# Fully Automated Shape Analysis Based on Forest Automata

Lukáš Holík **Ondřej Lengál** Adam Rogalewicz
Jiří Šimáček Tomáš Vojnar

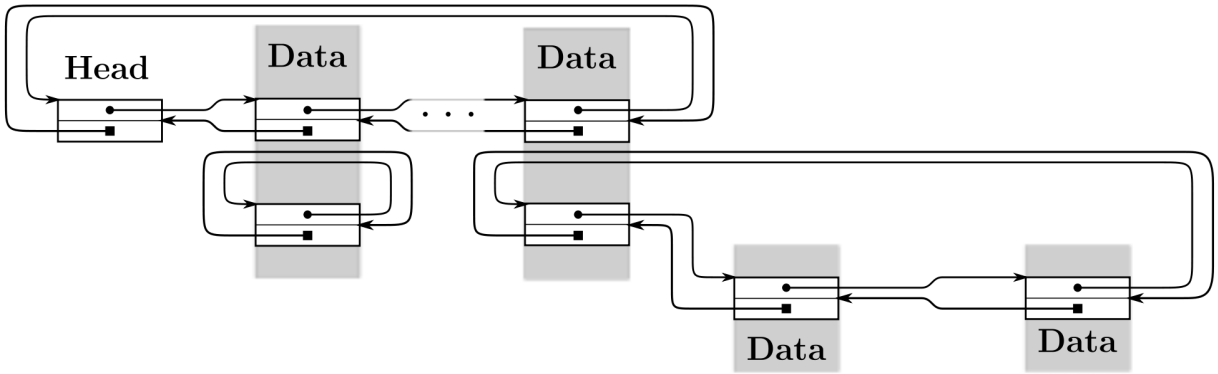
Brno University of Technology, Czech Republic

@CAV'13, St. Petersburg

July 17, 2013

# Shape Analysis

- Precise shape analysis:
  - a notoriously difficult problem
    - dealing with $\infty$ sets of complex graphs
  - many different solutions: logic, automata, ...



# The Forest Automata-based Approach

- Introduced at CAV'11.

# The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ local reasoning of separation logic

# The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ local reasoning of separation logic

  with
  - ☺ flexibility and generality of tree automata (TA)

# The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ local reasoning of separation logic

  with
  - ☺ flexibility and generality of tree automata (TA)

  by
    - splitting the heap into tree components

# The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ local reasoning of separation logic

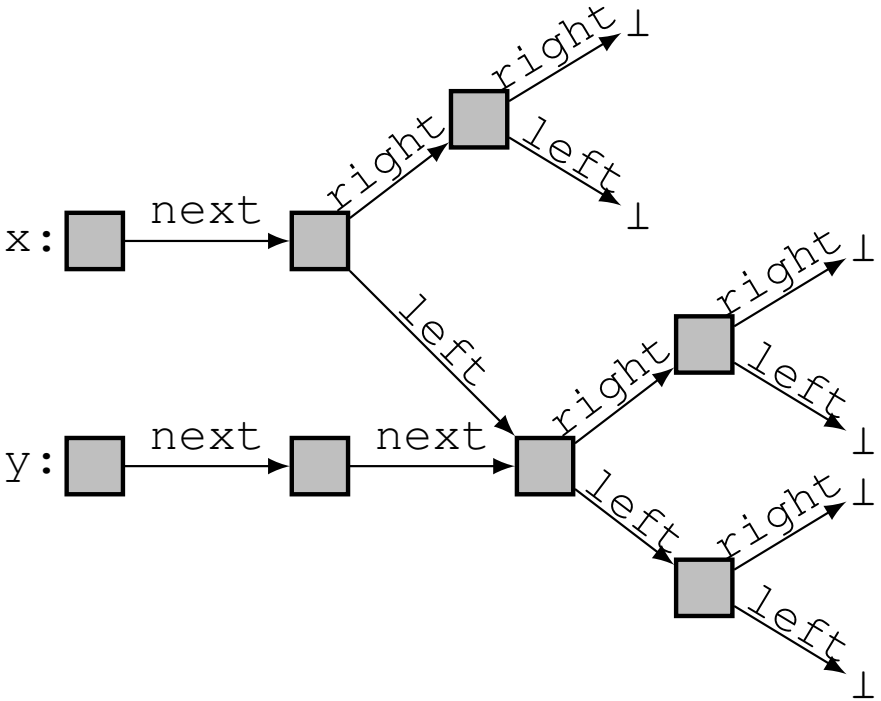
  with
  - ☺ flexibility and generality of tree automata (TA)

  by
  - splitting the heap into tree components

  and
  - TA-based representation of sets of heaps

# Heap Representation

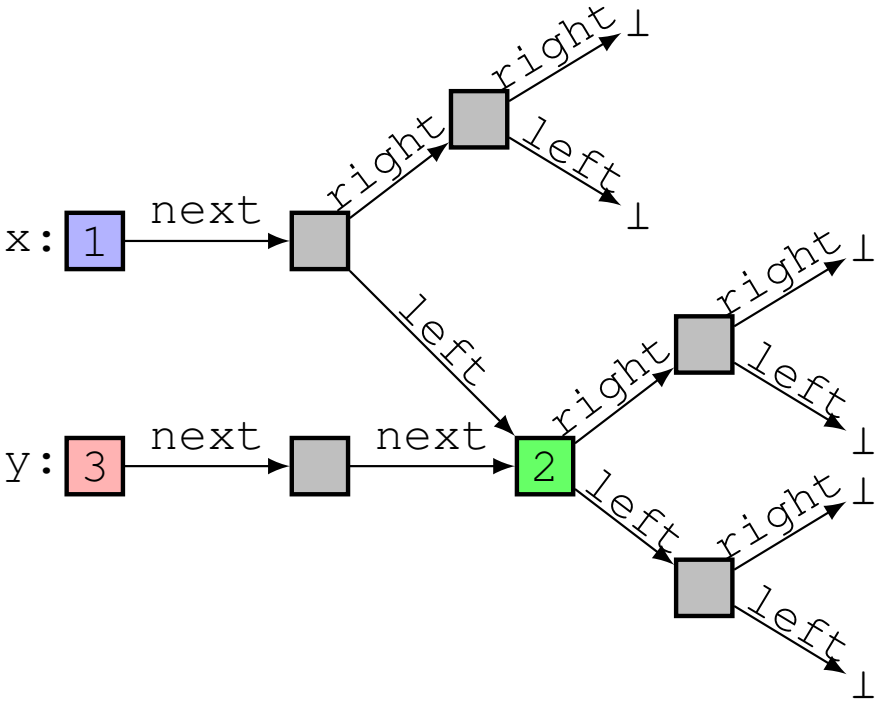
- Forest decomposition of a heap



# Heap Representation

- Forest decomposition of a heap
  - Identify cut-points

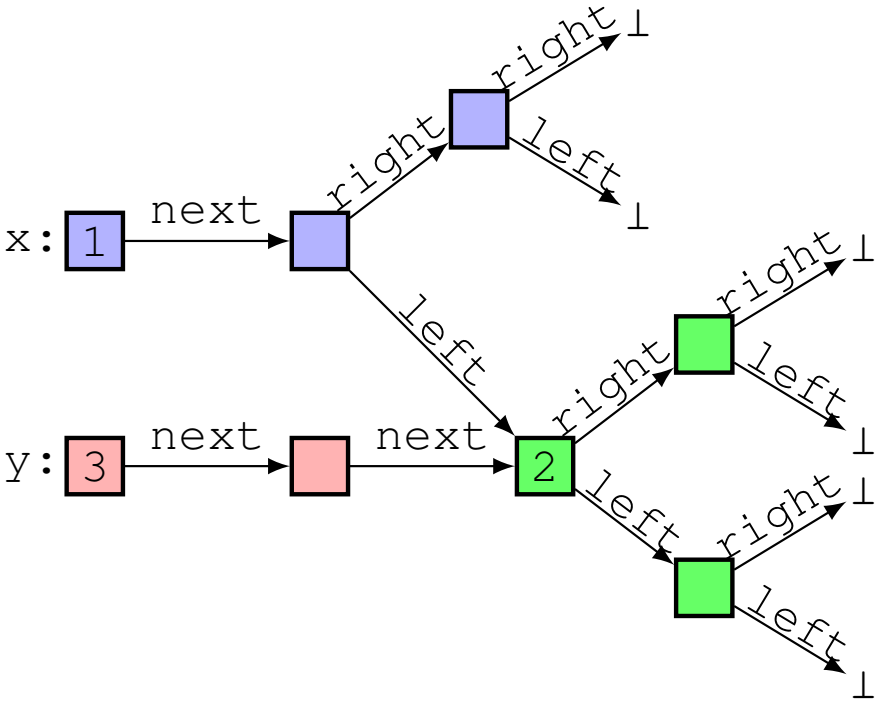
nodes referenced:
- by variables, or
- multiple times



# Heap Representation

- Forest decomposition of a heap
  - Identify cut-points
  - Split the heap into tree components

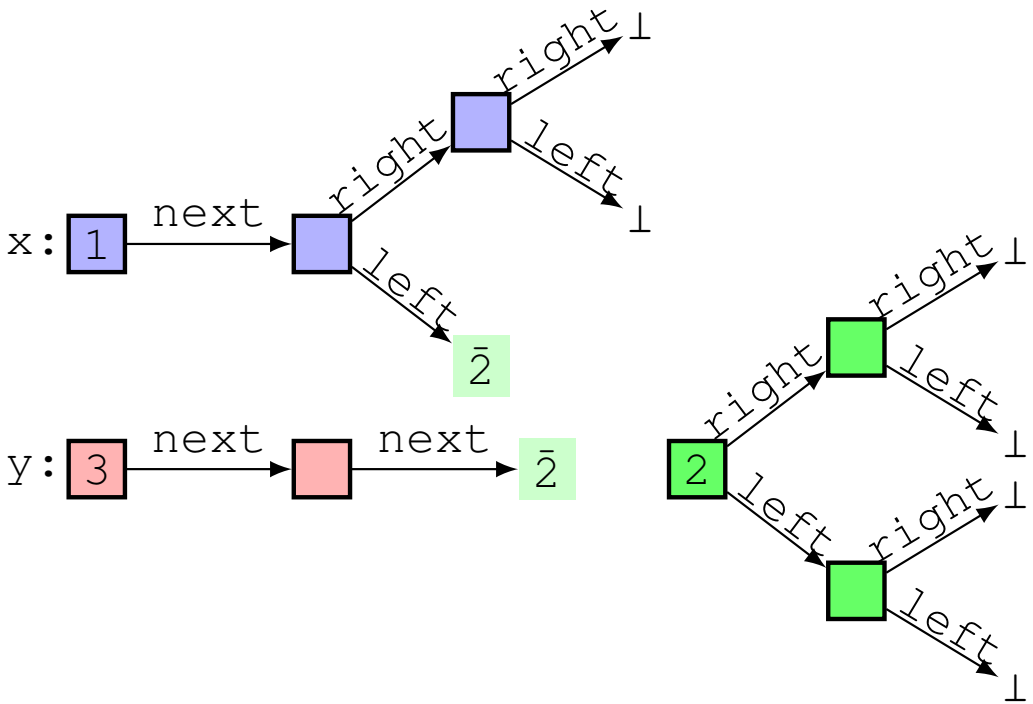
nodes referenced:
- by variables, or
- multiple times



# Heap Representation

- Forest decomposition of a heap
  - Identify cut-points
  - Split the heap into tree components
    - references are explicit

nodes referenced:
- by variables, or
- multiple times

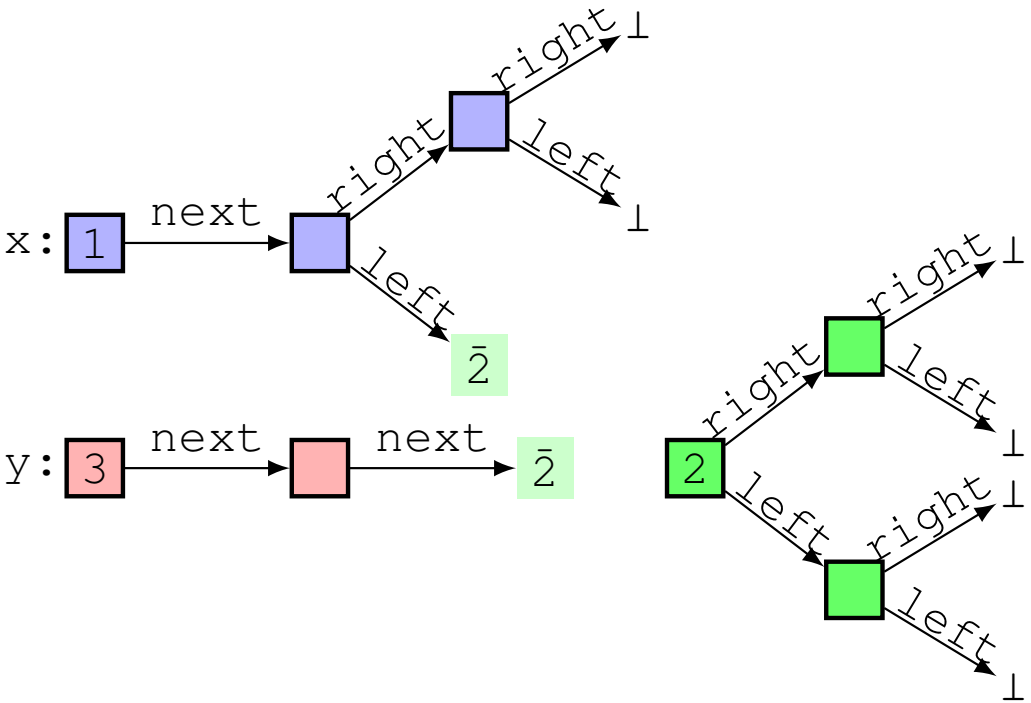


# Heap Representation

- Forest decomposition of a heap
  - Identify cut-points
  - Split the heap into tree components
    - references are explicit
  - Sets of heaps:
    - tree automata to represent sets of tree components
    - tuple of tree automata $(TA_1, \ldots, TA_n) \rightsquigarrow$ **forest automaton** (FA)

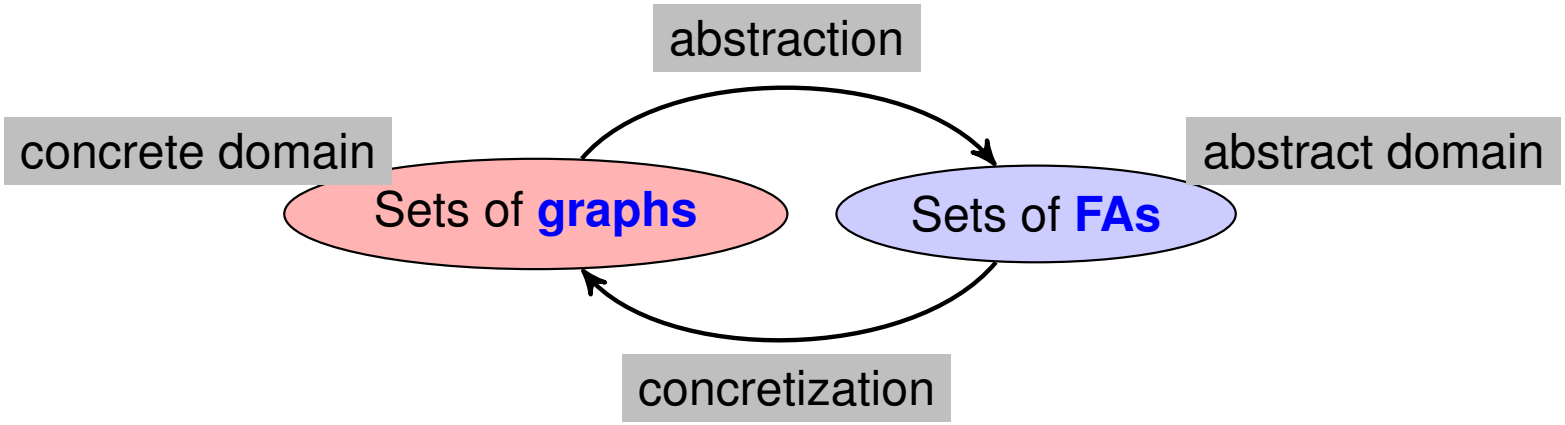
nodes referenced:
- by variables, or
- multiple times

Cartesian semantics



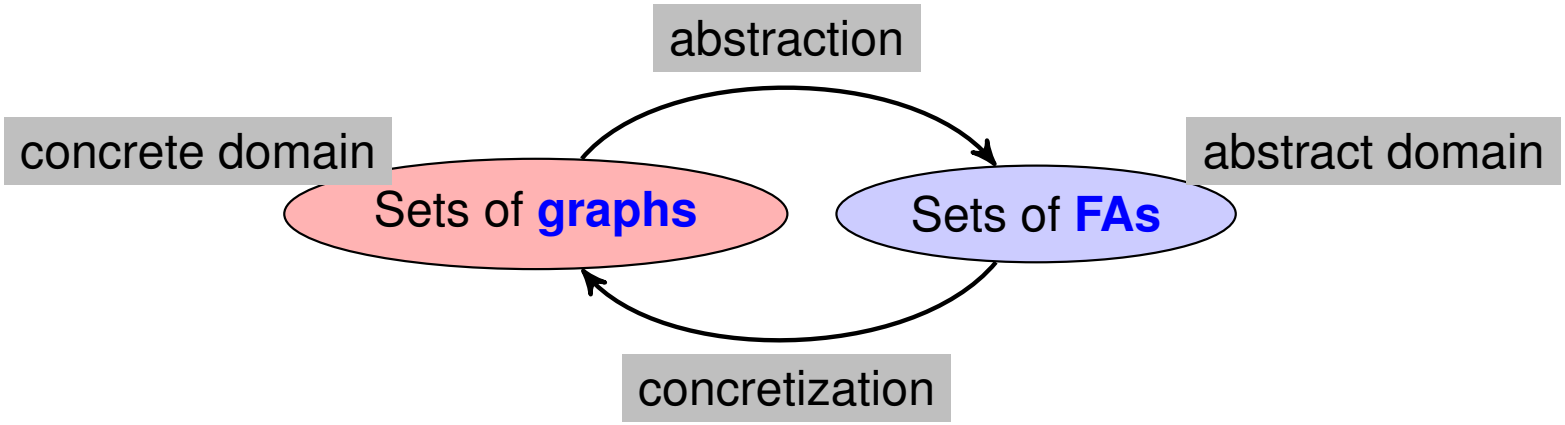
# Analysis

- Abstract Interpretation



# Analysis

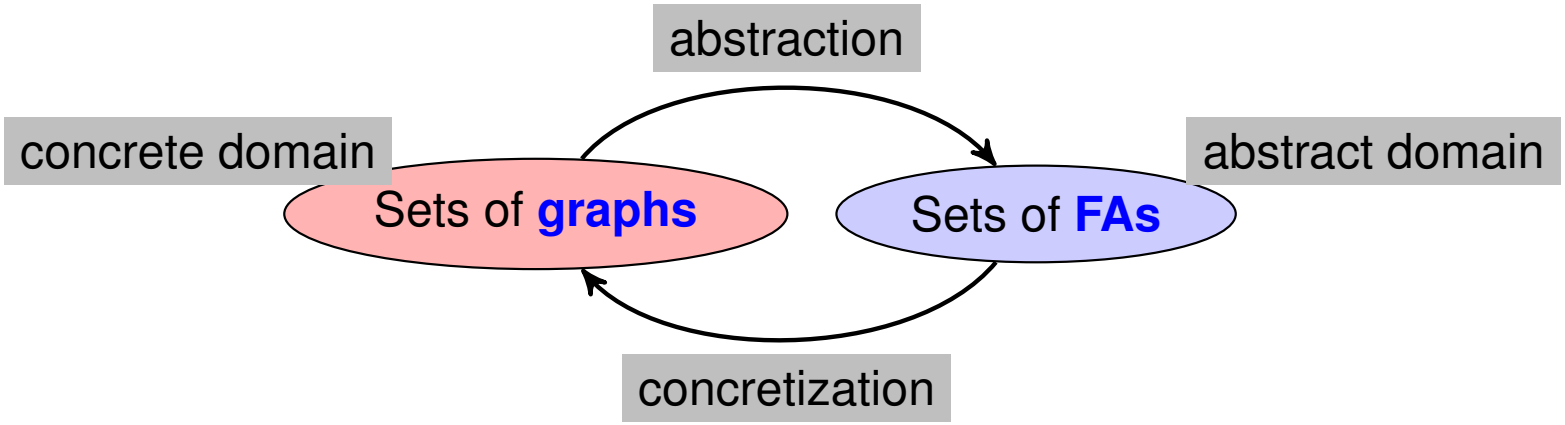
- Abstract Interpretation




- Standard memory manipulating statements

# Analysis

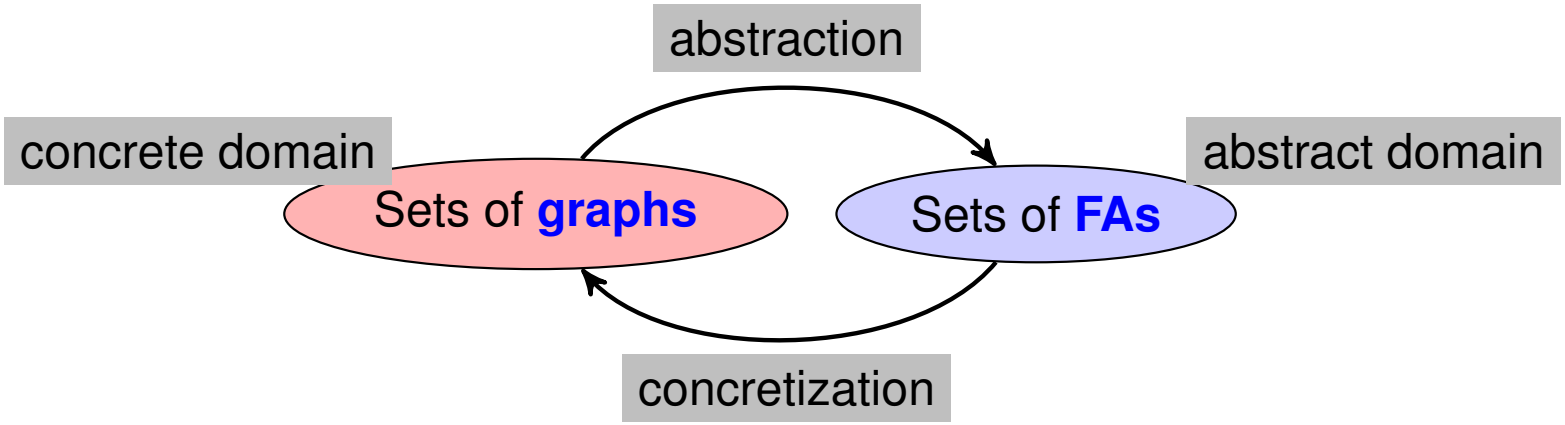
- Abstract Interpretation




- Standard memory manipulating statements
- Abstract transformers execute the statements on FAs

# Analysis

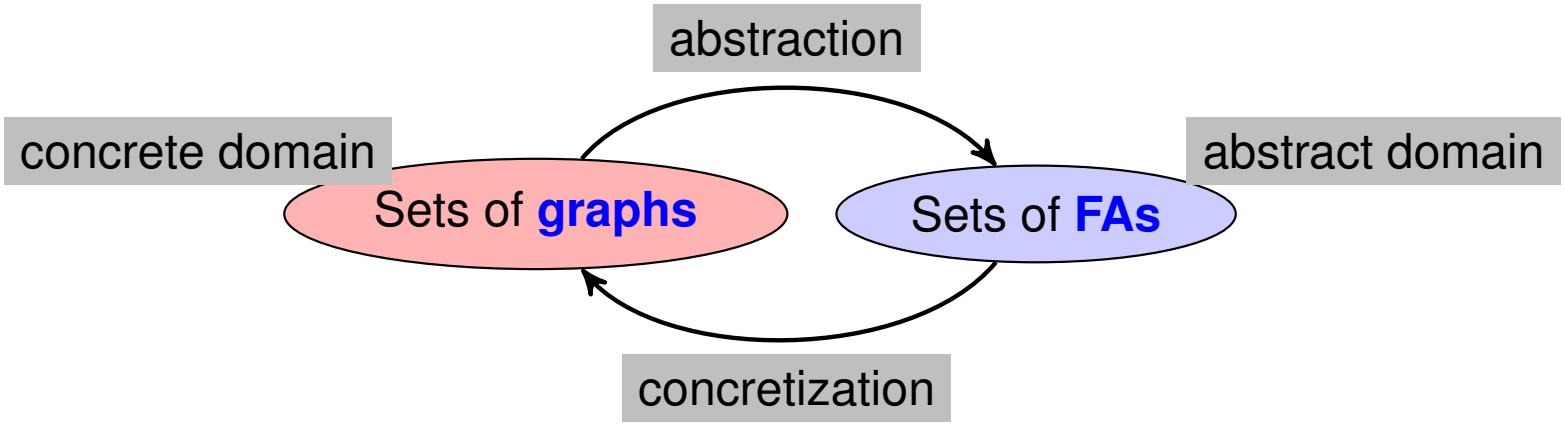
- Abstract Interpretation



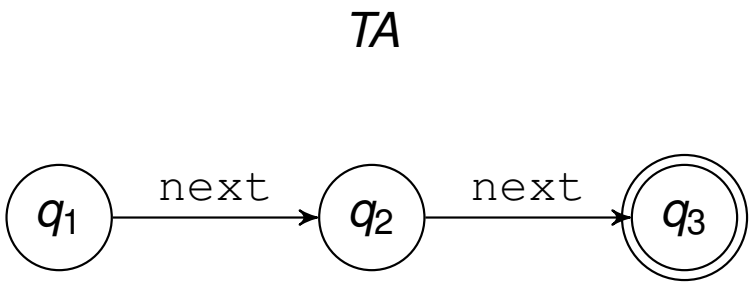

- Standard memory manipulating statements
- Abstract transformers execute the statements on FAs
- Acceleration collapses states of component TAs

# Analysis

- Abstract Interpretation

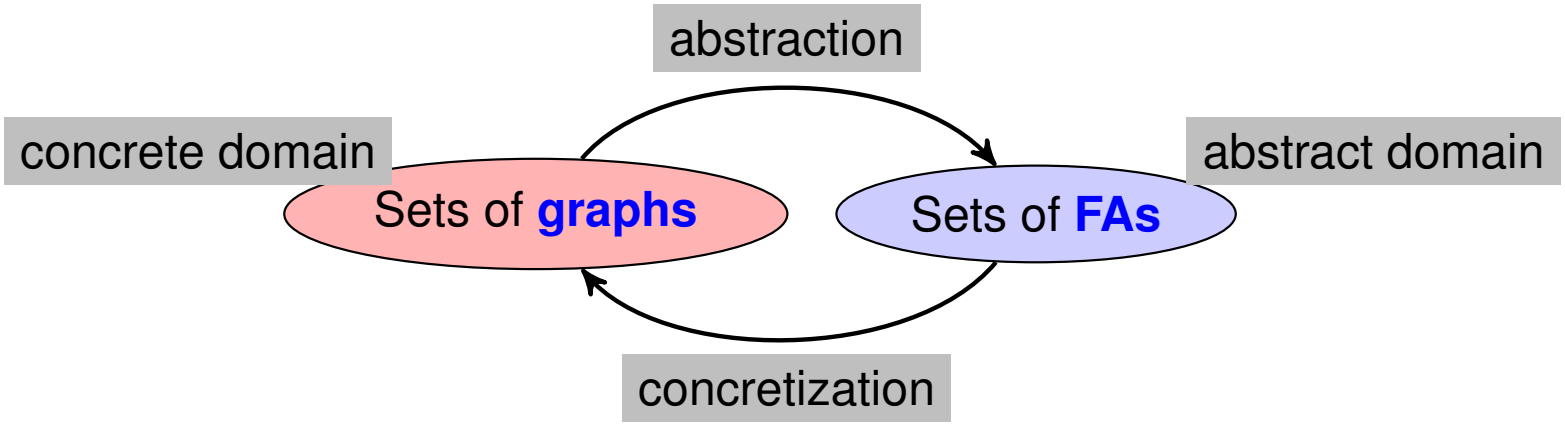



- Standard memory manipulating statements
- Abstract transformers execute the statements on FAs
- Acceleration collapses states of component TAs

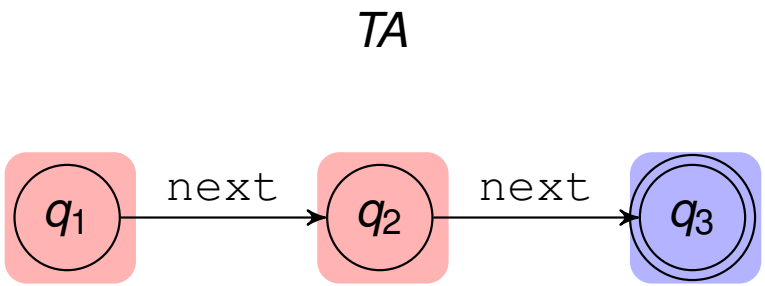


# Analysis

- Abstract Interpretation

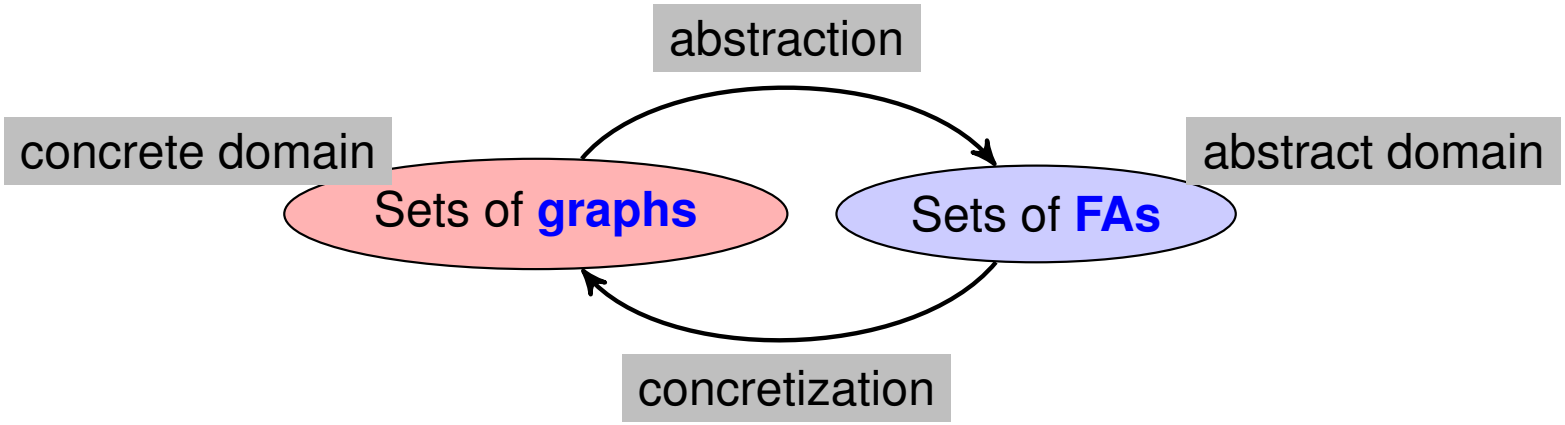



- Standard memory manipulating statements
- Abstract transformers execute the statements on FAs
- Acceleration collapses states of component TAs

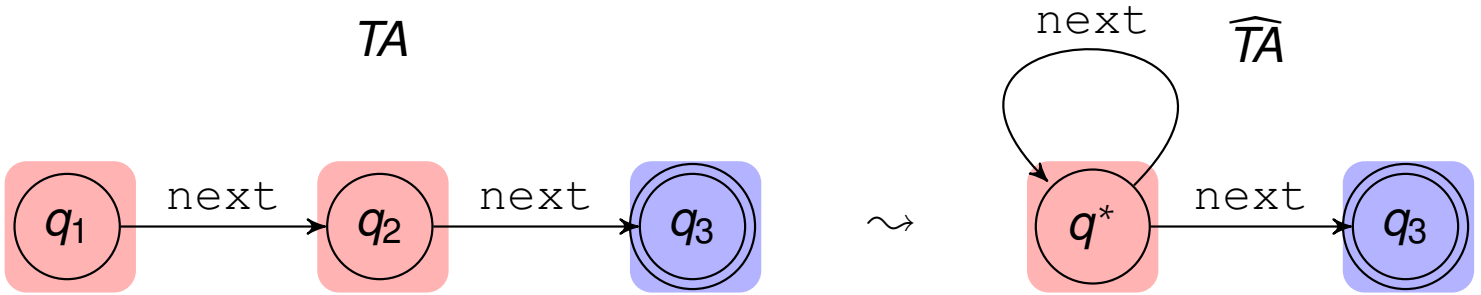


# Analysis

- Abstract Interpretation




- Standard memory manipulating statements
- Abstract transformers execute the statements on FAs
- Acceleration collapses states of component TAs



# Summary

The so-far-presented:

# Summary

The so-far-presented:

- ☺ works well for singly linked lists (SLLs), trees

# Summary

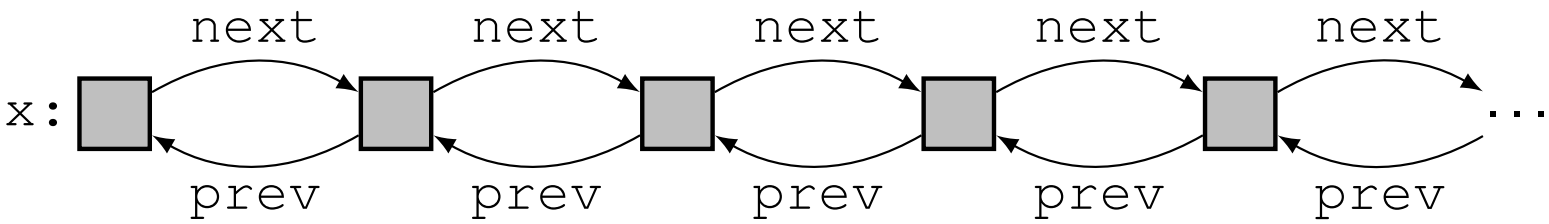
The so-far-presented:

- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\leadsto$ sets of unboundedly many FAs

# Summary

The so-far-presented:

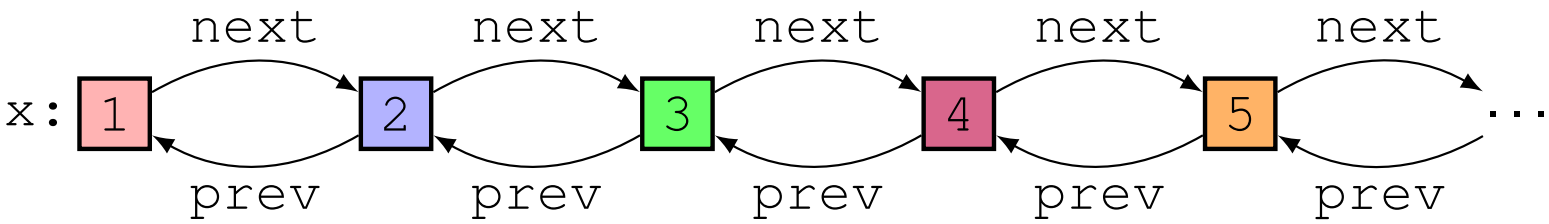
- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\rightsquigarrow$ sets of unboundedly many FAs



# Summary

The so-far-presented:

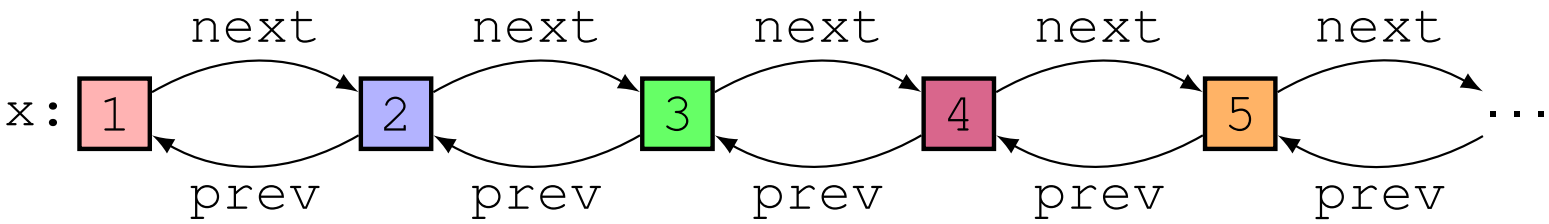
- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\rightsquigarrow$ sets of unboundedly many FAs



# Summary

The so-far-presented:

- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\rightsquigarrow$ sets of unboundedly many FAs




- doubly linked lists (DLLs), lists of circular lists,
- trees with parent pointers,
- skip lists, ...

# Hierarchical Forest Automata

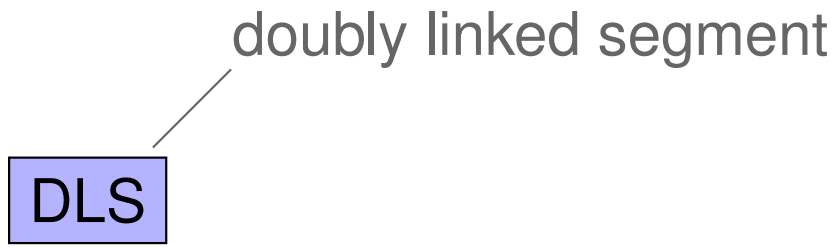
- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs

# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

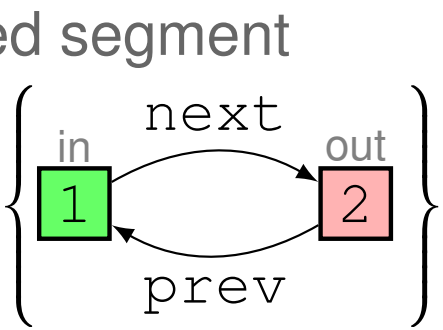



- Example: a box

# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

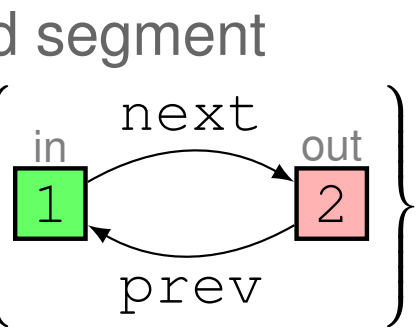
- Example: a box DLS : $\mathcal{L}$(DLS) =

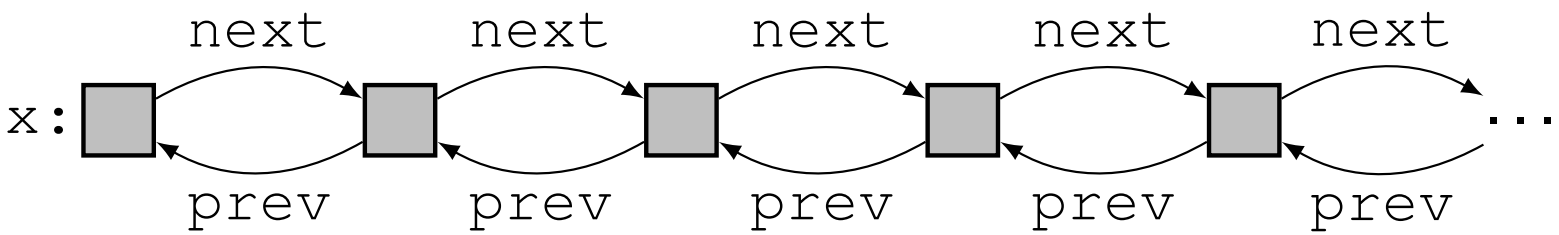


# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

- Example: a box DLS : $\mathcal{L}$(DLS) =

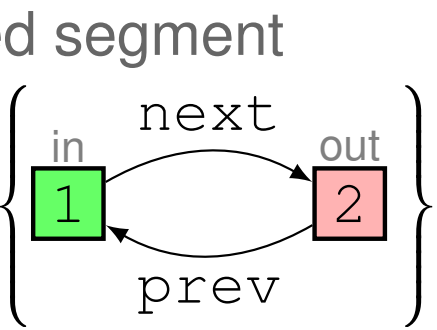


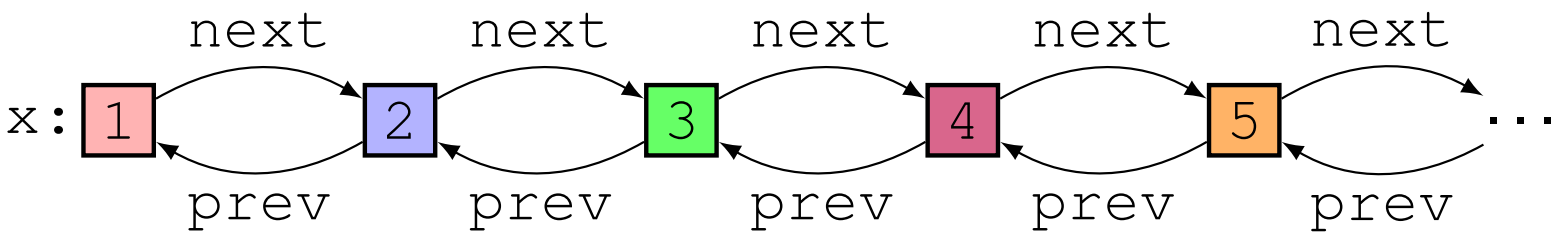



# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

- Example: a box DLS : $\mathcal{L}$(DLS) =

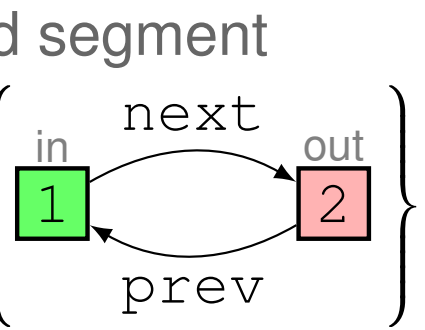


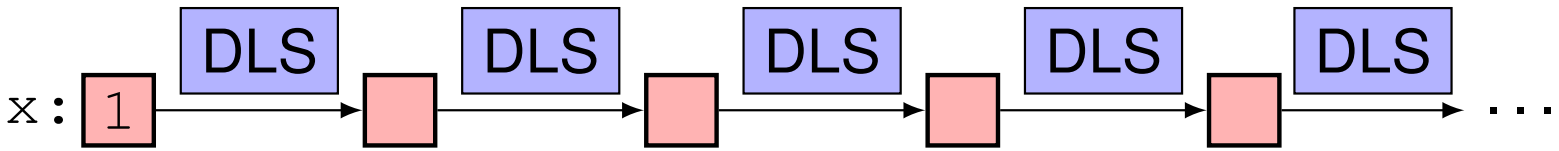



# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

- Example: a box DLS : $\mathcal{L}$(DLS) =

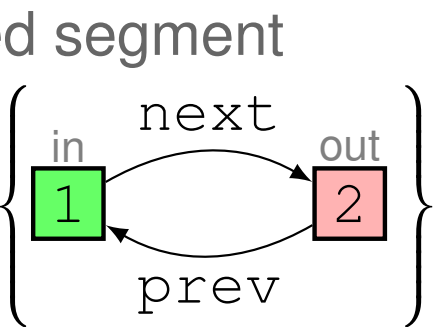


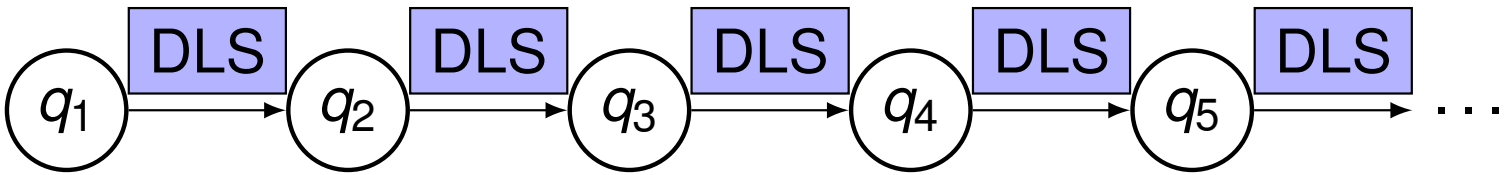



# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

- Example: a box DLS : $\mathcal{L}$(DLS) =

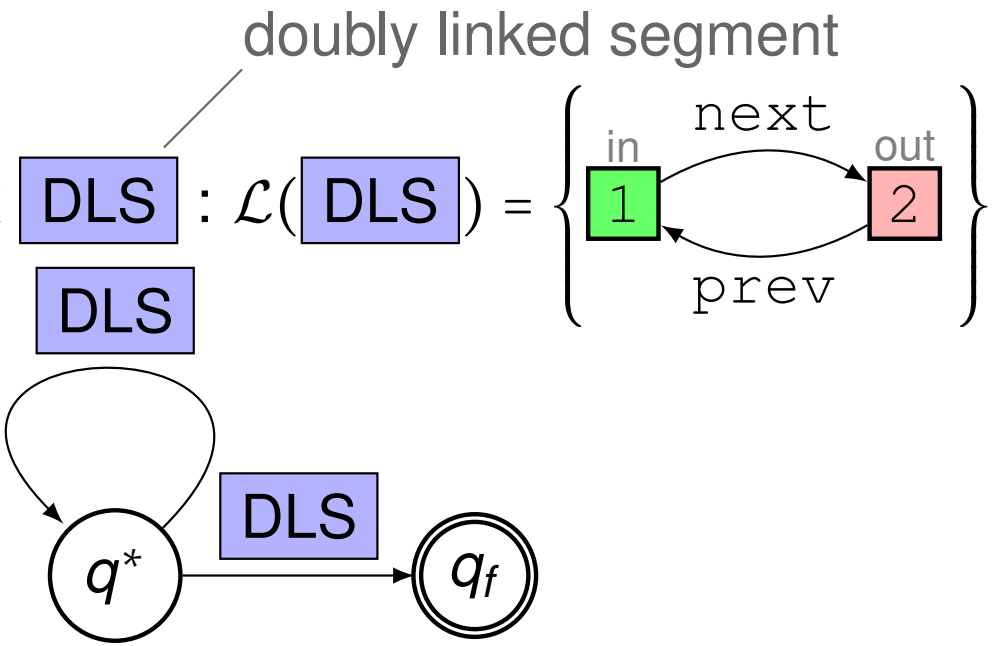


# Hierarchical Forest Automata

- Hierarchical Forest Automata
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - Intuition: replace repeated subgraphs with a single symbol

- Example: a box



# Learning of Boxes

## The Challenge

Where to get the boxes?

# Learning of Boxes

**The Challenge**

Where to get the boxes?

CAV'11 — database of boxes

here — automatic learning

# Learning of Boxes

## Task of the learning algorithm

Identify suitable subgraphs of the heap to be folded into boxes.

# Learning of Boxes

**Task of the learning algorithm**

Identify suitable subgraphs of the heap to be folded into boxes.

- suitable subgraph:
  - when replaced with box , in-degree of some cutpoint drops to 1,

# Learning of Boxes

**Task of the learning algorithm**

Identify suitable subgraphs of the heap to be folded into boxes.

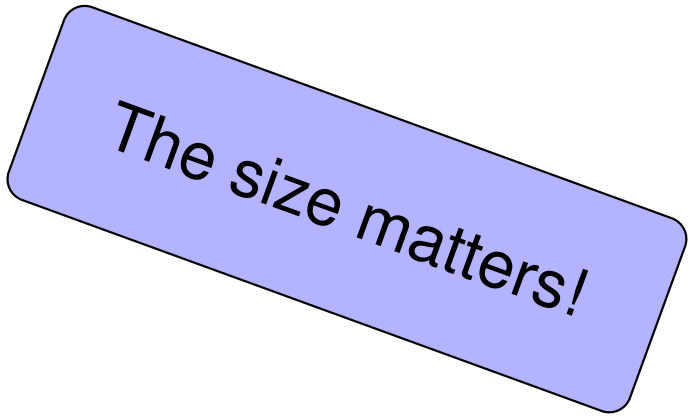
- suitable subgraph:
  - when replaced with box , in-degree of some cutpoint drops to 1,
  - acceleration $\rightsquigarrow$ FA looping over box
    - representing heaps with unboundedly many cut-points

# Learning of Boxes

- suitable subgraph: compromise between

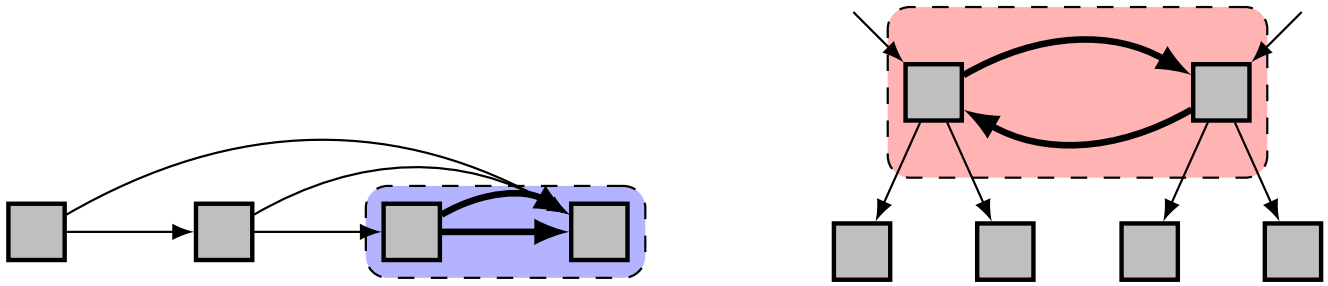
# Learning of Boxes

- suitable subgraph: compromise between
  - small size
    - reusability — acceleration can collapse states
  - large size
    - effectively hide cutpoints — subgraphs with small interfaces



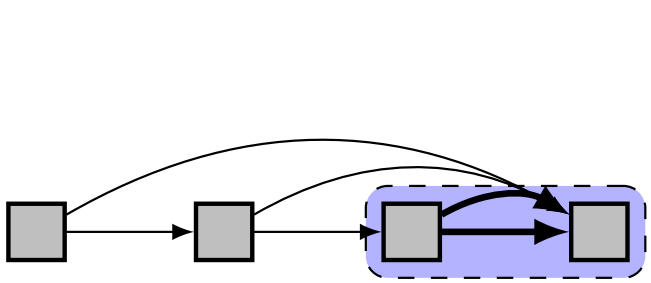
# Learning of Boxes

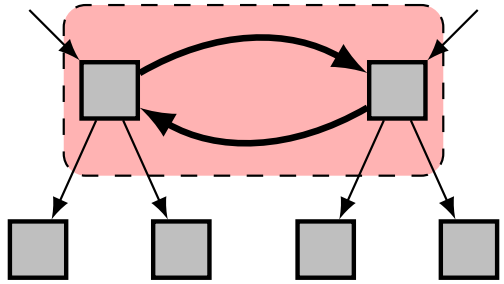
1. building stones — smallest subgraphs meaningful to be folded:



# Learning of Boxes

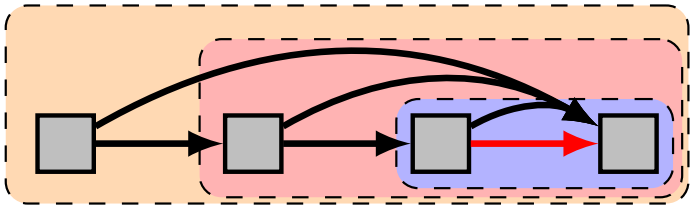
1. building stones — smallest subgraphs meaningful to be folded:





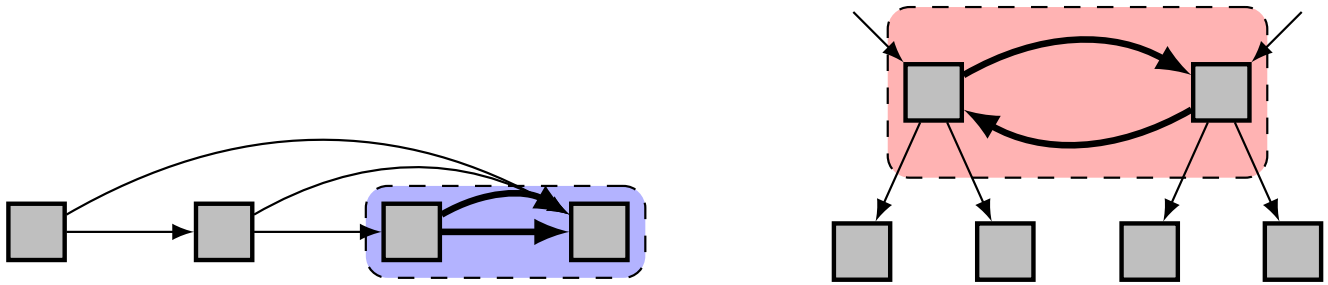
2. handle interface
   - compose intersecting subgraphs

prevent $\infty$ nesting



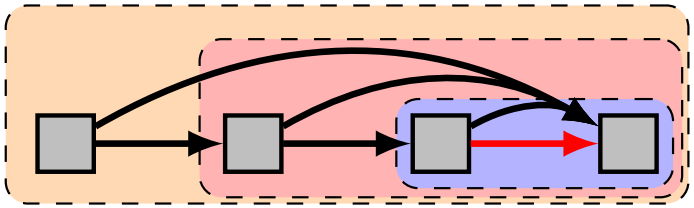
# Learning of Boxes

1. building stones — smallest subgraphs meaningful to be folded:



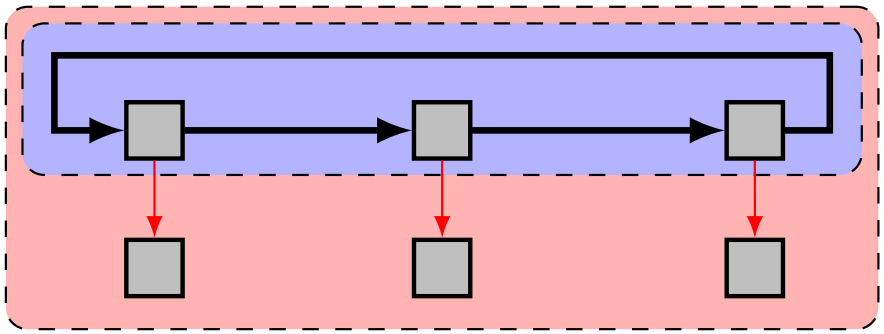
2. handle interface
   - compose intersecting subgraphs

prevent ∞ nesting



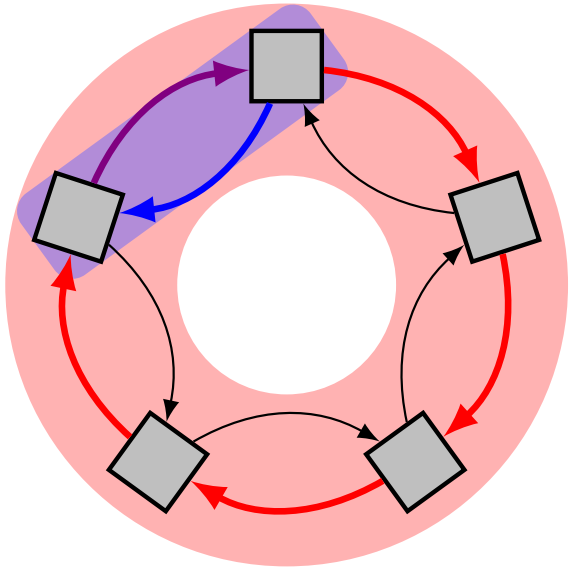
   - enclose paths from inner nodes to leaves

prevent ∞ interface nodes



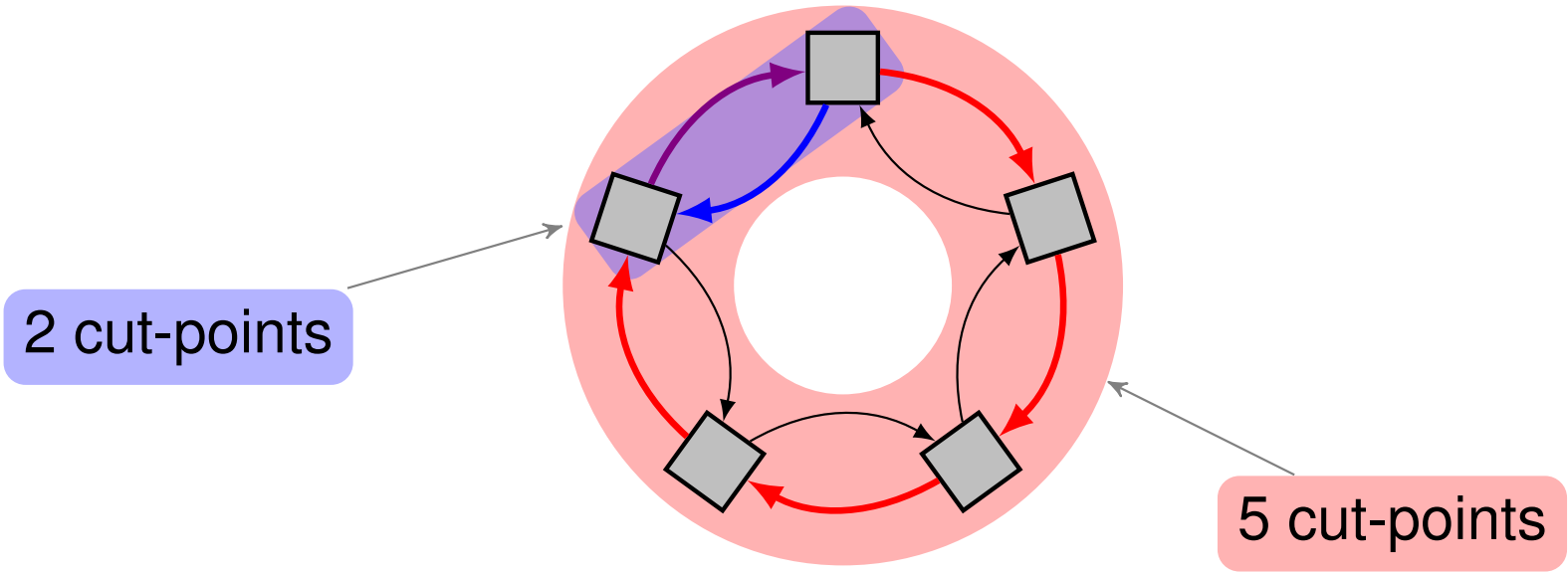
# Learning of Boxes

3 where to start?



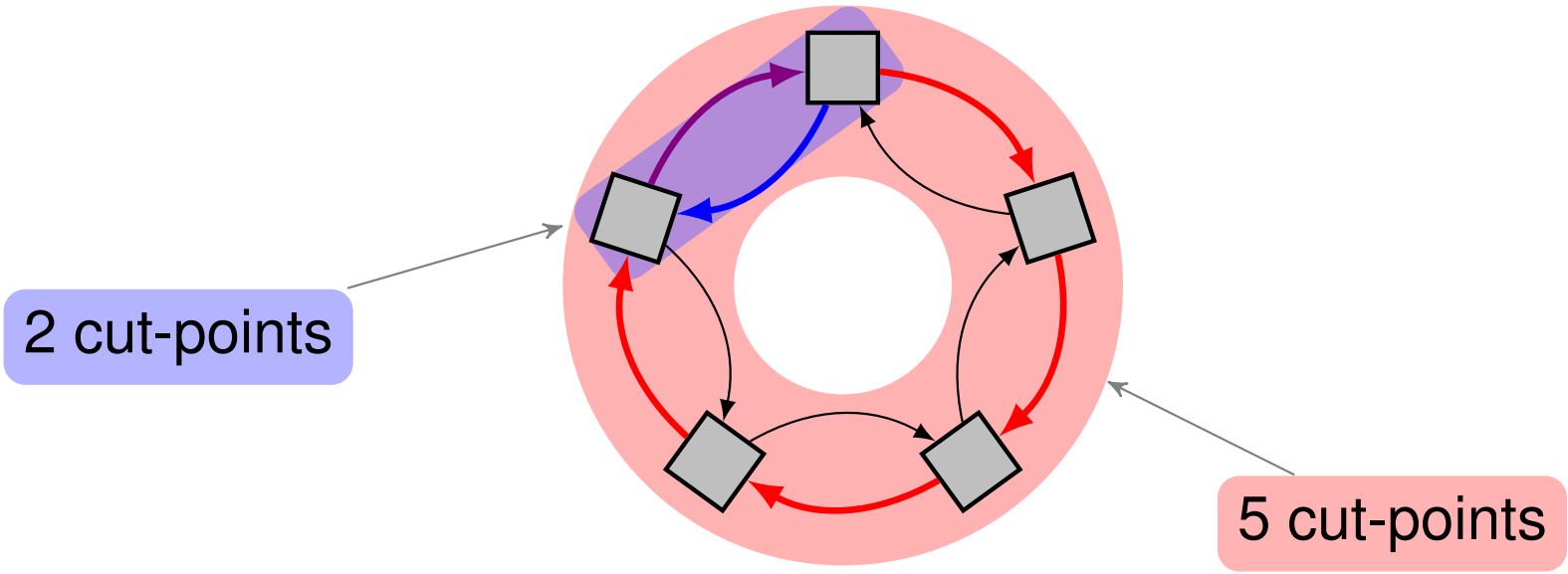
# Learning of Boxes

3 where to start?



# Learning of Boxes

3 where to start?




- prefer subgraphs with less cut-points

# Learning inside Acceleration

- learning and folding of boxes in the acceleration loop

# Learning inside Acceleration

- learning and folding of boxes in the acceleration loop

**The Goal**

Fold boxes that will, after acceleration, appear on cycles of automata.

⇒ hide unboundedly many cut-points

# Learning inside Acceleration

- learning and folding of boxes in the acceleration loop

**The Goal**

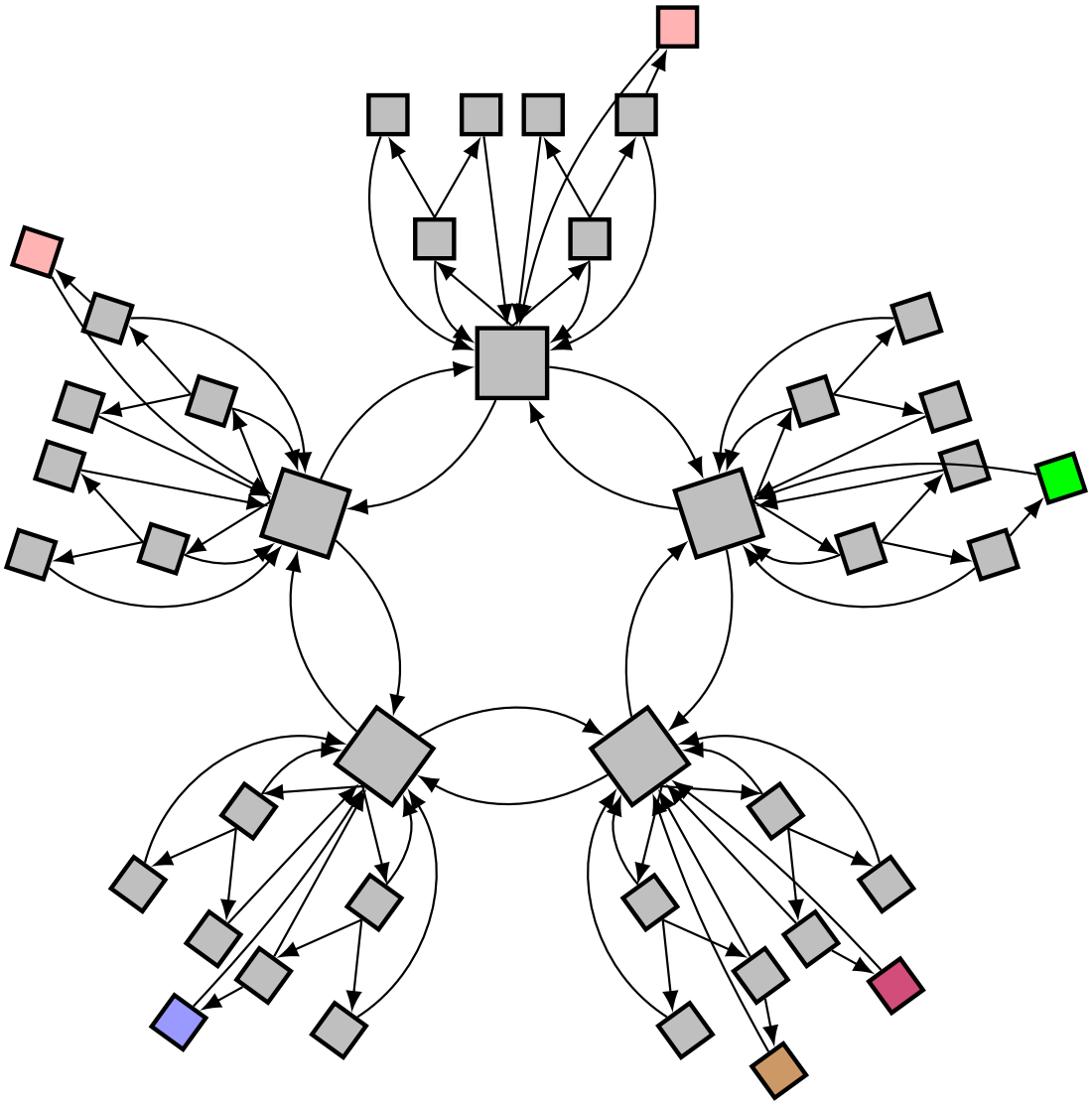
Fold boxes that will, after acceleration, appear on cycles of automata.

⇒ hide unboundedly many cut-points

1 **Algorithm:** Acceleration Loop
2 *Unfold solo boxes*
3 **repeat**
4 *Accelerate*
5 *Fold*
6 **until** *fixpoint*

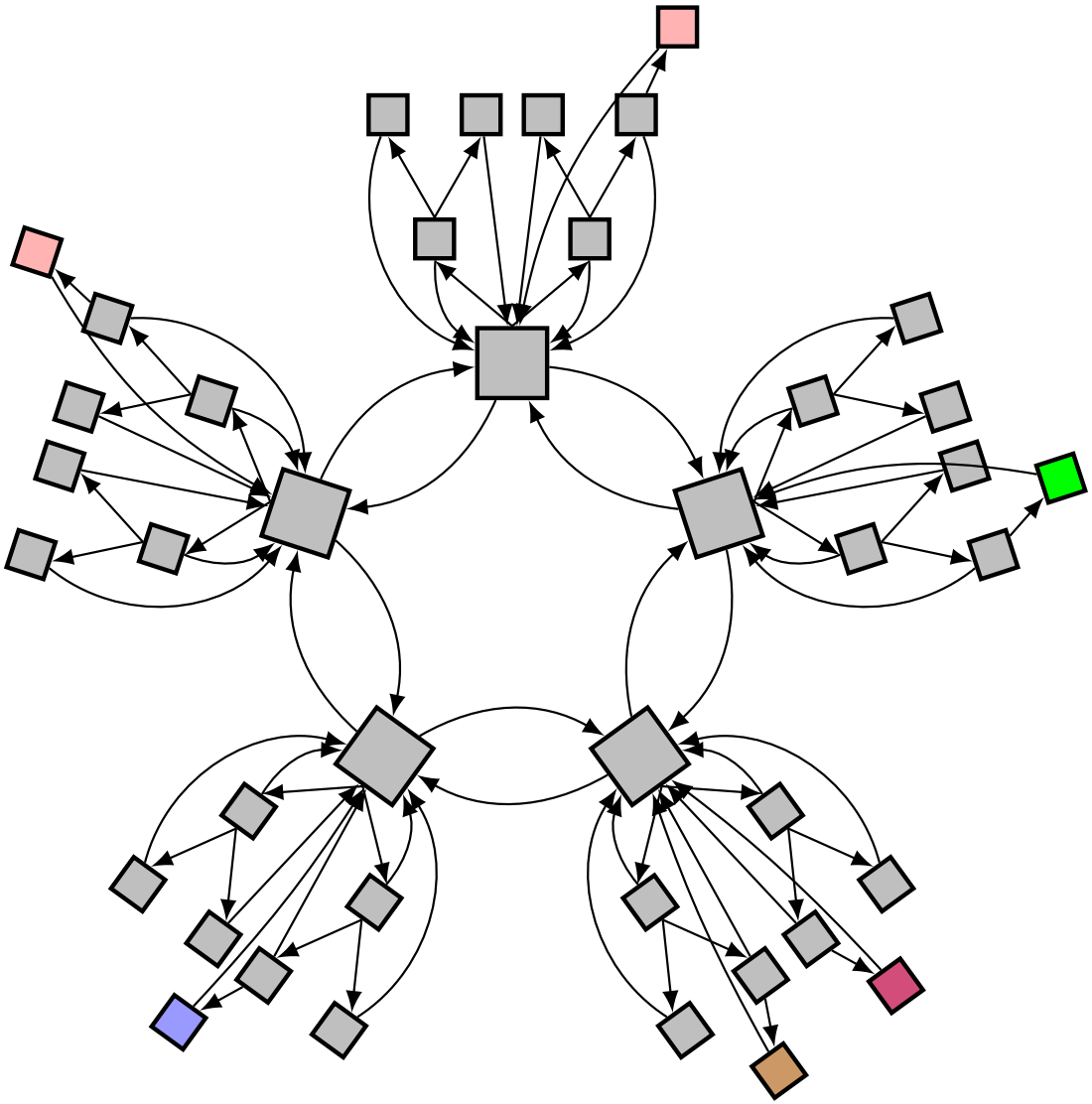
not on a cycle

# Learning of Boxes: Example



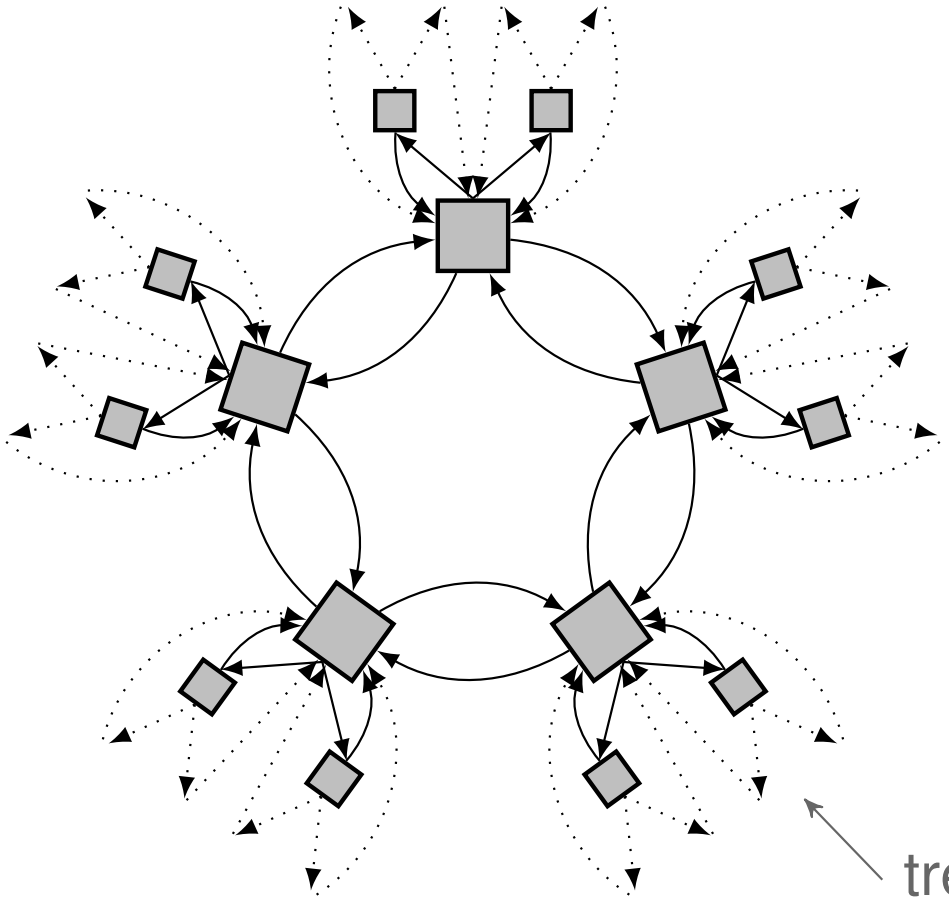
1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

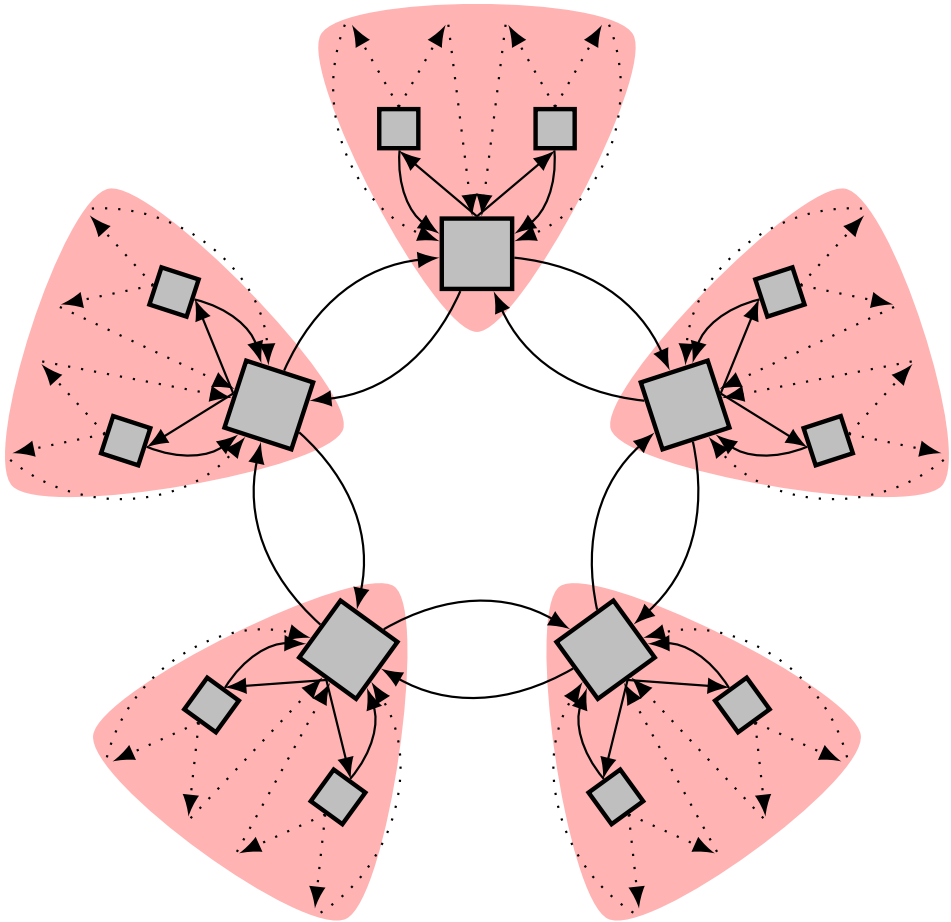
# Learning of Boxes: Example



tree with root ptrs of any height

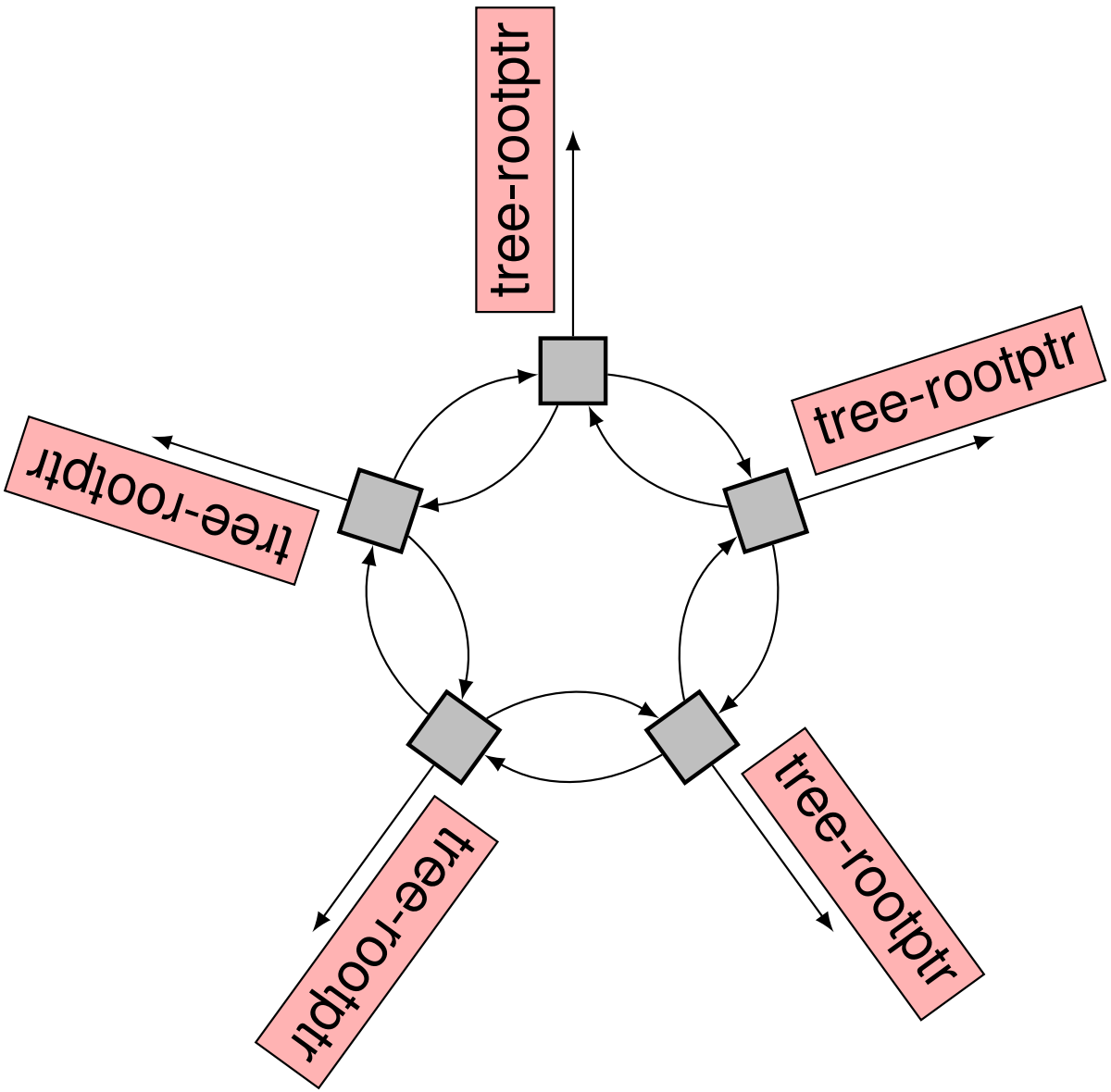
1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



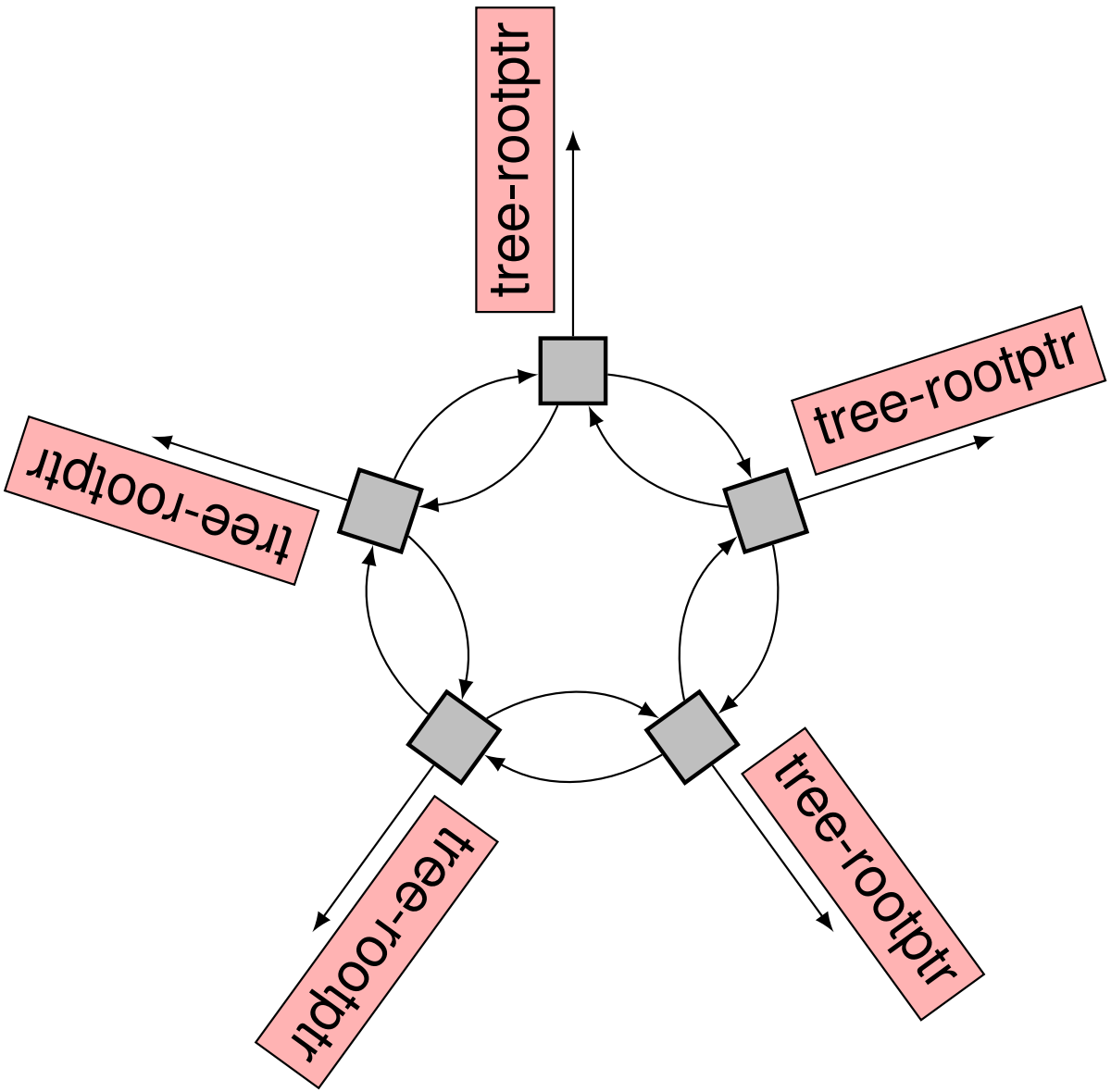
1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



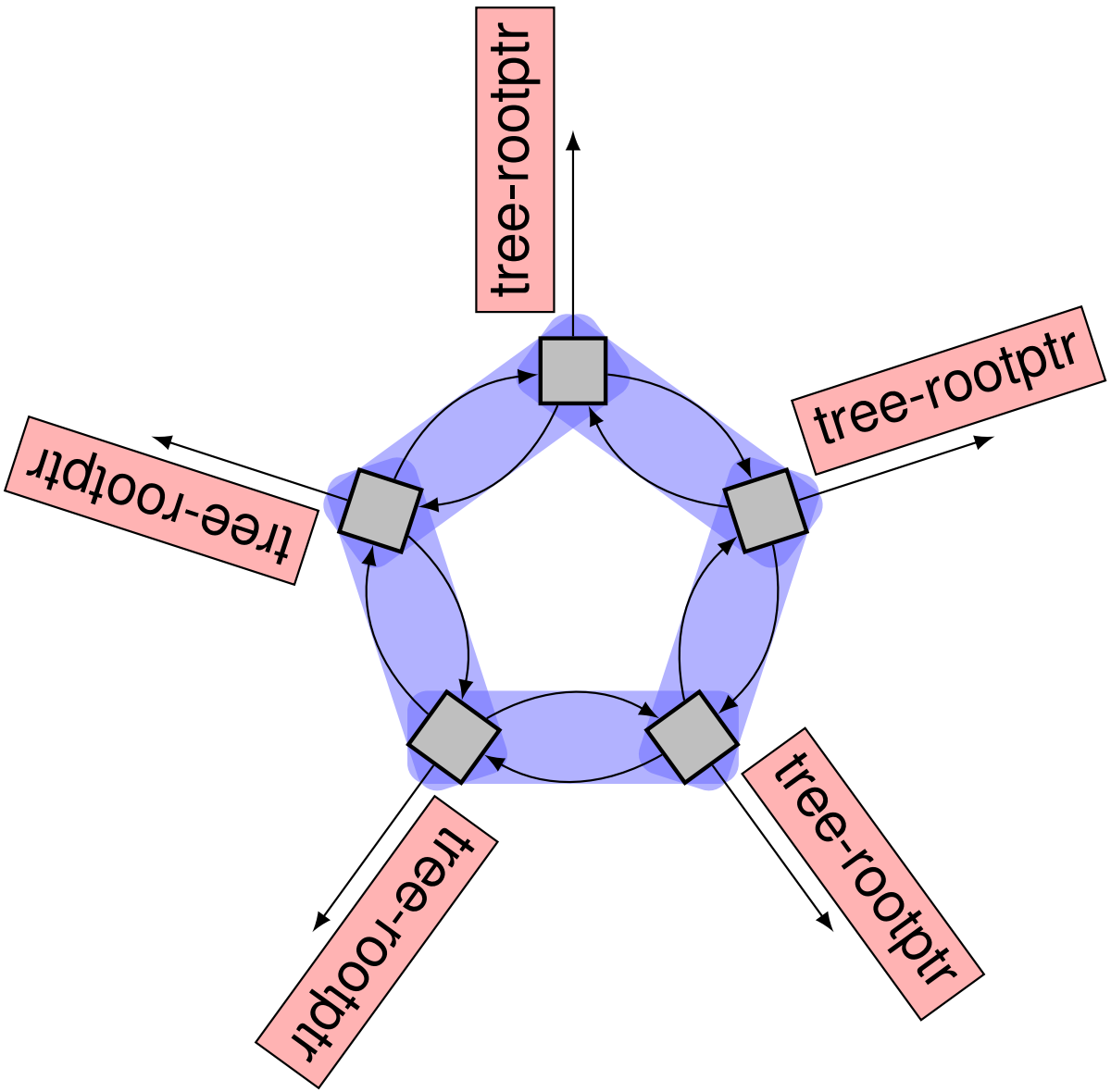

1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



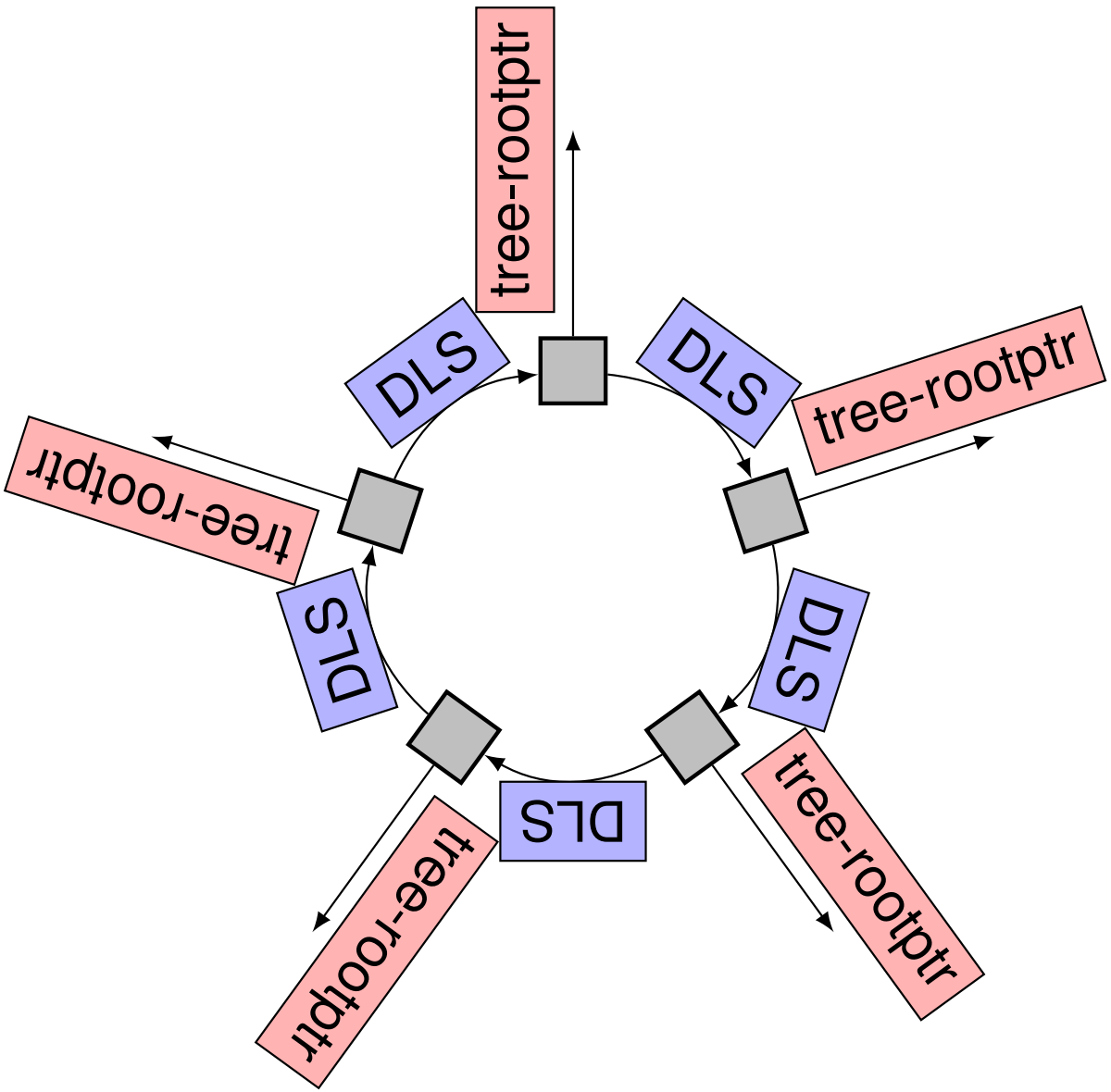

1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



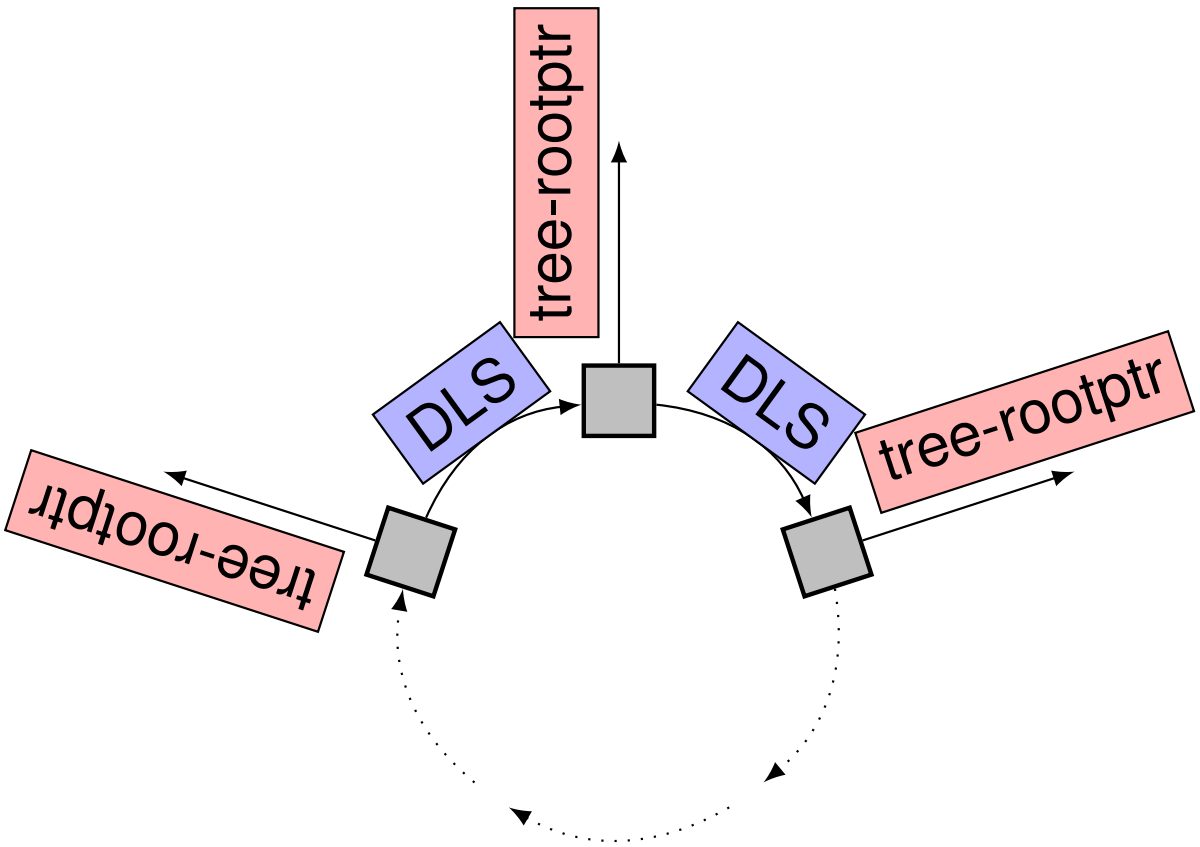

1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example



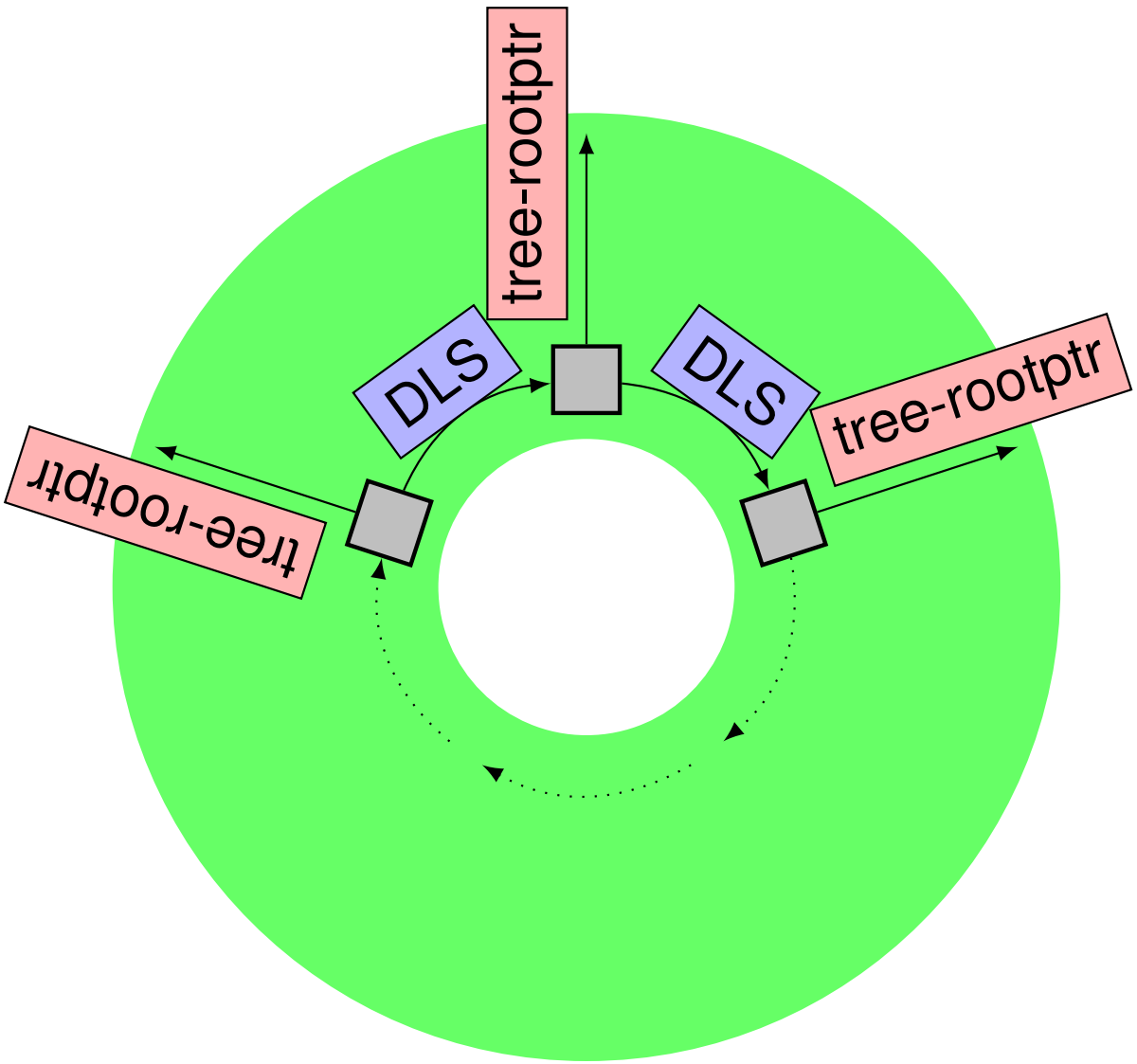

1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example




1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example




1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Learning of Boxes: Example

circular-DLL-of
-trees-rootptr

1 *Unfold solo boxes*
2 **repeat**
3 *Accelerate*
4 *Fold*
5 **until** *fixpoint*

# Experimental Results

- implemented in **Forester** tool: verifies memory safety

# Experimental Results

- implemented in **Forester** tool: verifies memory safety
- comparison with Predator
  - winner of HeapManipulation and MemorySafety of SV-COMP'13

# Experimental Results

- implemented in **Forester** tool: verifies memory safety
- comparison with Predator
  - winner of HeapManipulation and MemorySafety of SV-COMP'13

Table : Results of the experiments [s]

| Example | **FA** | Predator | Example | **FA** | Predator |
|---|---|---|---|---|---|
| SLL (delete) | 0.04 | 0.04 | DLL (reverse) | 0.06 | 0.03 |
| SLL (bubblesort) | 0.04 | 0.03 | DLL (insert) | 0.07 | 0.05 |
| SLL (mergesort) | 0.15 | 0.10 | DLL (insertsort$_1$) | 0.40 | 0.11 |
| SLL (insertsort) | 0.05 | 0.04 | DLL (insertsort$_2$) | 0.12 | 0.05 |
| SLL (reverse) | 0.03 | 0.03 | DLL of CDLLs | 1.25 | 0.22 |
| SLL+head | 0.05 | 0.03 | DLL+subdata | 0.09 | T |
| SLL of 0/1 SLLs | 0.03 | 0.11 | CDLL | 0.03 | 0.03 |
| SLL$_{\text{Linux}}$ | 0.03 | 0.03 | tree | 0.14 | Err |
| SLL of CSLLs | 0.73 | 0.12 | tree+parents | 0.21 | T |
| SLL of 2CDLLs$_{\text{Linux}}$ | 0.17 | 0.25 | tree+stack | 0.08 | Err |
| skip list$_2$ | 0.42 | T | tree (DSW) Deutsch-Schorr-Waite | 0.40 | Err |
| skip list$_3$ | 9.14 | T | tree of CSLLs | 0.42 | Err |

timeout

false positive

# Conclusion

Shape analysis with forest automata:

# Conclusion

Shape analysis with forest automata:

- fully automated

# Conclusion

Shape analysis with forest automata:

- fully automated
- very flexible framework

# Conclusion

Shape analysis with forest automata:

- fully automated
- very flexible framework
- **Forester** tool

# Conclusion

Shape analysis with forest automata:

- fully automated
- very flexible framework
- **Forester** tool
- successfully verified:
  - (singly/doubly linked (circular)) lists (of (. . . ) lists)
  - trees
  - skip lists

# Conclusion

Shape analysis with forest automata:

- fully automated
- very flexible framework
- **Forester** tool
- successfully verified:
  - (singly/doubly linked (circular)) lists (of (. . . ) lists)
  - trees
  - skip lists

- a follow-up work:
  - tracking ordering relations
    - P. Abdulla, L. Holík, B. Jonsson, O. Lengál, C.Q. Tring, and T. Vojnar. **Verification of Heap Manipulating Programs with Ordered Data by Extended Forest Automata.** To appear in *Proc. of ATVA'13*.

# Future work

- CEGAR loop
  - **red**-**black** trees, . . .

# Future work

- CEGAR loop
  - **red**-**black** trees, . . .
- concurrent data structures
  - lockless skip lists, . . .

# Future work

- CEGAR loop
  - **red**-**black** trees, . . .
- concurrent data structures
  - lockless skip lists, . . .
- recursive boxes
  - B+ trees, . . .