Fully Automated Shape Analysis Based on Forest Automata[†]

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

Brno University of Technology, Czech Republic

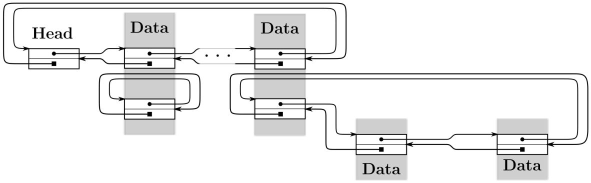
@Rich Model Toolkit COST Action Meeting, Malta 2013

June 17, 2013

[†]To appear in *Proc. of CAV'13.*

Shape Analysis

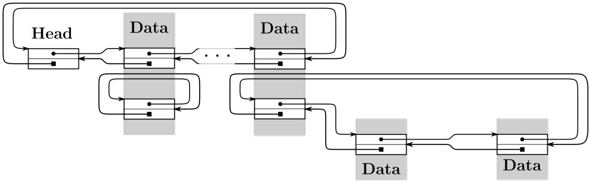
- Precise shape analysis:
  - a notoriously difficult problem



  - specialized solutions (lists)
  - help from the outside (loop invariants, inductive predicates)

Shape Analysis

- Precise shape analysis:
  - a notoriously difficult problem



  - specialized solutions (lists)
  - help from the outside (loop invariants, inductive predicates)

- Classes of errors:
  - error line reachability
  - invalid pointer dereference
  - occurrence of garbage

Inspiration

- Separation Logic
  - ☺ local reasoning, well scalable
  - ☹ fixed abstraction

Inspiration

- Separation Logic
  - ☺ local reasoning, well scalable
  - ☹ fixed abstraction

- Abstract Regular Tree Model Checking (ARTMC)
  - ☺ uses tree automata (TA), flexible and refinable abstraction
  - ☹ monolithic encoding of the heap, not very scalable

The Forest Automata-based Approach

- Introduced at CAV'11.

The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ flexibility of ARTMC

The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ flexibility of ARTMC

  with
  - ☺ local reasoning of SL

The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ flexibility of ARTMC

  with
  - ☺ local reasoning of SL

  by
  - splitting the heap into tree components

The Forest Automata-based Approach

- Introduced at CAV'11.
- Combines
  - ☺ flexibility of ARTMC

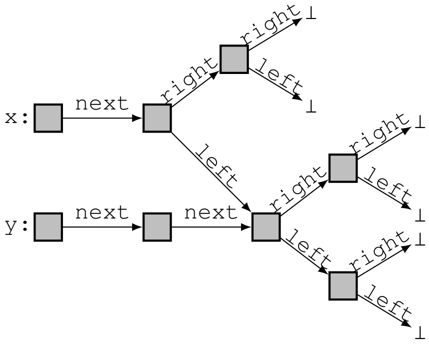
  with
  - ☺ local reasoning of SL

  by
  - ‣ splitting the heap into tree components

  and
  - ‣ representing sets of heaps using TA

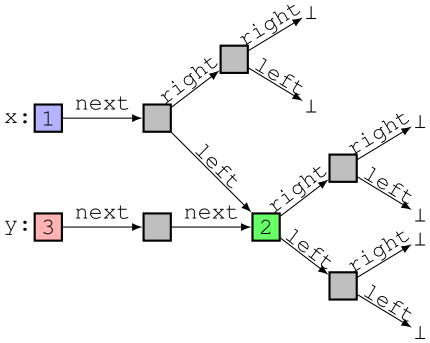
Heap Representation

- Forest decomposition of a heap



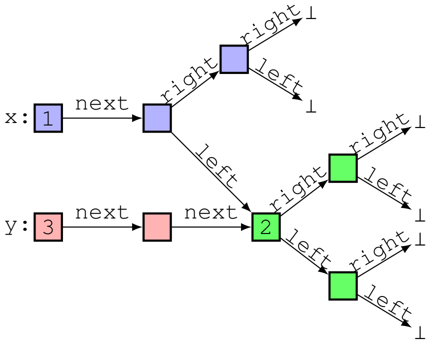
Heap Representation

- Forest decomposition of a heap
  - Identify cut-points ← nodes referenced:
    - by variables, or
    - multiple times



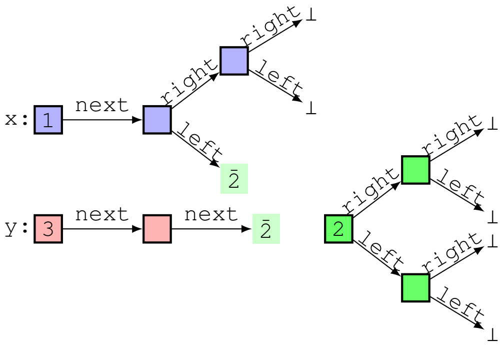
Heap Representation

- Forest decomposition of a heap
  - Identify cut-points ← nodes referenced:
    - by variables, or
    - multiple times
  - Split the heap into tree components



Heap Representation

- Forest decomposition of a heap
  - Identify cut-points ← nodes referenced:
    - by variables, or
    - multiple times
  - Split the heap into tree components
  - References are explicit



Heap Representation

- a heap $h \mapsto$ a forest $(\text{⯭}_1, \text{⯭}_2, \ldots, \text{⯭}_n)$

Heap Representation

- a heap $h \mapsto$ a forest $(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n), (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_m), \ldots\}$

Heap Representation

- a heap $h \mapsto$ a forest $(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n), (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n) \approx (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_n)$$

iff $\forall i : \twoheaduparrow_i$ and $\twoheaduparrow'_i$ contain the same references in the same order

Heap Representation

- a heap $h \mapsto$ a forest $(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n), (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n) \approx (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_n)$$

iff $\forall i : \twoheaduparrow_i$ and $\twoheaduparrow'_i$ contain the same references in the same order

Heap Representation

- a heap $h \mapsto$ a forest $(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n), (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\twoheaduparrow_1, \twoheaduparrow_2, \ldots, \twoheaduparrow_n) \approx (\twoheaduparrow'_1, \twoheaduparrow'_2, \ldots, \twoheaduparrow'_n)$$

iff $\forall i : \twoheaduparrow_i$ and $\twoheaduparrow'_i$ contain the same references in the same order

- the same general structure

Heap Representation

- a heap $h \mapsto$ a forest $(\text{⮝}_1, \text{⮝}_2, \ldots, \text{⮝}_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\text{⮝}_1, \text{⮝}_2, \ldots, \text{⮝}_n), (\text{⮝}'_1, \text{⮝}'_2, \ldots, \text{⮝}'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\text{⮝}_1, \text{⮝}_2, \ldots, \text{⮝}_{n}) \approx (\text{⮝}'_1, \text{⮝}'_2, \ldots, \text{⮝}'_{n})$$

iff $\forall i : \text{⮝}_i$ and $\text{⮝}'_i$ contain the same references in the same order

  - the same general structure

- for every class of $\mathcal{H}_{\approx}$:

$$\{(\text{⮝}_1, \text{⮝}_2, \ldots, \text{⮝}_n), (\text{⮝}'_1, \text{⮝}'_2, \ldots, \text{⮝}'_n), \ldots\}$$

Heap Representation

- a heap $h \mapsto$ a forest $(⇞_1, ⇞_2, \dots, ⇞_n)$
- a set of heaps $\mathcal{H} \mapsto \{(⇞_1, ⇞_2, \dots, ⇞_n), (⇞'_1, ⇞'_2, \dots, ⇞'_m), \dots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(⇞_1, ⇞_2, \dots, ⇞_n) \approx (⇞'_1, ⇞'_2, \dots, ⇞'_n)$$

iff $\forall i : ⇞_i$ and $⇞'_i$ contain the same references in the same order

- the same general structure

- for every class of $\mathcal{H}_\approx$:

$$\{(⇞_1, ⇞_2, \dots, ⇞_n), (⇞'_1, ⇞'_2, \dots, ⇞'_n), \dots\}$$

$$\rightsquigarrow$$

$$(\{⇞_1, ⇞'_1, \dots\}, \{⇞_2, ⇞'_2, \dots\}, \dots, \{⇞_n, ⇞'_n, \dots\})$$

Heap Representation

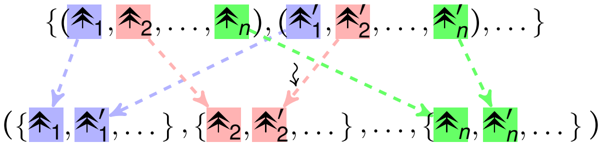
- a heap $h \mapsto$ a forest $(↟_1, ↟_2, \ldots, ↟_n)$
- a set of heaps $\mathcal{H} \mapsto \{(↟_1, ↟_2, \ldots, ↟_n), (↟'_1, ↟'_2, \ldots, ↟'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(↟_1, ↟_2, \ldots, ↟_n) \approx (↟'_1, ↟'_2, \ldots, ↟'_n)$$

iff $\forall i : ↟_i$ and $↟'_i$ contain the same references in the same order

  - the same general structure

- for every class of $\mathcal{H}_\approx$:



$$\{(↟_1, ↟_2, \ldots, ↟_n), (↟'_1, ↟'_2, \ldots, ↟'_n), \ldots\}$$

$$\wr$$

$$(\{↟_1, ↟'_1, \ldots\}, \{↟_2, ↟'_2, \ldots\}, \ldots, \{↟_n, ↟'_n, \ldots\})$$

Heap Representation

- a heap $h \mapsto$ a forest $(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n), (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_m), \dots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n) \approx (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_n)$$

iff $\forall i : \Uparrow_i$ and $\Uparrow'_i$ contain the same references in the same order

  - the same general structure

- for every class of $\mathcal{H}_\approx$:

$$\{(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n), (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_n), \dots\}$$

$$\rightsquigarrow$$

assume regular convex classes

$$(\{\Uparrow_1, \Uparrow'_1, \dots\}, \{\Uparrow_2, \Uparrow'_2, \dots\}, \dots, \{\Uparrow_n, \Uparrow'_n, \dots\})$$

Heap Representation

- a heap $h \mapsto$ a forest $(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n)$
- a set of heaps $\mathcal{H} \mapsto \{(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n), (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_m), \dots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_{\boxed{n}}) \approx (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_{\boxed{n}})$$

iff $\forall i : \Uparrow_i$ and $\Uparrow'_i$ contain the same references in the same order

  - the same general structure

- for every class of $\mathcal{H}_{\approx}$:

$$\{(\Uparrow_1, \Uparrow_2, \dots, \Uparrow_n), (\Uparrow'_1, \Uparrow'_2, \dots, \Uparrow'_n), \dots\}$$

$$\wr$$

assume regular convex classes

$$(\underbrace{\{\Uparrow_1, \Uparrow'_1, \dots\}}_{TA_1}, \underbrace{\{\Uparrow_2, \Uparrow'_2, \dots\}}_{TA_2}, \dots, \underbrace{\{\Uparrow_n, \Uparrow'_n, \dots\}}_{TA_n})$$

$$(\quad TA_1 \quad , \quad TA_2 \quad , \dots , \quad TA_n \quad)$$

Heap Representation

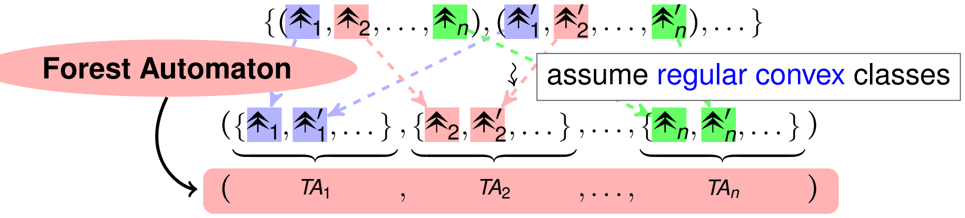
- a heap $h \mapsto$ a forest $(🌲_1, 🌲_2, \ldots, 🌲_n)$
- a set of heaps $\mathcal{H} \mapsto \{(🌲_1, 🌲_2, \ldots, 🌲_n), (🌲'_1, 🌲'_2, \ldots, 🌲'_m), \ldots\}$
  - partition $\mathcal{H}$ according to the $\approx$ relation:

$$(🌲_1, 🌲_2, \ldots, 🌲_{\boxed{n}}) \approx (🌲'_1, 🌲'_2, \ldots, 🌲'_{\boxed{n}})$$

iff $\forall i : 🌲_i$ and $🌲'_i$ contain the same references in the same order

- the same general structure

- for every class of $\mathcal{H}_{\approx}$:



$$\{(🌲_1, 🌲_2, \ldots, 🌲_n), (🌲'_1, 🌲'_2, \ldots, 🌲'_n), \ldots\}$$

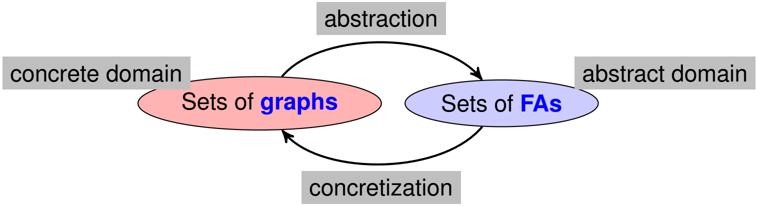
$$\wr$$

$$(\underbrace{\{🌲_1, 🌲'_1, \ldots\}}_{TA_1}, \underbrace{\{🌲_2, 🌲'_2, \ldots\}}_{TA_2}, \ldots, \underbrace{\{🌲_n, 🌲'_n, \ldots\}}_{TA_n})$$

$$(\quad TA_1 \quad , \quad TA_2 \quad , \ldots, \quad TA_n \quad)$$

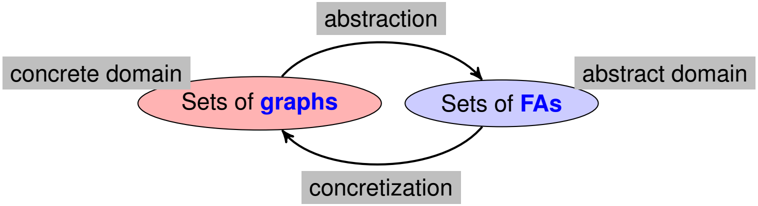
Symbolic Execution

- Abstract Interpretation



Symbolic Execution

- Abstract Interpretation

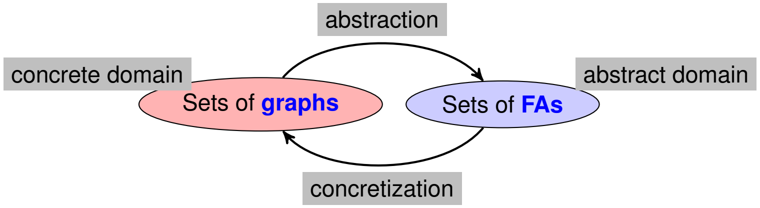


Statements

- `x := new T()`
- `delete(x)`
- `x := null`
- `x := y`
- `x := y.next`
- `x.next := y`
- `if/while (x == y)`

Symbolic Execution

- Abstract Interpretation



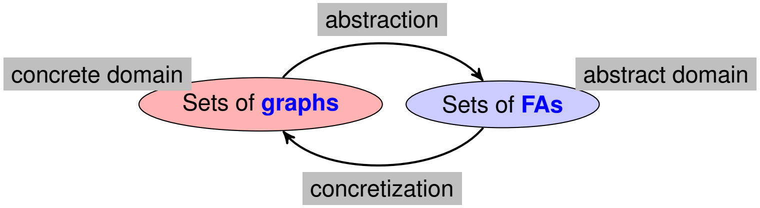
Statements

- `x := new T()`
- `delete(x)`
- `x := null`
- `x := y`
- `x := y.next`
- `x.next := y`
- `if/while (x == y)`

Abstract Transformers

Symbolic Execution

- Abstract Interpretation



Statements

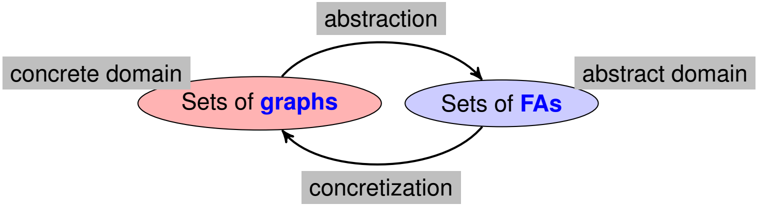
- `x := new T()`
- `delete(x)`
- `x := null`
- `x := y`
- `x := y.next`
- `x.next := y`
- `if/while (x == y)`

Abstract Transformers

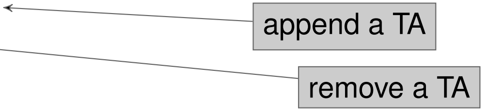
append a TA

Symbolic Execution

- Abstract Interpretation



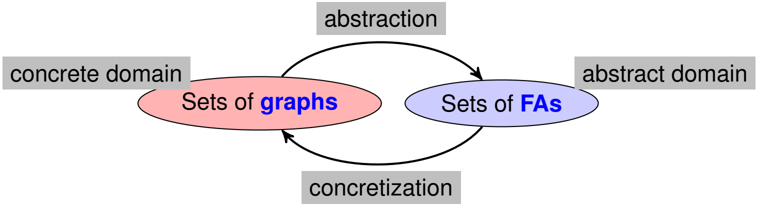
Statements **Abstract Transformers**

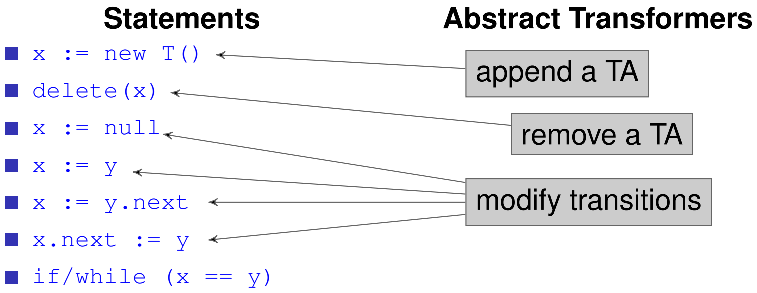


- `x := new T()`
- `delete(x)`
- `x := null`
- `x := y`
- `x := y.next`
- `x.next := y`
- `if/while (x == y)`

Symbolic Execution

- Abstract Interpretation

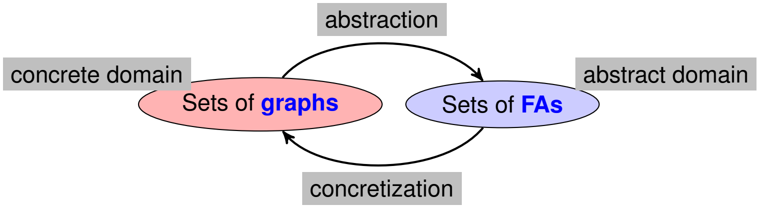


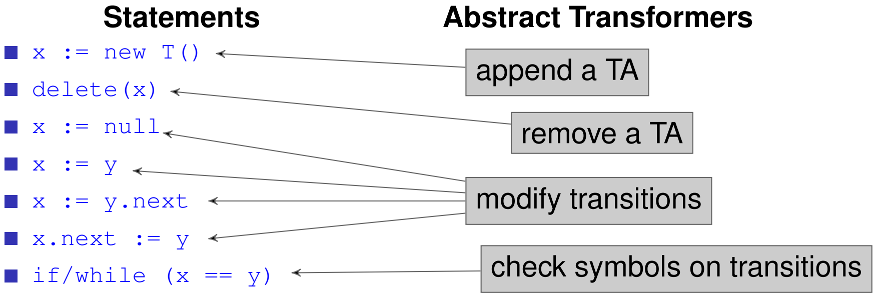


- `x := new T()`
- `delete(x)`
- `x := null`
- `x := y`
- `x := y.next`
- `x.next := y`
- `if/while (x == y)`

Symbolic Execution

- Abstract Interpretation





Statements — **Abstract Transformers**

- `x := new T()` ← append a TA
- `delete(x)` ← remove a TA
- `x := null` ← modify transitions
- `x := y` ← modify transitions
- `x := y.next` ← modify transitions
- `x.next := y` ← modify transitions
- `if/while (x == y)` ← check symbols on transitions

Widening

- abstraction on forest automaton $(TA_1, \ldots, TA_n)$

Widening

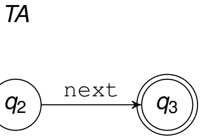
- abstraction on forest automaton $(TA_1, \ldots, TA_n)$
  - collapse states of component TAs $\leadsto$ $(TA_1^\alpha, \ldots, TA_n^\alpha)$

Widening

- abstraction on forest automaton $(TA_1, \ldots, TA_n)$
  - collapse states of component TAs $\leadsto$ $(TA_1^\alpha, \ldots, TA_n^\alpha)$
  - finite-height abstraction (from ARTMC)
    - collapse states with languages whose prefixes match up to height k

Widening

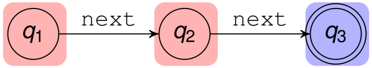
- abstraction on forest automaton $(TA_1, \ldots, TA_n)$
  - collapse states of component TAs $\rightsquigarrow$ $(TA_1^\alpha, \ldots, TA_n^\alpha)$
  - finite-height abstraction (from ARTMC)
    - collapse states with languages whose prefixes match up to height k



Widening

- abstraction on forest automaton $(TA_1, \ldots, TA_n)$
  - collapse states of component TAs $\rightsquigarrow$ $(TA_1^\alpha, \ldots, TA_n^\alpha)$
  - finite-height abstraction (from ARTMC)
    - collapse states with languages whose prefixes match up to height k

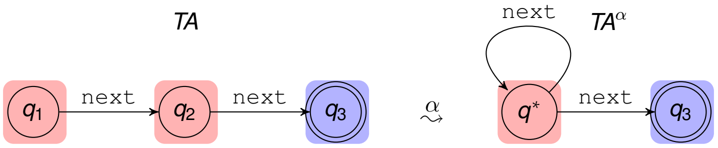
TA



$k = 1$

Widening

- abstraction on forest automaton $(TA_1, \dots, TA_n)$
  - collapse states of component TAs $\rightsquigarrow$ $(TA_1^\alpha, \dots, TA_n^\alpha)$
  - finite-height abstraction (from ARTMC)
    - collapse states with languages whose prefixes match up to height k



$k = 1$

Summary

The so-far-presented:

Summary

The so-far-presented:

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

Summary

$(\Uparrow_1, \Uparrow_2, \ldots, \Uparrow_n) \approx (\Uparrow'_1, \Uparrow'_2, \ldots, \Uparrow'_n)$
iff ...

The so-far-presented:

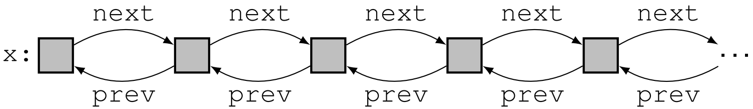
- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\rightsquigarrow \infty$ index of $\mathcal{H}_{\approx}$

Summary

$(\Lsh_1, \Lsh_2, \ldots, \Lsh_n) \approx (\Lsh'_1, \Lsh'_2, \ldots, \Lsh'_n)$
iff ...

The so-far-presented:

- ☺ works well for singly linked lists (SLLs), trees
- ☹ fails for more complex data structures
  - unbounded number of cut-points $\rightsquigarrow \infty$ index of $\mathcal{H}_{\approx}$



Summary

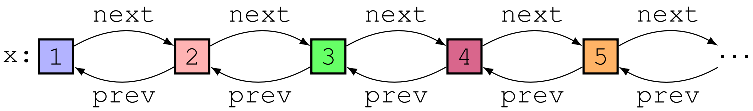
$(\Uparrow_1, \Uparrow_2, \ldots, \Uparrow_n) \approx (\Uparrow'_1, \Uparrow'_2, \ldots, \Uparrow'_n)$
iff ...

The so-far-presented:

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

☹ fails for more complex data structures

- unbounded number of cut-points $\leadsto \infty$ index of $\mathcal{H}_{\approx}$



Summary

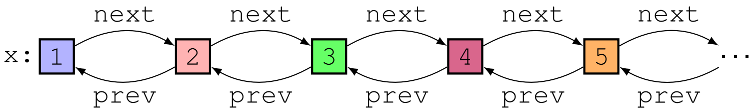
$(\Uparrow_1, \Uparrow_2, \ldots, \Uparrow_n) \approx (\Uparrow'_1, \Uparrow'_2, \ldots, \Uparrow'_n)$
iff ...

The so-far-presented:

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

☹ fails for more complex data structures

- unbounded number of cut-points $\rightsquigarrow$ ∞ index of $\mathcal{H}_{\approx}$



- doubly linked lists (DLLs), circular lists, nested 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

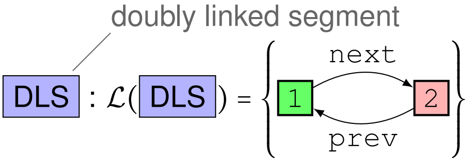
doubly linked segment

- Example: a box $\boxed{\text{DLS}}$

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

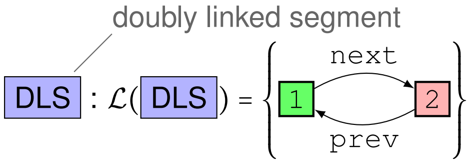
- Example: a box $\boxed{\text{DLS}}$: $\mathcal{L}(\boxed{\text{DLS}}) = $

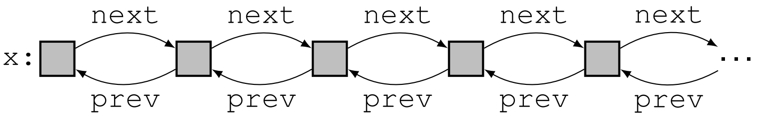


Hierarchical Forest Automata

- 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 $\boxed{\text{DLS}}$: $\mathcal{L}(\boxed{\text{DLS}}) =$

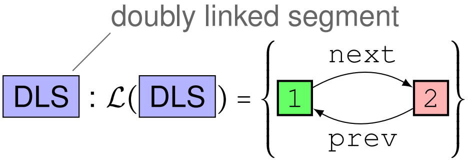


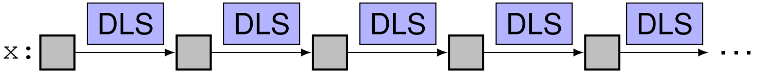


Hierarchical Forest Automata

- 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 $\boxed{\text{DLS}}$: $\mathcal{L}(\boxed{\text{DLS}}) = \{$ … $\}$

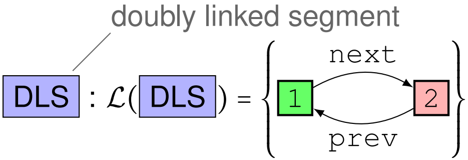


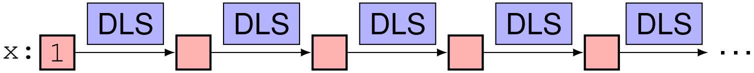


Hierarchical Forest Automata

- 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 $\boxed{\text{DLS}}$: $\mathcal{L}(\boxed{\text{DLS}}) = \left\{ \ \right\}$

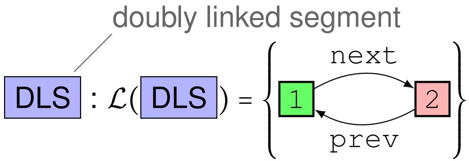


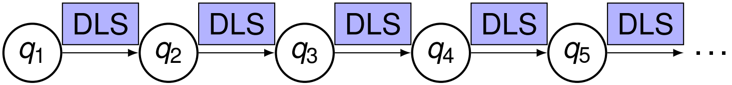


Hierarchical Forest Automata

- 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 $\boxed{\text{DLS}}$: $\mathcal{L}(\boxed{\text{DLS}}) = $



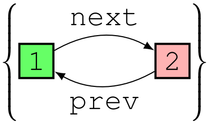


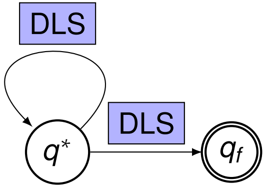
Hierarchical Forest Automata

- 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

doubly linked segment

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





Learning of Boxes

The Challenge

How to find “the right” boxes?

Learning of Boxes

The Challenge

How to find “the right” boxes?

- CAV’11 — database of boxes
- CAV’13 — automatic discovery

Learning of Boxes

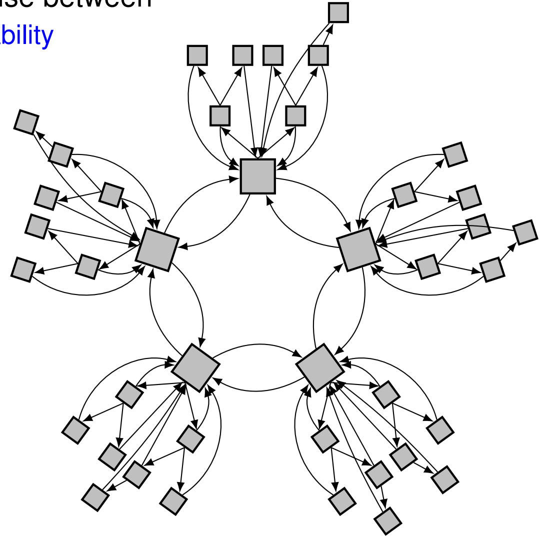
- compromise between

Learning of Boxes

- compromise between
  - reusability

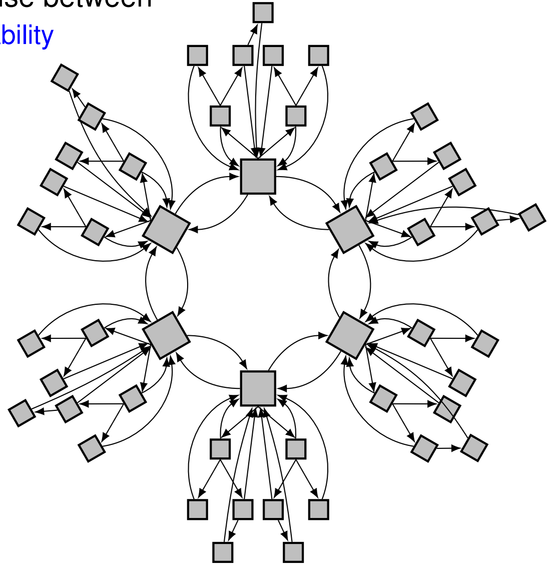
Learning of Boxes

- compromise between
  - reusability



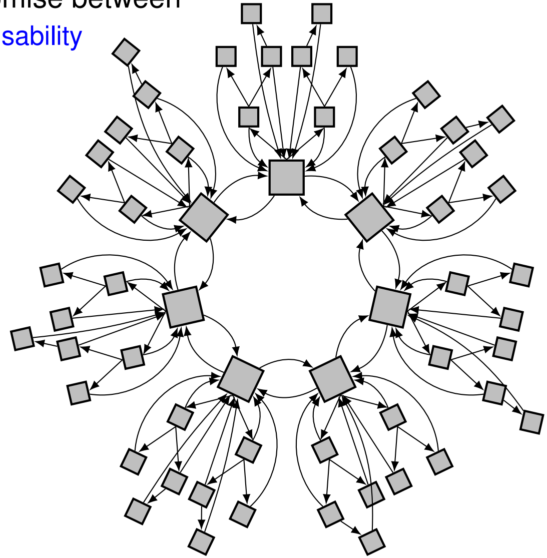
Learning of Boxes

- compromise between
  - reusability



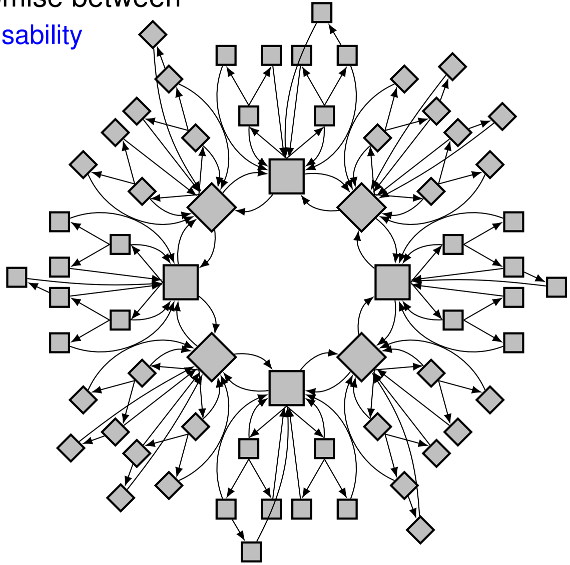
Learning of Boxes

- compromise between
  - reusability



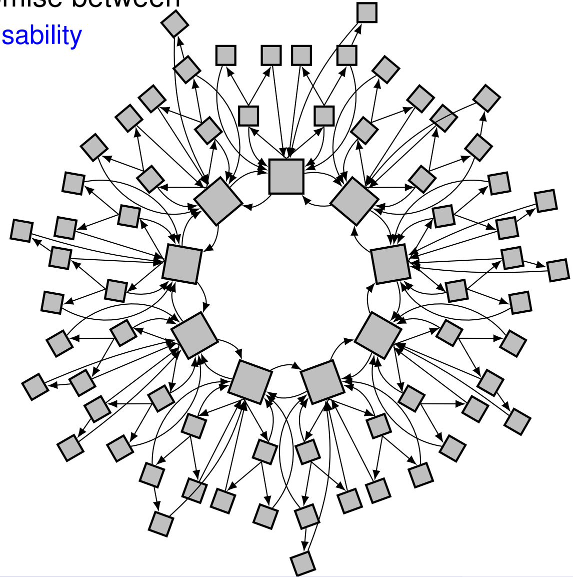
Learning of Boxes

- compromise between
  - reusability



Learning of Boxes

- compromise between
  - reusability



Learning of Boxes

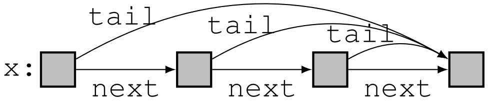
- compromise between
  - reusability

Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points

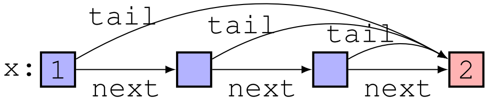
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



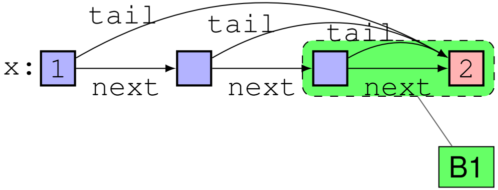
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



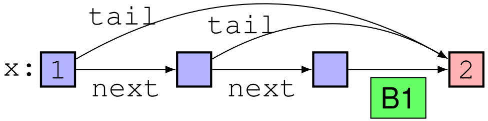
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



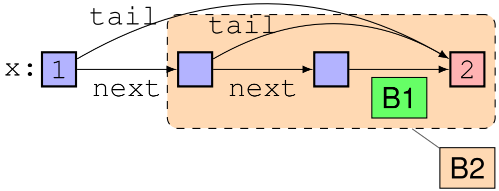
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



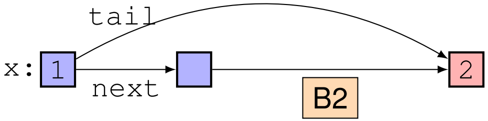
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



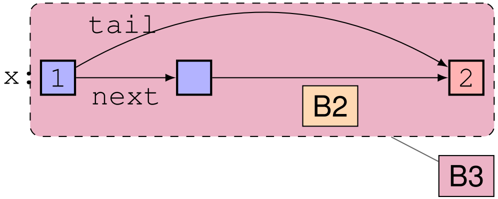
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



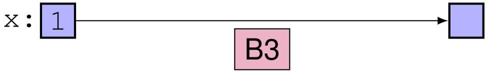
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



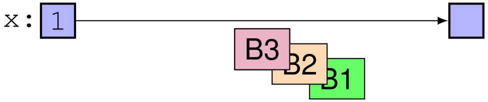
Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



Learning of Boxes

- compromise between
  - reusability
  - ability to hide cut-points



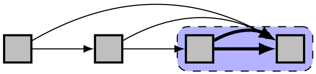
Learning of Boxes: Knots

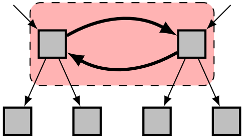
Knots

Learning of Boxes: Knots

Knots

1. smallest subgraphs meaningful to be folded:

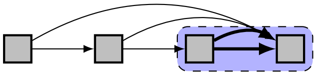


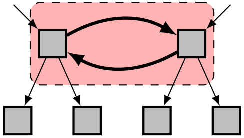


Learning of Boxes: Knots

Knots

1. smallest subgraphs meaningful to be folded:



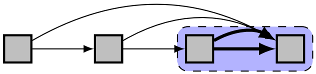


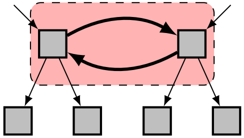
2. handle inputs/outputs

Learning of Boxes: Knots

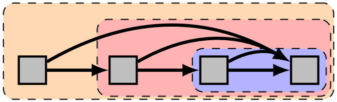
Knots

1. smallest subgraphs meaningful to be folded:





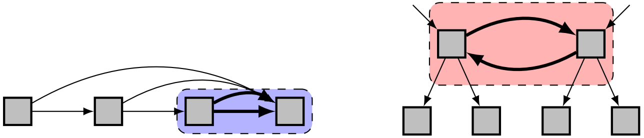
2. handle inputs/outputs
   - join intersecting knots



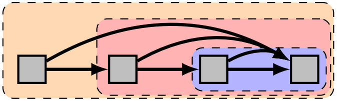
Learning of Boxes: Knots

Knots

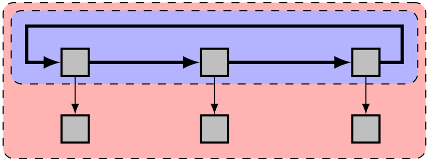
1. smallest subgraphs meaningful to be folded:



2. handle inputs/outputs
   - join intersecting knots



   - enclose paths from inner nodes to leaves

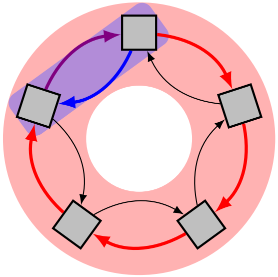


Learning of Boxes: Knots

3. complexity

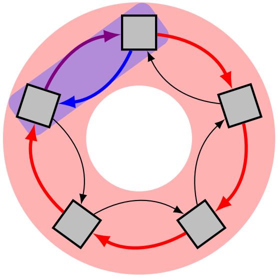
Learning of Boxes: Knots

3. complexity



Learning of Boxes: Knots

3 complexity



- find basic knots with $1, 2, \ldots$ cut-points

Widening Revisited

- learning and folding of boxes in the abstraction loop

Widening Revisited

- learning and folding of boxes in the abstraction loop

The Goal

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

$\Rightarrow$ hide unboundedly many cut-points

Widening Revisited

- learning and folding of boxes in the abstraction loop

The Goal

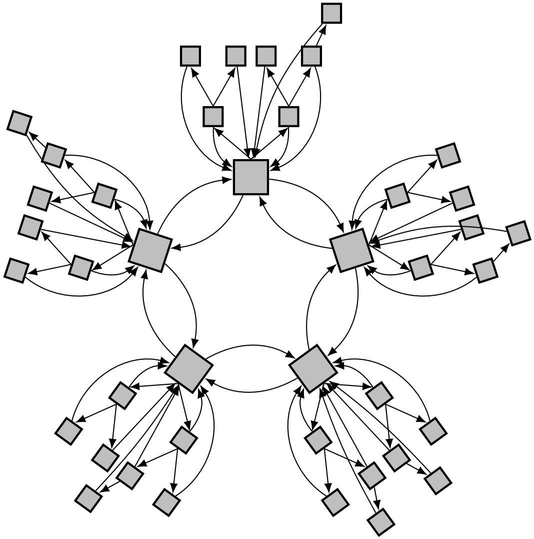
Fold boxes that will, after abstraction, appear on cycles of automata.

⇒ hide unboundedly many cut-points

```
1 Algorithm: Abstraction Loop
2 Unfold solo boxes
3 repeat
4     Abstract
5     Fold
6 until fixpoint
```

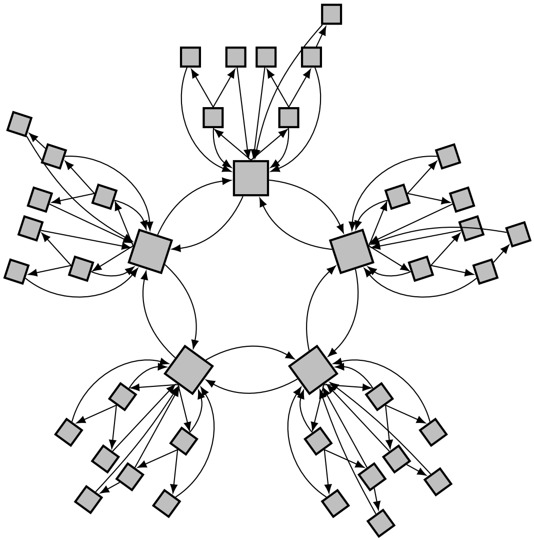
not on a cycle

Learning of Boxes: Example



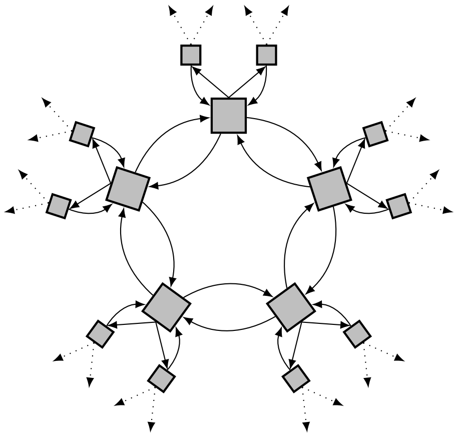
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



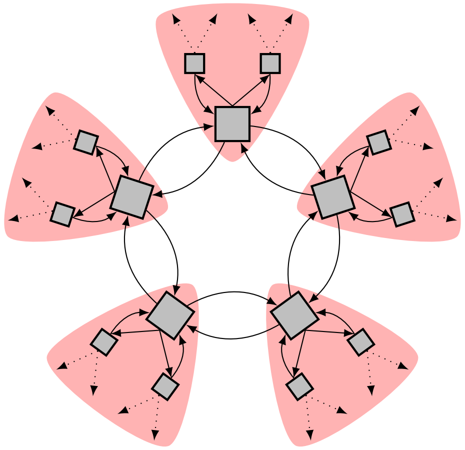
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



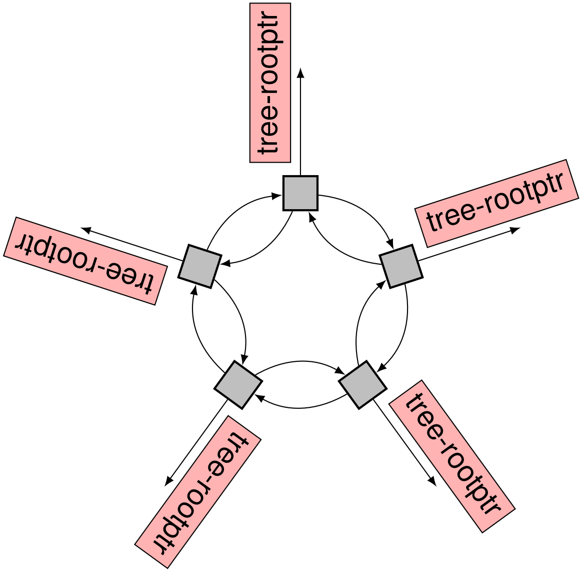
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



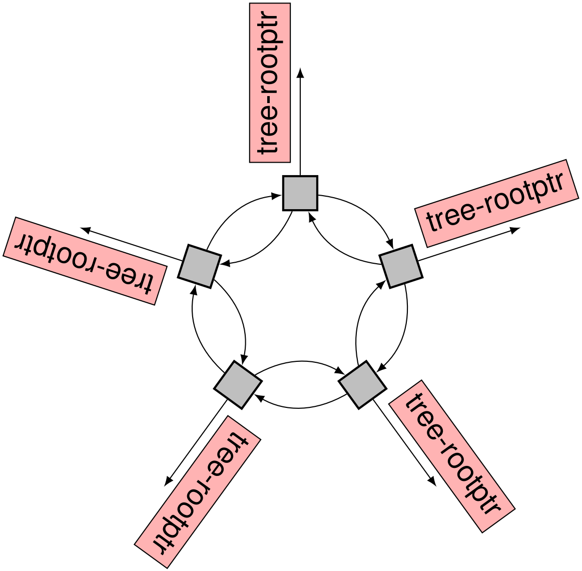
```
1 Unfold solo boxes
2 repeat
3       Abstract
4       Fold
5 until fixpoint
```

Learning of Boxes: Example



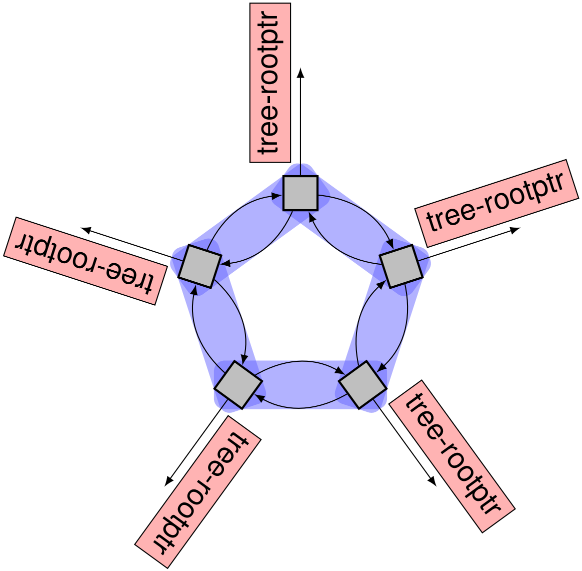
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



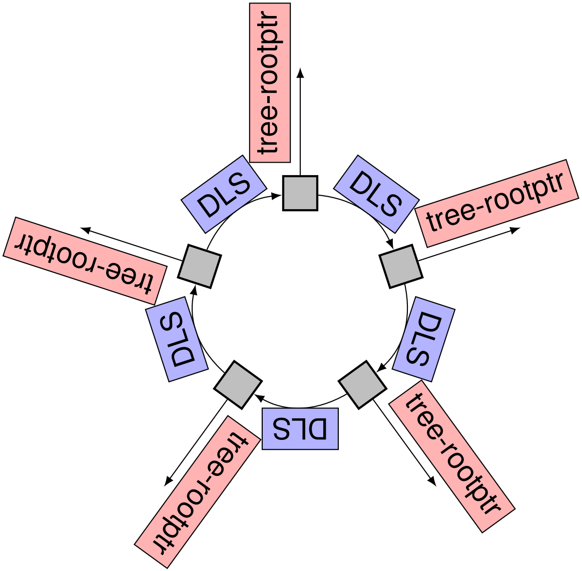
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



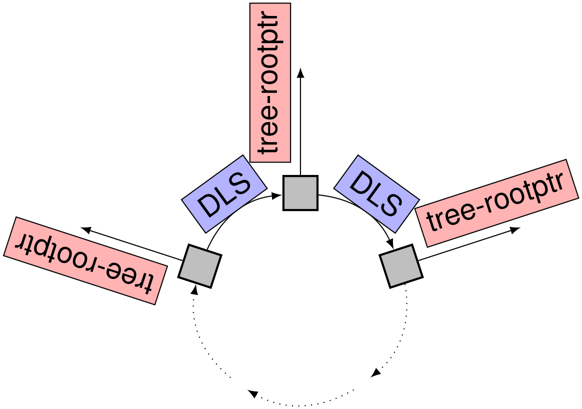
```
1  Unfold solo boxes
2  repeat
3      Abstract
4      Fold
5  until fixpoint
```

Learning of Boxes: Example



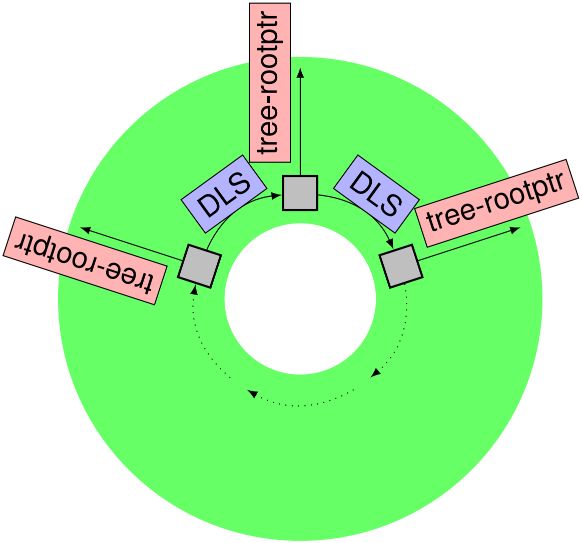
```
1  Unfold solo boxes
2  repeat
3      Abstract
4      Fold
5  until fixpoint
```

Learning of Boxes: Example



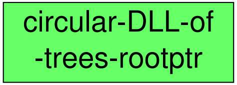
```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Learning of Boxes: Example



```
1 Unfold solo boxes
2 repeat
3     Abstract
4     Fold
5 until fixpoint
```

Experimental Results

- implemented in **Forester** tool

Experimental Results

- implemented in **Forester** tool
- comparison with Predator (state-of-the-art tool for lists)
    - winner of HeapManipulation and MemorySafety of SV-COMP'13

Experimental Results

- implemented in **Forester** tool
- comparison with Predator (state-of-the-art tool for lists)
  - 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
- not covered here:
  - support for pointer arithmetic
  - 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, . . .