_{l2} = lub(x_{l1}, x_{l3}) \qquad y_{l2} = lub(y_{l1}, y_{l3})$$
$$x_{l3} = x_{l2} - y_{l2} \qquad y_{l3} = ??_2 + x_{l3}$$
$$x_{l3} = even$$

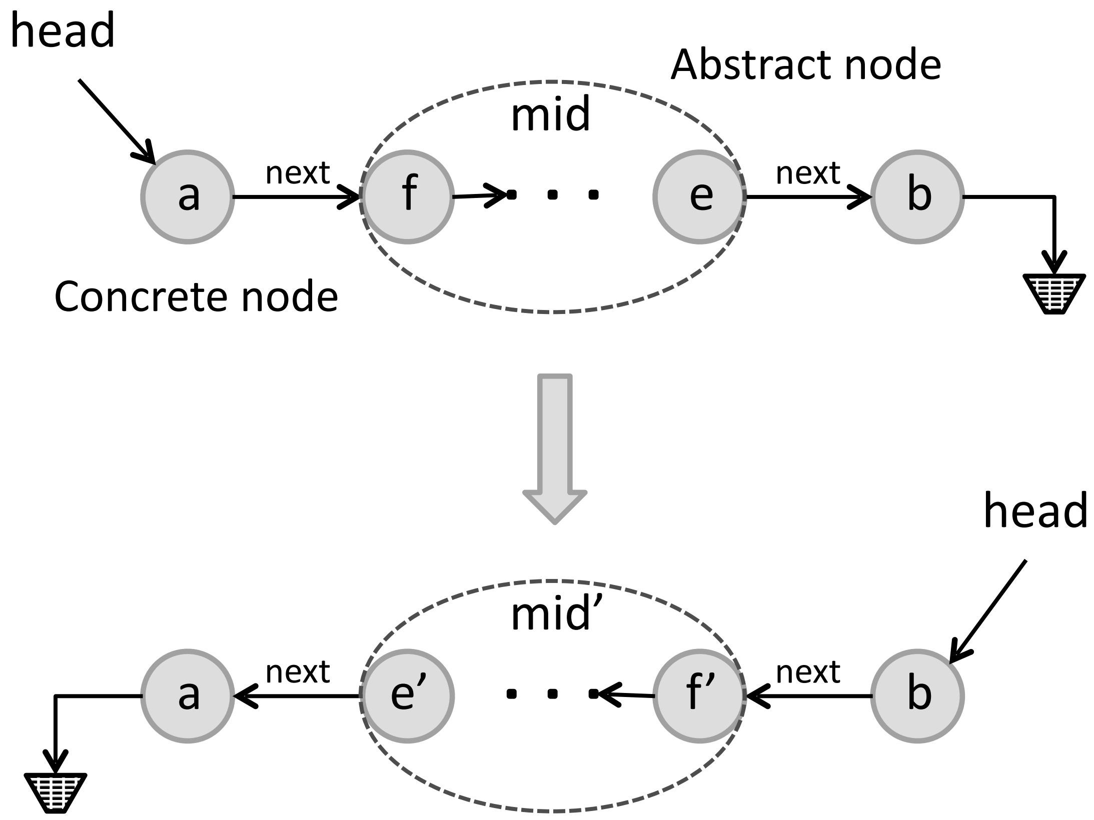
Definitions for +, -, etc.
are given in the abstract domain

Solution: ??1 = even , ??2 = even

Concretize: ??1 = even , ??2 = even

Core Idea: Do the synthesis in the abstract domain and then concretize

Storyboard Programming: An abstract domain for heap-based data structures



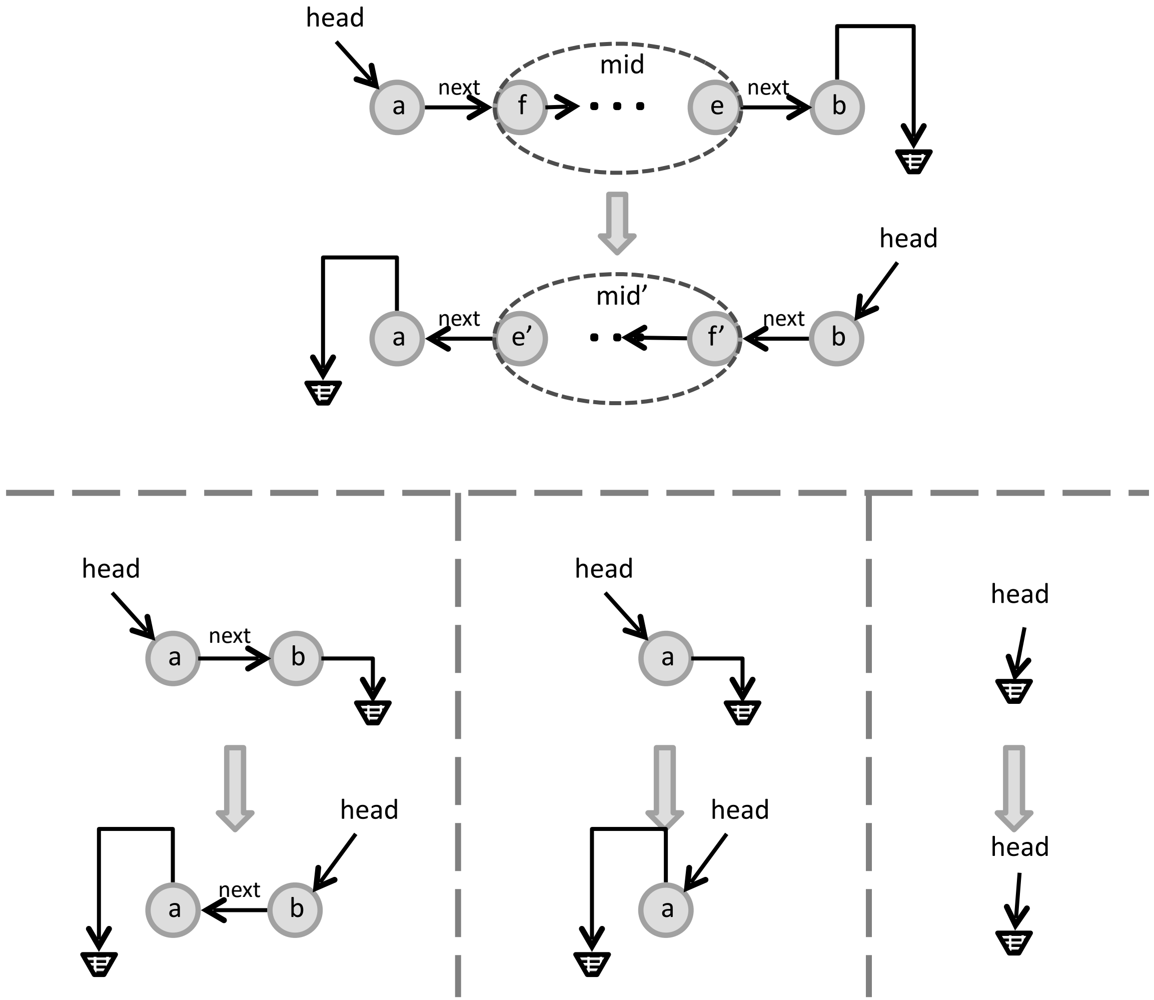
idea : Allow the programmer to describe the behavior of a data-structure manipulation by using abstract shapes as inp
output examples.

Storyboard

Three Inputs:

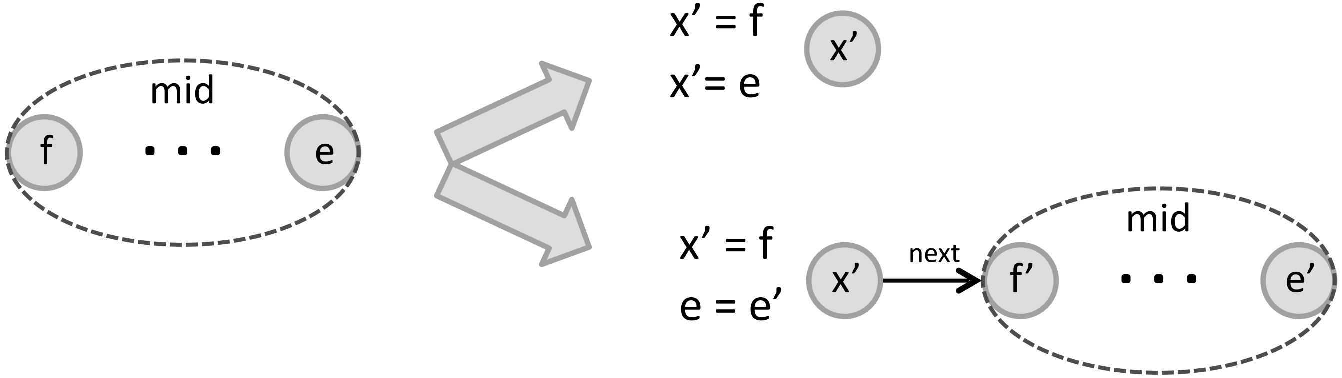
- a set of scenarios,
  - each of which corresponds to an abstract input-output pair;
- a set of fold and unfold definitions,
- a skeleton of the looping structure of the desired algorithm

Scenarios for LL-reversal



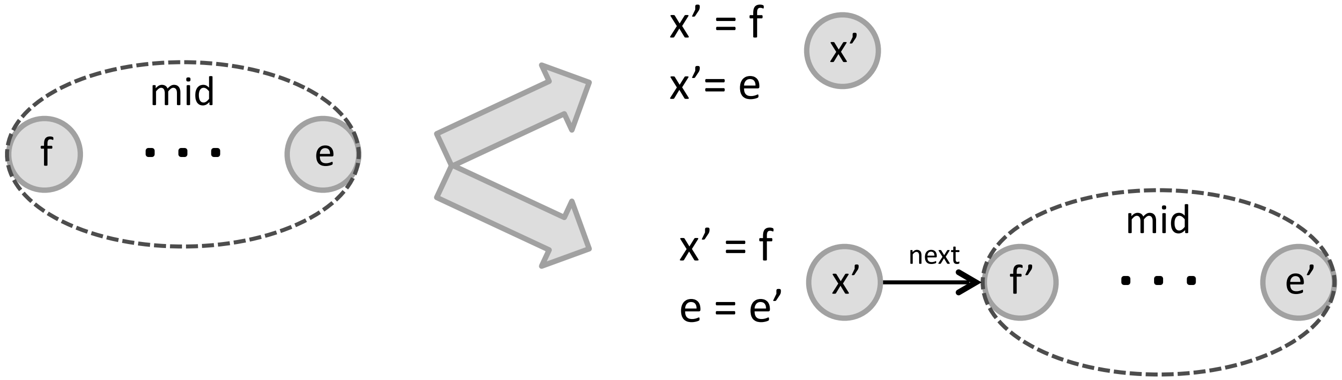
Inductive insights about the datastructure with fold/unfold

Unfold:

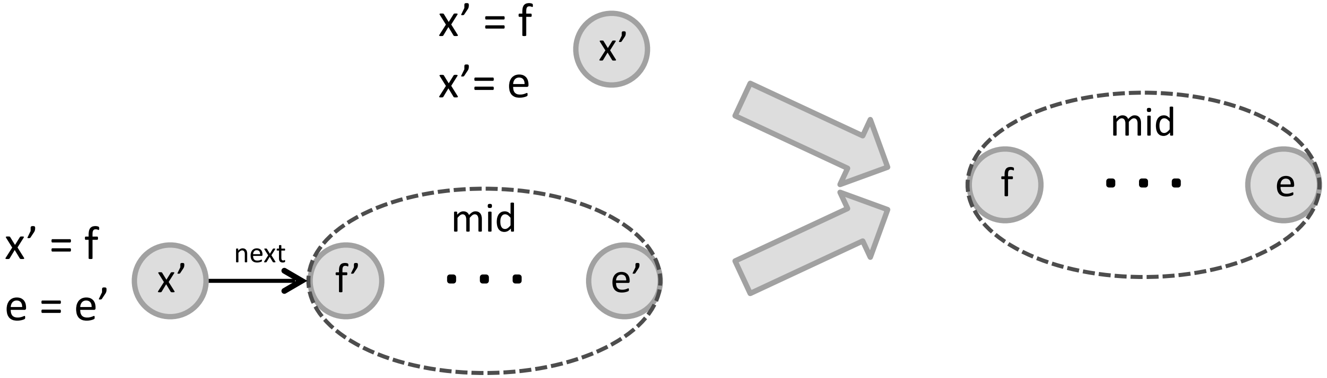


Inductive insights with fold/unfold

Unfold:



Fold:

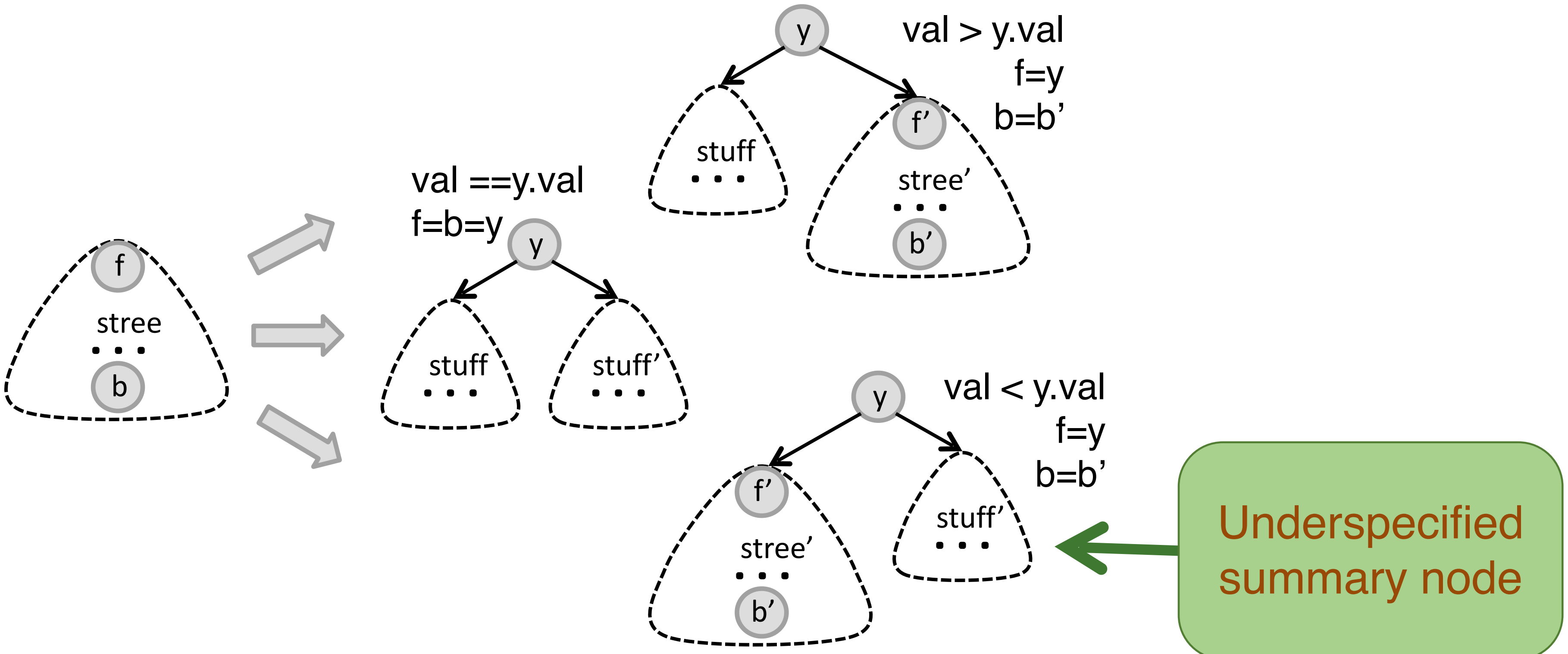


Fold/Unfold

These rules are part of the specification

- without them the scenarios are too imprecise

They can also serve to communicate insights



Next Reading:

Rishabh Singh, Armando Solar-Lezama, *Synthesizing data structure manipulations from storyboards*, 2011