Automata in Software Verification and Testing

Martin Hruška
supervisors: Tomáš Vojnar, Lukáš Holík

Brno University of Technology, Czech Republic

March 5, 2024

Introduction

- Two main topics of the thesis:
  - Shape analysis (majority of the work).
  - Automated software testing (described later).

Introduction

- Two main topics of the thesis:
  - Shape analysis (majority of the work).
  - Automated software testing (described later).
- Shape analysis
  - Formal verification of programs with dynamic data structures.

Introduction

- Two main topics of the thesis:
  - Shape analysis (majority of the work).
  - Automated software testing (described later).
- Shape analysis
  - Formal verification of programs with dynamic data structures.
  - Goal: Derive reachable shapes of dynamic data structures (so called shape invariant).

Introduction

- Two main topics of the thesis:
  - Shape analysis (majority of the work).
  - Automated software testing (described later).
- Shape analysis
  - Formal verification of programs with dynamic data structures.
  - Goal: Derive reachable shapes of dynamic data structures (so called shape invariant).
  - Can be used to find bugs such as invalid memory dereferences, invalid frees, memory leaks.
  - When an error is found, the method should provide a counterexample.
  - Have to to deal with infinite state spaces.

Introduction

- Two main topics of the thesis:
  - Shape analysis (majority of the work).
  - Automated software testing (described later).
- Shape analysis
  - Formal verification of programs with dynamic data structures.
  - Goal: Derive reachable shapes of dynamic data structures (so called shape invariant).
  - Can be used to find bugs such as invalid memory dereferences, invalid frees, memory leaks.
  - When an error is found, the method should provide a counterexample.
  - Have to to deal with infinite state spaces.
  - Application: Proving correctness of critical software systems (e.g., operating system kernel).

Automata Theory in Shape Analysis

- Data structures can be viewed as graphs.
- Graphs can be accepted by automata.
- An automaton can represent a set of graphs.
- Goal of analysis: Derive an automaton for each program location representing all possible shapes of data structures at the location,
- Applied in Abstract Regular Tree Model Checking (ARTMC).

Automata Theory in Shape Analysis

- Data structures can be viewed as graphs.
- Graphs can be accepted by automata.
- An automaton can represent a set of graphs.
- Goal of analysis: Derive an automaton for each program location representing all possible shapes of data structures at the location,
- Applied in Abstract Regular Tree Model Checking (ARTMC).

Advantages

- Existing efficient algorithms for handling finite automata.
- Genericity — can represent various kinds of data structures.

Counterexample Validation and Abstraction Refinement for Shape Analysis based on Forest Automata

Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Vojnar, T.: Counterexample Validation and Interpolation-Based Refinement for Forest Automata. In *Proc. of VMCAI'17*, LNCS 10145, Springer, 2017.

Forest Automata Encoding of Heap

- A Forest Automaton (FA)* is a tuple of tree automata (TA).

* P. Habermehl, L. Holík, J. Šimáček, A. Rogalewicz, and T. Vojnar. *Forest Automata for Verification of Heap Manipulation*. FMSD, 41(1):83–106, Springer, 2012.

Forest Automata Encoding of Heap

- A Forest Automaton (FA)* is a tuple of tree automata (TA).
- A TA accepts trees.

*P. Habermehl, L. Holík, J. Šimáček, A. Rogalewicz, and T. Vojnar. Forest Automata for Verification of Heap Manipulation. FMSD, 41(1):83–106, Springer, 2012.

Forest Automata Encoding of Heap

- A Forest Automaton (FA)* is a tuple of tree automata (TA).
- A TA accepts trees.
- Trees can contain leaves referencing the roots of other trees.

* P. Habermehl, L. Holík, J. Šimáček, A. Rogalewicz, and T. Vojnar. *Forest Automata for Verification of Heap Manipulation.* FMSD, 41(1):83–106, Springer, 2012.

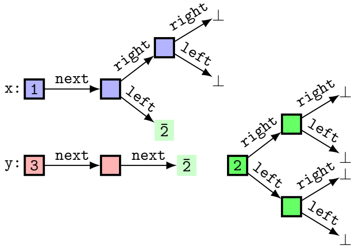
Forest Automata Encoding of Heap

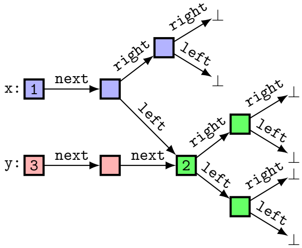
- A Forest Automaton (FA)* is a tuple of tree automata (TA).
- A TA accepts trees.
- Trees can contain leaves referencing the roots of other trees.
- An FA $F = (TA_1, \ldots, TA_n)$ represents tree decompositions (tuples of trees $t_1, \ldots, t_n$) of heap graphs such that $\forall 1 \leq i \leq n : t_i \in L(TA_i)$.

*P. Habermehl, L. Holík, J. Šimáček, A. Rogalewicz, and T. Vojnar. Forest Automata for Verification of Heap Manipulation. FMSD, 41(1):83–106, Springer, 2012.

Forest Automata Encoding of Heap

- A Forest Automaton (FA)* is a tuple of tree automata (TA).
- A TA accepts trees.
- Trees can contain leaves referencing the roots of other trees.
- An FA $F = (TA_1, \ldots, TA_n)$ represents tree decompositions (tuples of trees $t_1, \ldots, t_n$) of heap graphs such that $\forall 1 \leq i \leq n : t_i \in L(TA_i)$.
- Encoded heap graphs obtained by connecting leaves with the referenced roots.

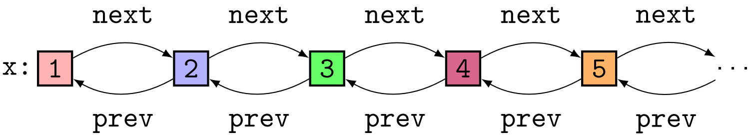


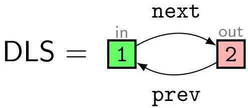


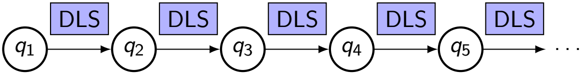
* P. Habermehl, L. Holík, J. Šimáček, A. Rogalewicz, and T. Vojnar. Forest Automata for Verification of Heap Manipulation. FMSD, 41(1):83–106, Springer, 2012.

Boxes

- Boxes are used to extend the expressive power of FA.
- A box is an FA that can be used as a symbol of another FA.
- A box represents repeating subgraphs of a heap.
- FA having boxes in the alphabet are called hierarchical.







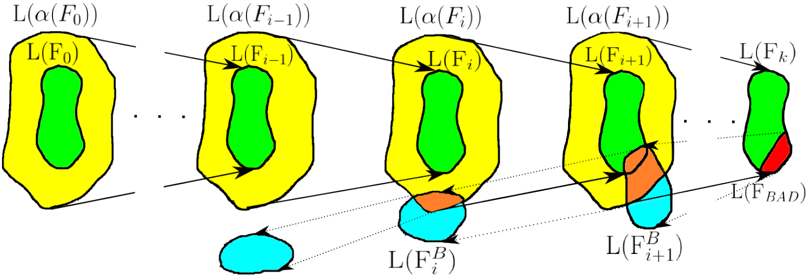
An Overview of Verification Method

- Based on Abstract Regular Tree Model Checking (ARTMC) † and Counterexample-guided Abstraction Refinement (CEGAR). ‡
  - Sets of heap configurations are represented by automata.
  - Employs abstraction over automata to overapproximate the set of reachable configurations (allowing termination on ∞ state spaces).

† *A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular (Tree) Model Checking. STTT, 14(2):167–191, Springer, 2012*

‡ *E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)*

- Based on Abstract Regular Tree Model Checking (ARTMC) † and Counterexample-guided Abstraction Refinement (CEGAR). ‡
  - Sets of heap configurations are represented by automata.
  - Employs abstraction over automata to overapproximate the set of reachable configurations (allowing termination on ∞ state spaces).



† A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Regular (Tree) Model Checking. STTT, 14(2):167–191, Springer, 2012

‡ E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)

Abstraction

- Overapproximates reachable configurations and accelerates analysis
- Collapses states in the same equivalence class of a relation $\sim$
- Height Abstraction
  - Equivalence $\sim_H$ is defined as

    $$q_1 \sim_H q_2 \overset{DEF}{\equiv} L^n(q_1) = L^n(q_2)$$

    where $L^n(q)$ is the language of prefixes of trees accepted from $L(q)$ with height up to n
  - Refinement: By increasing the height
  - Not informed refinement $\Rightarrow$ state explosion

Abstraction

- Overapproximates reachable configurations and accelerates analysis
- Collapses states in the same equivalence class of a relation $\sim$
- Height Abstraction
  - Equivalence $\sim_H$ is defined as

$$q_1 \sim_H q_2 \overset{DEF}{\equiv} L^n(q_1) = L^n(q_2)$$

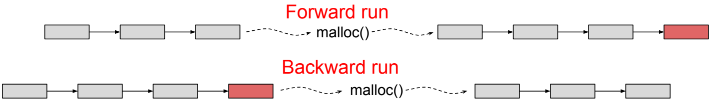
    where $L^n(q)$ is the language of prefixes of trees accepted from $L(q)$ with height up to n
  - Refinement: By increasing the height
  - Not informed refinement $\Rightarrow$ state explosion
- Predicate Language Abstraction
  - Given a set of predicate languages $P = \{p_1, \ldots, p_n\}$, equivalence $\sim_P$ is defined as

$$q_1 \sim_P q_2 \overset{DEF}{\equiv} \forall p \in P : L(q_1) \cap p \neq \emptyset \Leftrightarrow L(q_2) \cap p \neq \emptyset$$

  - Refinement: Interpolating languages from counterexamples
  - Informed refinement

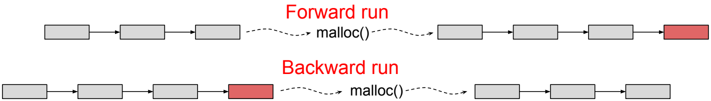
Backward Run and Abstraction for Forest Automata

- Counterexample validation via backward run.
- Ingredients for backward run:
  - Reversion of abstract transformations.



Backward Run and Abstraction for Forest Automata

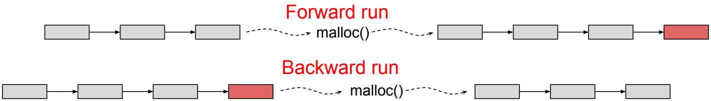
- Counterexample validation via backward run.
- Ingredients for backward run:
  - Reversion of abstract transformations.



  - Reversion of abstraction → intersection of FA.
    - Consider FA $F_1 = (TA_1^1, \ldots, TA_n^1)$ and $F_2 = (TA_1^2, \ldots, TA_n^2)$, intersection is done component-wise using TA intersection, i.e., $F_1 \cap F_2 = (TA_1^1 \cap TA_1^2, \ldots, TA_n^1 \cap TA_n^2)$.
  - Precise reversion of folding and unfolding of boxes ⇒ compatible form.

Backward Run and Abstraction for Forest Automata

- Counterexample validation via backward run.
- Ingredients for backward run:
  - Reversion of abstract transformations.



  - Reversion of abstraction $\rightarrow$ intersection of FA.
    - Consider FA $F_1 = (TA_1^1, \ldots, TA_n^1)$ and $F_2 = (TA_1^2, \ldots, TA_n^2)$, intersection is done component-wise using TA intersection, i.e., $F_1 \cap F_2 = (TA_1^1 \cap TA_1^2, \ldots, TA_n^1 \cap TA_n^2)$.
  - Precise reversion of folding and unfolding of boxes $\Rightarrow$ compatible form.
- Ingredients for predicate language abstraction:
  - Predicate languages represented by TA.
  - New predicate TAs obtained by intersecting FAs from FW and BW run.

Experimental Evaluation

- Shape analysis based on Forest automata implemented in the Forester tool

Program	Status	LoC	Time [s]	Refnm	Preds	Program	Status	LoC	Time [s]	Refnm	Preds
SLL (delete)	safe	33	0.02	0	0	DLL (rev)	safe	39	0.70	0	0
SLL (bubblesort)	safe	42	0.02	0	0	CDLL	safe	32	0.02	0	0
SLL (insertsort)	safe	36	0.04	0	0	DLL (insertsort)	safe	42	0.56	0	0
SLLOfCSLL	safe	47	0.02	0	0	DLLOfCDLL	safe	54	1.76	0	0
SLL01	safe	70	1.20	1	1	**DLL01**	safe	73	0.65	2	2
CircularSLL	safe	49	3.57	3	3	**CircularDLL**	safe	52	37.22	18	24
OptPtrSLL	safe	59	1.90	3	3	**OptPtrDLL**	safe	62	1.87	5	5
QueueSLL	safe	71	11.32	10	10	**QueueDLL**	safe	74	44.68	14	14
GBSLL	safe	64	0.84	3	3	**GBDLL**	safe	71	1.89	4	4
GBSLLSent	safe	68	0.85	3	3	**GBDLLSent**	safe	75	2.19	4	4
RGSLL	safe	72	14.41	22	38	**RGDLL**	safe	76	78.76	26	26
WBSLL	safe	62	0.84	5	5	**WBDLL**	safe	71	1.37	7	7
SortedSLL	safe	76	227.12	15	15	**SortedDLL**	safe	82	36.67	11	11
EndSLL	safe	45	0.07	2	2	**EndDLL**	safe	49	0.10	3	3
TreeRB	error	130	0.08	0	0	**TreeWB**	error	125	0.05	0	0
TreeCnstr	safe	52	0.31	0	0	**TreeCnstr**	error	52	0.03	0	0
TreeOfCSLL	safe	109	0.57	0	0	**TreeOfCSLL**	error	109	0.56	1	3
TreeStack	safe	58	0.20	0	0	**TreeStack**	error	58	0.01	0	0
TreeDSW	safe	72	1.87	0	0	**TreeDSW**	error	72	0.02	0	0
TreeRootPtr	safe	62	1.43	0	0	**TreeRootPtr**	error	62	0.17	2	6
SkipList	safe	84	3.36	0	0	**SkipList**	error	84	0.08	1	1

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018 in the categories related to memory safety (*MemSafety, HeapReach*).

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018 in the categories related to memory safety (*MemSafety, HeapReach*).
- 2015
  - Using the VATA library for tree automata as backend.
  - Counterxample witness generation.

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018 in the categories related to memory safety (*MemSafety, HeapReach*).
- 2015
  - Using the VATA library for tree automata as backend.
  - Counterxample witness generation.
- 2016
  - Added support to run Forester using Benchexec.
  - Counterexample analysis and abstraction refinement for basic forest automata.

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018 in the categories related to memory safety (*MemSafety, HeapReach*).
- 2015
  - Using the VATA library for tree automata as backend.
  - Counterxample witness generation.
- 2016
  - Added support to run Forester using Benchexec.
  - Counterexample analysis and abstraction refinement for basic forest automata.
- 2017
  - Generating correctness witness (forest automaton representing shape invariants of program).
  - Counterexample analysis and abstraction refinement for hierarchical forest automata.
- Forester has never won any medal, but was able to verify difficult data structures (skip-lists, or various trees) which were not solved by any other tool.

Towards Efficient Shape Analysis with Tree Automata

Holík, L., Hruška, M. NETYS'21.

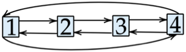
Towards Efficient Shape Analysis with Tree Automata (TA)

- We proposed new automata able to represent graphs with bounded tree-width.
- They are based on tree automata but we work with single TA instead of tuple of TA (like in FA).
- We sketched efficient algorithm for entailment for these automata.

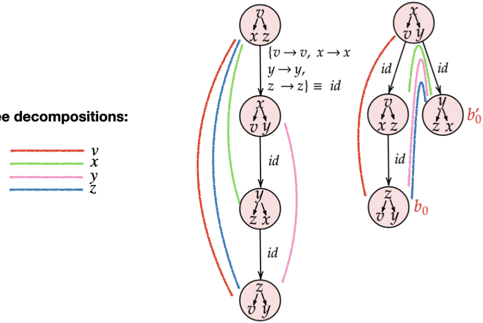
Tree Decomposition with Variables

- How to represent a graph with bounded tree-width by a tree automaton? By tree decompositions with variables.

Graph of circular DLL



Tree decompositions:



Towards Entailment

- A graph may have more decompositions and we want a TA to represent all of them (shape invariant).

Towards Entailment

- A graph may have more decompositions and we want a TA to represent all of them (shape invariant).
- We can create TA representing all tree decompositions that is of a polynomial size assuming a fixed tree-width of the original automaton.

Towards Entailment

- A graph may have more decompositions and we want a TA to represent all of them (shape invariant).
- We can create TA representing all tree decompositions that is of a polynomial size assuming a fixed tree-width of the original automaton.
- Since automata inclusion is EXPTIME-complete, we have entailment which is singly exponential (assuming a fixed maximum tree width).

Shape Analysis based on SMT Solving in 2LS Framework

Malík, V., Hruška, M., Schrammel P., Vojnar T. FMCAD'18.

Shape Analysis based on SMT Solving in 2LS Framework

- Shape analysis using SMT solving to compute the points-to relation between pointers and (abstract) memory addresses.
- Domain designed for representation of linked-lists.
- Therefore more straightforward than the automata-based approaches, but lacks generality of automata based approaches.

Shape Analysis based on SMT Solving in 2LS Framework

- Shape analysis using SMT solving to compute the points-to relation between pointers and (abstract) memory addresses.
- Domain designed for representation of linked-lists.
- Therefore more straightforward than the automata-based approaches, but lacks generality of automata based approaches.
- Implemented within the 2LS framework for program analysis.
  - Combination with other domains in the framework, e.g., the numerical domain.

Shape Analysis based on SMT Solving in 2LS Framework

- Verification procedure takes:
  - A first order formula over combination of SMT theories that represents the program in SSA form (i.e., transition relation).
  - A set of invariants based on predefined templates (proposed for various domains).
  - The property of interest (e.g., memory safety properties).

Shape Analysis based on SMT Solving in 2LS Framework

- Verification procedure takes:
  - A first order formula over combination of SMT theories that represents the program in SSA form (i.e., transition relation).
  - A set of invariants based on predefined templates (proposed for various domains).
  - The property of interest (e.g., memory safety properties).
- And verifies that there is no reachable invariants violate the properties of interest.

Shape Analysis based on SMT Solving in 2LS Framework

- Technically, unsatisfiability of this formula:
  - $\exists \vec{x}, \vec{x}'. \neg(\mathit{Init}(\vec{x}) \Longrightarrow \mathcal{T}(\vec{x}, \vec{d})) \vee \neg(\mathcal{T}(\vec{x}, \vec{d}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \Longrightarrow \mathcal{T}(\vec{x}', \vec{d}))$
  - where $\mathit{Init}(\vec{x})$ is an initial state of variables, $\mathcal{T}$ is a set of invariants based on templates, *Trans* formula representing program

Shape Analysis based on SMT Solving in 2LS Framework

- Technically, unsatisfiability of this formula:
  - $\exists \vec{x}, \vec{x}'. \neg(Init(\vec{x}) \implies \mathcal{T}(\vec{x}, \vec{d})) \vee \neg(\mathcal{T}(\vec{x}, \vec{d}) \wedge Trans(\vec{x}, \vec{x}') \implies \mathcal{T}(\vec{x}', \vec{d}))$
  - where $Init(\vec{x})$ is an initial state of variables, $\mathcal{T}$ is a set of invariants based on templates, $Trans$ formula representing program
- The template for shape analysis is: $\mathcal{T}^S \equiv \bigwedge_{p_i^{lb} \in Ptr^{lb}} = \mathcal{T}^S_{p_i^{lb}}(d_{p_i^{lb}})$,
  - where $\mathcal{T}^S_{p_i^{lb}}(d_{p_i^{lb}}) \equiv (\bigvee_{a \in d_{p_i^{lb}}} p_i^{lb} = a)$.
  - Basically, it describes the points-to relation between a pointer (p_i^{lb}) and addresses a that the pointer may point to.

Experimental Evaluation

Table: Comparison of 2LS with other tools on examples that need reasoning about unbounded data structures and their stored data.

	2LS	**CPA-Seq**	**PredatorHP**	**Forester**	**Symbiotic**	**UAutomizer**
Calendar	2.88	timeout	false	unknown	timeout	timeout
Cart	23.70	timeout	false	unknown	timeout	timeout
Hash Function	3.65	8.51	unknown	unknown	unknown	timeout
MinMax	5.14	timeout	false	unknown	timeout	timeout
Packet Filter	431.00	timeout	timeout	unknown	unknown	timeout
Process Queue	6.62	7.68	timeout	unknown	timeout	timeout
Quick Sort	18.20	3.50	timeout	unknown	unknown	5.75
Running Example	1.24	timeout	timeout	unknown	timeout	unknown
SM1	0.53	timeout	0.31	false	timeout	timeout
SM2	0.55	5.41	false	false	timeout	14.50

Automated Software Testing (with Automata)

Generating Scenarios for Digital Twins of Distributed Manufacturing Execution Systems.

Fiedor, T., Hruška, M., Smrčka, A. EUROCAST'22.

Introduction

- Manufacturing Execution System (MES)
  - Software managing production in factory.
  - This includes communication with information system, machines, etc.
  - Difficult to test:
    - Distributed nature of manufacturing, different communication protocols, different formats of structured data.

Introduction

- Manufacturing Execution System (MES)
  - Software managing production in factory.
  - This includes communication with information system, machines, etc.
  - Difficult to test:
    - Distributed nature of manufacturing, different communication protocols, different formats of structured data.
- Digital twin
  - Generally, digital copy of a cyber-physical system — simulation of reality on computer, often with graphical interface.
  - Not everything needs to be simulated, some software may be used in digital twin natively.

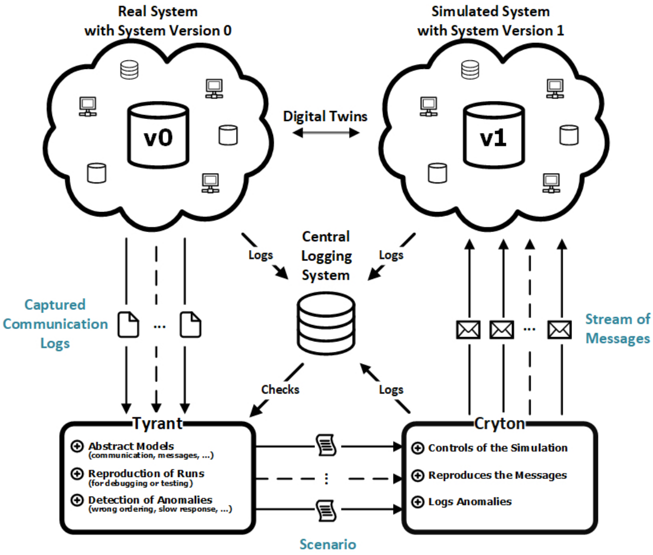
Introduction

- Manufacturing Execution System (MES)
  - Software managing production in factory.
  - This includes communication with information system, machines, etc.
  - Difficult to test:
    - Distributed nature of manufacturing, different communication protocols, different formats of structured data.
- Digital twin
  - Generally, digital copy of a cyber-physical system — simulation of reality on computer, often with graphical interface.
  - Not everything needs to be simulated, some software may be used in digital twin natively.

Introduction

- Manufacturing Execution System (MES)
  - Software managing production in factory.
  - This includes communication with information system, machines, etc.
  - Difficult to test:
    - Distributed nature of manufacturing, different communication protocols, different formats of structured data.
- Digital twin
  - Generally, digital copy of a cyber-physical system — simulation of reality on computer, often with graphical interface.
  - Not everything needs to be simulated, some software may be used in digital twin natively.
- Project solved with the Unis company (MES Pharis) and Masaryk University (Cryton for orchestrating digital twin)

Our Solution



Our Solution

- Representing messages in different formats (XML, or JSON) using Trie structure (prefix tree)

Our Solution

- Representing messages in different formats (XML, or JSON) using Trie structure (prefix tree)
- Representing communication using event calendar.
  - Direct generation of scenario for digital twin.
  - Can be transformed to finite automaton — application of abstraction to overapproximate langauge.
  - New strings in overapproximated langauge $\Rightarrow$ new series of events $\Rightarrow$ new scenario $\Rightarrow$ new test case.

Our Solution

- Representing messages in different formats (XML, or JSON) using Trie structure (prefix tree)
- Representing communication using event calendar.
  - Direct generation of scenario for digital twin.
  - Can be transformed to finite automaton — application of abstraction to overapproximate langauge.
  - New strings in overapproximated langauge $\Rightarrow$ new series of events $\Rightarrow$ new scenario $\Rightarrow$ new test case.
- Implemented in the Tyrant tool.
  - Generated valid scenarios for orchestration of digital twin with deployed Pharis.
  - More engineering work needed to be deployed in production.

Conclusion

Conclusion

- Work presented in the thesis:
  - Counterexample analysis and abstraction refinement for Forest automata.
  - Forester participation in SV-COMP, editions 2015-2018.
  - A chapter on forest automata in book on software verification with refined presentation of the approach.
  - Automata over graphs with bounded tree-width.
  - Shape analysis based on SMT solving.
  - Automated testing of distributed manufacturing execution systems.

Conclusion

- Work presented in the thesis:
  - Counterexample analysis and abstraction refinement for Forest automata.
  - Forester participation in SV-COMP, editions 2015-2018.
  - A chapter on forest automata in book on software verification with refined presentation of the approach.
  - Automata over graphs with bounded tree-width.
  - Shape analysis based on SMT solving.
  - Automated testing of distributed manufacturing execution systems.
- Work not presented (or only mildly mentioned) in the thesis:
  - Connection of Predator and Symbiotic participating in SV-COMP'20.
  - The design and implementation of efficient automata library called MATA (TACAS'24).

Publications

- Chocholatý D., Fiedor T., Havlena V., Holík L., Hruška M., Lengál, O., Síč J.: *Mata: A Fast and Simple Finite Automata Library.* In Proc. of TACAS'24.
- Holík L., Fiedor T., Rogalewicz A., Vargovčík P., Hruška M., Síč J.,: *Reasoning about Regular Properties: A Comparative Study.* In Proc. of CADE'23.
- Habermehl P., Holík L., Hruška M., Lengál O., Rogalewicz A., Šimáček J., Vojnar T. *Forester: Tree Automata in Shape Analysis.* Accepted for publication in the book ASV on the State-of-the-art Tools for Software Verification.
- Fiedor T., Hruška M., Smrčka A. *Orchestrating Digital Twins for Distributed Manufacturing Execution Systems.* In Proc. of EUROCAST'22.
- Holík L., Hruška M.. *Towards Efficient Shape Analysis with Tree Automata.* In Proc. of NETYS'21.
- Ayaziová P., Hruška M., Chalupa M., Jašek M., Strejček M., Šoková V., Tomovič L. *Symbiotic 7: Integration of Predator and More (Competition Contribution).* In Proc. of TACAS'20
- Malík, V., Hruška, M., Schrammel, P., Vojnar, T.: *Template-Based Verification of Heap-Manipulating Programs.* In *Proc. of FMCAD'18*, 2018.
- Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Vojnar, T.: *Counterexample Validation and Interpolation-Based Refinement for Forest Automata.* In *Proc. of VMCAI'17*, LNCS 10145, Springer, 2017.
- Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Šimáček, J., Vojnar, T.: *Forester: From Heap Shapes to Automata Predicates (Competition Contribution).* To Apper *Proc. of TACAS'17*, Springer, 2017.
- Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Šimáček, J., Vojnar, T.: *Run Forester, Run Backwards! (Competition Contribution).* In *Proc. of TACAS'16*, LNCS 9636, Springer, 2016.
- Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Šimáček, J., Vojnar, T.: *Forester: Shape Analysis Using Tree Automata (Competition Contribution).* In *Proc. of TACAS'15*, LNCS 9035, Springer, 2015.

Software Products

- Chocholatý D., Fiedor T., Havlena V., Holík L., Hruška M., Lengál, O., Síč J.: *Mata: A Finite Automata Library.* 2024.
- Fiedor T., Hruška M., Smrčka A., Švéda M., Hradský T. *Analyser of Metrics Measured in Monitoring Center.* 2022.
- Fiedor T., Hruška M., Smrčka A., *Software for measurement and evaluation of performance parameters.* 2021.
- Fiedor T., Hruška M., Smrčka A., Panov S., Rozsival M., Tureček D., Čeleda P., Pospíšil L. *Tyrant: The Ruler of Digital Twins.* 2021.

The Questions of Reviewers

- Is a user-non-friendlines of library for finite automata a problem if it used as backend of verification tool?
  - Context: In conclusion of my thesis I proposed that one of the future directions is to design a new efficient, simple, and user-friendly automata library.
  - Depends on definition of user-friendliness.
  - In general, if you work in small team with limited amount of resources you want library that has a clear interface and predicatable behaviour, which is a kind of user friendliness.

The Questions of Reviewers

- When generating abstract messages, could the abstraction lead to generation of messages that really do not make sense in practice? Is it still worth to examine the behaviour of the system after transferring such mesages?
  - Context: In work on testing MES we proposed methods for generating new test cases using abstraction over messages and communication models.
  - Yes, it can lead to messages or seqeunces of messages that does not make sense but the system under testing should fail when these messages.
  - From practical point of view, it is necessary to tune the abstractions to avoid generating such things.