Automata in Infinite-State Formal Verification

Ondřej Lengál

Advisor: prof. Ing. Tomáš Vojnar, Ph.D.
(Co-supervised by: Mgr. Lukáš Holík, Ph.D.)

Faculty of Information Technology
Brno University of Technology

Scope of the Thesis

Formal verification of programs with complex dynamic data structures,

- e.g. lists, trees, skip lists, . . .
- used in OS kernels, standard libraries, . . .

decision procedures of logics:

- WS1S, separation logic,

using the theory of automata,

- $\rightsquigarrow$ development of efficient automata manipulation techniques.

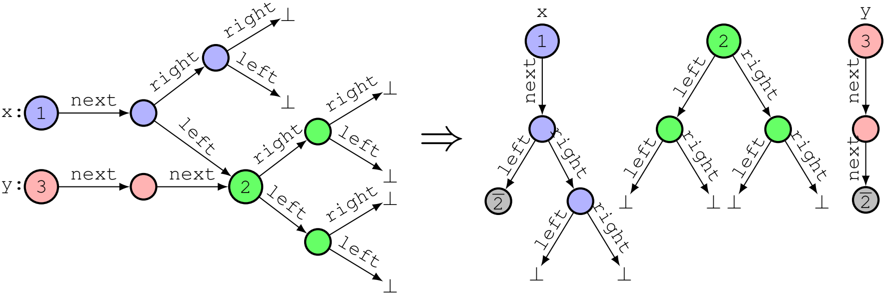
Forest Automata-based Verification

- Verification of memory-safety of heap-manipulating programs,
- infinitely many heap configurations $\rightsquigarrow$ symbolic representation,
- representation mostly based on logics, graphs, automata.

Forest Automata-based Verification

Our approach:

- decompose heap into **cutpoint-free** tree components (a forest)

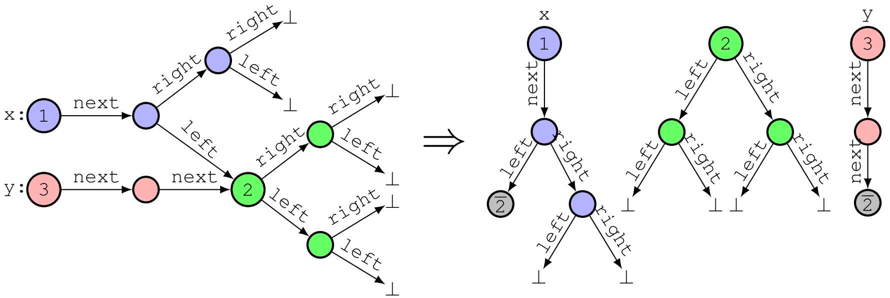


a) a graph, and b) its **forest** representation

Forest Automata-based Verification

Our approach:

- decompose heap into **cutpoint-free** tree components (a forest)



a) a graph, and

b) its **forest** representation

- sets of heaps:
  - collect 1st, 2nd, . . . trees from all forests into sets of trees,
  - represent each set of trees by a tree automaton,
  - tuple of tree automata $\rightsquigarrow$ a **forest automaton**: $FA = (TA_1, \ldots, TA_n)$.

Forest Automata-based Verification

The analysis:

- based on abstract interpretation:
- for every line of code, compute forest automata representing reachable heap configurations at this line, until fixpoint,
- program statements are substituted by abstract transformers performing the corresponding operation on forest automata,
- at loop points, do widening (over-approximation).

Forest Automata-based Verification

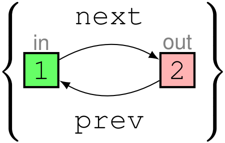
- Hierarchical Forest Automata
  - deal with families of graphs with unbounded number of cutpoints,
    - doubly linked lists, skip lists, **red-black** trees, ...
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - intuition: replace repeated subgraphs by a symbol, hide cut-points

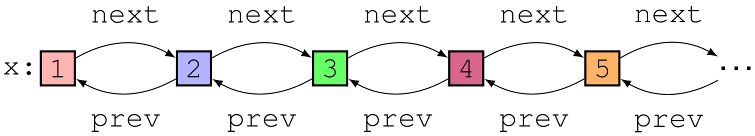
Forest Automata-based Verification

- Hierarchical Forest Automata
  - deal with families of graphs with unbounded number of cutpoints,
    - doubly linked lists, skip lists, **red-black** trees, . . .
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - intuition: replace repeated subgraphs by a symbol, hide cut-points

doubly linked segment

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

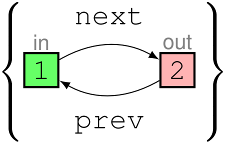


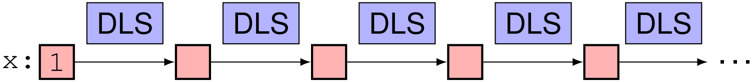


Forest Automata-based Verification

- Hierarchical Forest Automata
  - deal with families of graphs with unbounded number of cutpoints,
    - doubly linked lists, skip lists, **red-black** trees, . . .
  - FAs are symbols (**boxes**) of FAs of a higher level
  - a hierarchy of FAs
  - intuition: replace repeated subgraphs by a symbol, hide cut-points

doubly linked segment

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



Result 1

Fully Automated Shape Analysis with Forest Automata

Fully Automated Shape Analysis with Forest Automata

The need to construct automatically a good hierarchy of boxes;

- finding the right boxes is hard,

Contribution:

- an algorithm that finds suitable subgraphs to fold into boxes,
- works for a large class of data structures
  - (nested) lists, trees, skip lists, . . .

Fully Automated Shape Analysis with Forest Automata

The need to construct automatically a good hierarchy of boxes;

- finding the right boxes is hard,

Contribution:

- an algorithm that finds suitable subgraphs to fold into boxes,
- works for a large class of data structures
  - (nested) lists, trees, skip lists, . . .

Suitable subgraphs: a compromise:

- smaller subgraphs are better,
  - can be reused,
- bigger subgraphs are better,
  - can hide cutpoints,
- $\rightsquigarrow$ find small enough subgraphs that effectively hide cutpoints.

Fully Automated Shape Analysis with FAs—Results

- implemented in **Forester** tool

Table: comparison with Predator (many SV-COMP medals) [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
$\text{SLL}_{\text{Linux}}$	0.03	0.03	tree	0.14	Err
SLL of CSLLs	0.73	0.12	tree+parents	0.21	T
SLL of $\text{2CDLLs}_{\text{Linux}}$	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 (T) — false positive (Err)

- Holík, Lengál, Rogalewicz, Šimáček, and Vojnar. Fully Automated Shape Analysis Based on Forest Automata. In *Proc. of CAV'13*, LNCS 8044.

Result 2

Verification of Heap Programs with Ordered Data

Verification of Heap Programs with Ordered Data

Sometimes, correctness of programs manipulating heap depends on relations among data values stored inside,

- verification of sorting algorithms, search trees, skip lists, ...

Contribution:

- extension of the formalism of FAs with ordering constraints,
- extension of the FA-based shape analysis for the extended FAs.

Verification of Heap Programs with Ordered Data

2 **types** of constraints:

- Local:
  - stored in symbols of tree automata,
  - encode relations between neighbouring nodes.

$$q \to a(r, s) : 0 \prec 1$$

- Global:
  - stored separately,
  - encode relations between distant nodes.

$$TA_1 \prec TA_2$$

Verification of Heap Programs with Ordered Data

2 **types** of constraints:

- Local:
  - stored in symbols of tree automata,
  - encode relations between neighbouring nodes.

$$q \to a(r, s) : 0 \prec 1$$

- Global:
  - stored separately,
  - encode relations between distant nodes.

$$TA_1 \prec TA_2$$

2 **scopes** of constraints:

- root-root $\prec_{rr}$: relation between 2 nodes,
- root-all $\prec_{ra}$: relation between node and all nodes in a (sub)tree.

Verification of Heap Programs with Ordered Data

2 **types** of constraints:

- Local:
  - stored in symbols of tree automata,
  - encode relations between neighbouring nodes.

$$q \to a(r, s) : 0 \prec 1$$

- Global:
  - stored separately,
  - encode relations between distant nodes.

$$TA_1 \prec TA_2$$

2 **scopes** of constraints:

- root-root $\prec_{rr}$: relation between 2 nodes,
- root-all $\prec_{ra}$: relation between node and all nodes in a (sub)tree.

Modification of analysis loop, abstraction, equivalence checking.

Verif. of Heap Programs with Ordered Data—Results

Table: Results of the experiments with the data extension of Forester

Example	time [s]	Example	time [s]
SLL insert	0.06	SL_2 insert	9.65
SLL delete	0.08	SL_2 delete	10.14
SLL reverse	0.07	SL_3 insert	56.99
SLL bubblesort	0.13	SL_3 delete	57.35
SLL insertsort	0.10		
DLL insert	0.14	BST insert	6.87
DLL delete	0.38	BST delete	15.00
DLL reverse	0.16	BST left rotate	7.35
DLL bubblesort	0.39	BST right rotate	6.25
DLL insertsort	0.43		

- Abdulla, Holík, Jonsson, Lengál, Trinh, and Vojnar. Verification of Heap Manipulating Programs with Ordered Data by Extended FAs. In *Proc. of ATVA'13*, LNCS 8172.

Result 3

Separation Logic

Decision Procedure for Separation Logic

Separation Logic:

- alternative way to reason about programs with dynamic memory.

Formulae:

$$\varphi = \Pi \wedge \Sigma$$

- Π: pure part (aliasing of variables: $X = Y, X \neq Y, \wedge$),
- Σ: shape part (structure of heap: $X \mapsto \{n : Y, p : Z\}, P(X, Y), *$).

Decision Procedure for Separation Logic

Separation Logic:

- alternative way to reason about programs with dynamic memory.

Formulae:

$$\varphi = \Pi \wedge \Sigma$$

- Π: pure part (aliasing of variables: $X = Y, X \neq Y, \wedge$),
- Σ: shape part (structure of heap: $X \mapsto \{n : Y, p : Z\}, P(X, Y), *$).

Entailment checking $\psi \overset{?}{\models} \varphi$:

- resolving verification conditions in deductive verification,
- fixpoint checking in abstract interpretation-based approaches,
- in general undecidable.

Decision Procedure for Separation Logic

Separation Logic:

- alternative way to reason about programs with dynamic memory.

Formulae:

$$\varphi = \Pi \wedge \Sigma$$

- Π: pure part (aliasing of variables: $X = Y, X \neq Y, \wedge$),
- Σ: shape part (structure of heap: $X \mapsto \{n : Y, p : Z\}, P(X, Y), *$).

Entailment checking $\psi \overset{?}{\models} \varphi$:

- resolving verification conditions in deductive verification,
- fixpoint checking in abstract interpretation-based approaches,
- in general undecidable.

Contribution:

- a decision procedure for a practical fragment:
  - lists (singly/doubly linked, nested, cyclic, skip lists, . . .),
- transforms the problem to checking TA membership.

Decision Procedure for Separation Logic

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \land \Sigma_\varphi}_{\varphi} \stackrel{?}{\models} \underbrace{\Pi_\psi \land \Sigma_\psi}_{\psi}$$

1. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)

Decision Procedure for Separation Logic

$$\underbrace{\exists \overrightarrow{X} . \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)
2. Test entailment of points-to $X \mapsto \{\dots\}$ in Σ_ψ and Σ_φ

Decision Procedure for Separation Logic

$$\underbrace{\exists \overrightarrow{X} . \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)
2. Test entailment of points-to $X \mapsto \{\dots\}$ in Σ_ψ and Σ_φ
3. Reduce the rest of Σ_φ and Σ_ψ to

$$\varphi_1 \overset{?}{\models} P_1 \quad \wedge \quad \varphi_2 \overset{?}{\models} P_2 \quad \wedge \quad \varphi_3 \overset{?}{\models} P_3 \quad \wedge \quad \dots$$

   1. Transform $\varphi_i \rightsquigarrow$ tree $\mathcal{T}_{\varphi_i}$
      - spanning tree + routing expressions
   2. Transform $P_i \rightsquigarrow$ tree automaton $\mathcal{A}_{P_i}$
      - all unfoldings of P_i
   3. Test

$$\mathcal{T}_{\varphi_i} \overset{?}{\in} \mathcal{L}(\mathcal{A}_{P_i})$$

Decision Procedure for Separation Logic—Results

Table: Results of SL-COMP'14

a) Results for extended acyclic lists (43 tasks)

Solver	Errors	Solved	Time
SPEN	0	43	0.61
Cyclist-SL	0	19	141.78
SLIDE	0	0	0.00
SLEEK-06	1	31	43.65

b) Results for singly linked lists

Solver	sll0a_entl (292 tasks)			sll0a_sat (110 tasks)		
	Errors	Solved	Time	Errors	Solved	Time
Asterix	0	292	2.98	0	110	1.06
SPEN	0	292	7.58	0	110	3.27
SLEEK-06	0	292	14.13	0	110	4.99
Cyclist-SL	0	55	11.78	55	55	0.55

- Enea, Lengál, Sighireanu, Vojnar. Compositional Entailment Checking for a Fragment of Separation Logic. In *Proc. of APLAS'14*, LNCS 8858.

Result 4

WS1S

Decision Procedure for WS1S

WS1S:

- 2^{nd}-order monadic logic over $\mathbb{N}$ with successor relation,
- a natural means for describing regular languages [Büchi'59],
  - logical connectives and $\exists$ quantif. $\mapsto$ set operations + projection,
- powerful, yet still decidable (out of **ELEMENTARY** though!),

state-of-the-art approach (MONA tool):

- decision procedure translating formulae to deterministic automata,
- every quantifier alternation yields complementation,
- projection yields nondeterminism $\to$ determinisation,
- $\leadsto$ exponential blow-up.

Decision Procedure for WS1S

Contribution:

- a decision procedure based on nondeterministic automata,
  - avoids full-scale determinisation,
- optimises evaluation of quantifier alternations,
  - the source of state explosion,
- uses symbolic terms to represent nested sets of states,
  - similar to the Antichains algorithm for testing NFA universality,
- new insights into the used NFA framework,
  - $\rightsquigarrow$ future work: exploration of more general structure of terms.

Decision Procedure for WS1S—Results (1/2)

Table: Results for practical formulae

Benchmark	Time [s]		Space [states]	
	MONA	**dWiNA**	MONA	**dWiNA**
`reverse-before-loop`	0.01	0.01	179	47
`insert-in-loop`	0.01	0.01	463	110
`bubblesort-else`	0.01	0.01	1 285	271
`reverse-in-loop`	0.02	0.02	1 311	274
`bubblesort-if-else`	0.02	0.23	4 260	1 040
`bubblesort-if-if`	0.12	1.14	8 390	2 065

- obtained from the decision procedure of STRAND

Decision Procedure for WS1S—Results (2/2)

Table: Results for generated formulae

k	Time [s]		Space [states]	
	MONA	**dWiNA**	MONA	**dWiNA**
1	0.11	0.01	10 718	39
2	0.20	0.01	25 517	44
3	0.57	0.01	60 924	50
4	1.79	0.02	145 765	58
5	4.98	0.02	349 314	70
6	∞	0.47	∞	90

- based on a formula expressing existence of an ascending chain of n sets ordered w.r.t. $\subset$,
- k — the number of quantifier alternations.

- Fiedor, Holík, Lengál, and Vojnar. Nested Antichains for WS1S. In *Proc. of TACAS'15*, LNCS 9035.

Result 5

Tree Automata Downward Inclusion Checking

Downward Inclusion Checking of TAs

The need to efficiently manipulate nondeterministic tree automata:

- including checking language inclusion,
- current approach: upward inclusion checking,
  - based on constructing deterministic bottom-up automaton,
  - uses the principle of Antichains to prune the searched space,
  - compatible with upward simulation (yet more pruning),
  - incompatible with (usually richer) downward simulation.

Downward Inclusion Checking of TAs

The need to efficiently manipulate nondeterministic tree automata:

- including checking language inclusion,
- current approach: upward inclusion checking,
  - based on constructing deterministic bottom-up automaton,
  - uses the principle of Antichains to prune the searched space,
  - compatible with upward simulation (yet more pruning),
  - incompatible with (usually richer) downward simulation.

Contribution:

- downward inclusion checking algorithm,
- traverses the automata downwards,
- uses ideas from Antichains to prune searched space
- can use downward simulation,
- later extended with another antichain optimisation,
- in many cases superior.

Downward Inclusion Checking of TAs—Results

Table: Results of the experiments with downward inclusion checking

Algorithm	All pairs		$\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$		$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$	
	Winner	Timeouts	Winner	Timeouts	Winner	Timeouts
down	36.35 %	32.51 %	39.85 %	26.01 %	0.00 %	90.80 %
down+s	4.15 %	18.27 %	0.00 %	20.31 %	47.28 %	0.00 %
down-op	32.20 %	32.51 %	35.30 %	26.01 %	0.00 %	90.80 %
down-op+s	3.15 %	18.27 %	0.00 %	20.31 %	35.87 %	0.00 %
up	24.14 %	0.00 %	24.84 %	0.00 %	16.85 %	0.00 %
up+s	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %

- Holík, Lengál, Šimáček, and Vojnar. Efficient Inclusion Checking on Explicit and Semi-Symbolic TAs. In *Proc. of ATVA'11*, LNCS 6996.

Result 6

An Efficient Library for Nondeterministic Automata

An Efficient Library for Nondeterministic Automata

Contribution:

- VATA: A highly efficient library for nondeterministic automata,
- word automata, tree automata,
- implementation of state-of-the-art algorithms,
  - inclusion checking, simulation computation, ...
- explicit/semi-symbolic representation,
  - semi-symbolic uses BDDs,
- open & free: being used by a number of researchers.

- Lengál, Šimáček, and Vojnar. VATA: A Library for Efficient Manipulation of Non-Deterministic TAs. In *Proc. of TACAS'12*, LNCS 7214.

Possible Directions for Future Research

Forest automata-based shape analysis:

- refinable abstraction (WIP),
- support for analysis of incomplete programs.

Possible Directions for Future Research

Forest automata-based shape analysis:

- refinable abstraction (WIP),
- support for analysis of incomplete programs.

Separation logic:

- extend the procedure to tree data structures.

Possible Directions for Future Research

Forest automata-based shape analysis:

- refinable abstraction (WIP),
- support for analysis of incomplete programs.

Separation logic:

- extend the procedure to tree data structures.

WS1S:

- extension to generalized symbolic terms (WIP),
- extension to WS*k*S (WIP).

Possible Directions for Future Research

Forest automata-based shape analysis:

- refinable abstraction (WIP),
- support for analysis of incomplete programs.

Separation logic:

- extend the procedure to tree data structures.

WS1S:

- extension to generalized symbolic terms (WIP),
- extension to WS*k*S (WIP).

Efficient techniques for manipulating automata:

- manipulation of symbolically represented automata (WIP),
- finding new techniques for checking language inclusion.

Publications

Journal:

- Abdulla, Holík, Jonsson, Lengál, Trinh, and Vojnar. Verification of Heap Manipulating Programs with Ordered Data by Extended FAs. *Acta Informatica*. 2015.

Conference:

- Fiedor, Holík, Lengál, and Vojnar. Nested Antichains for WS1S. In *Proc. of TACAS'15*, LNCS 9035.
- Abdulla, Holík, Jonsson, Lengál, Trinh, and Vojnar. Verification of Heap Manipulating Programs with Ordered Data by Extended FAs. In *Proc. of ATVA'13*, LNCS 8172.
- Holík, Lengál, Rogalewicz, Šimáček, and Vojnar. Fully Automated Shape Analysis Based on Forest Automata. In *Proc. of CAV'13*, LNCS 8044.
- Enea, Lengál, Sighireanu, and Vojnar. Compositional Entailment Checking for a Fragment of Separation Logic. In *Proc. of APLAS'14*, LNCS 8858.
- Lengál, Šimáček, and Vojnar. VATA: A Library for Efficient Manipulation of Non-Deterministic Tree Automata. In *Proc. of TACAS'12*, LNCS 7214.
- Holík, Lengál, Šimáček, and Vojnar. Efficient Inclusion Checking on Explicit and Semi-Symbolic Tree Automata. In *Proc. of ATVA'11*, LNCS 6996,

Other:

- 5 conference papers, 6 technical reports, 1 monography, 5 software tools