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 (most of the work).
  - Automated software testing (introduced later).
- Shape analysis — formal verification of programs with dynamic data structures.
- Goal of shape analysis: Derive reachable shapes of dynamic data structures.
  - 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,
  - So called shape invariant.
- Applied in Abstract Regular Tree Model Checking (ARTMC).

Advantages

- Existing efficient algorithms for handling finite automata.
- Flexibility to represent various kinds of data structures.

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,
  - So called shape invariant.
- Applied in Abstract Regular Tree Model Checking (ARTMC).

Advantages

- Existing efficient algorithms for handling finite automata.
- Flexibility to 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. Holik, J. Simacek, 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. Holik, J. Simacek, 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. Holik, J. Simacek, A. Rogalewicz, and T. Vojnar. Forest Automata for Verification of Heap Manipulation. FMSD, 41(1):83–106, Springer, 2012.

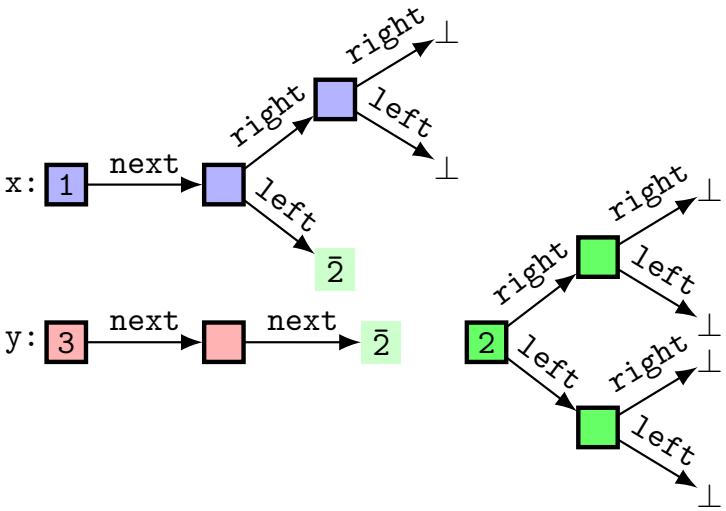
Forest Automata Encoding of Heap

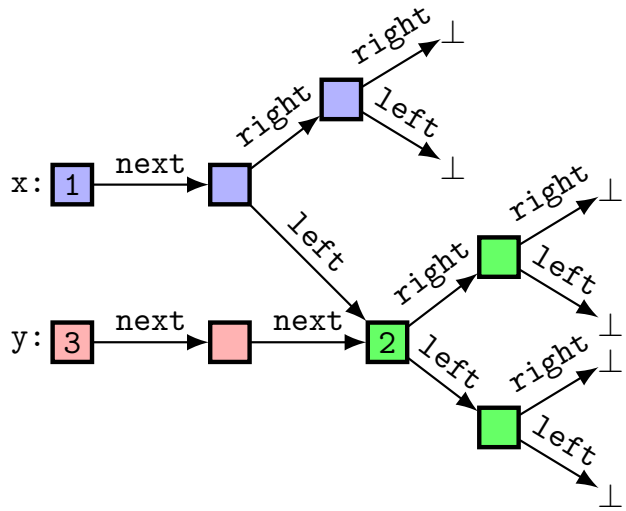
- A Forest Automaton (FA)* is a tuple of tree automata (TA).
- A TA accepts trees.
- Trees can contain leaves referencing the roots of other trees.
- An FA $F = (TA_1, \ldots, TA_n)$ represents tree decompositions (tuples of trees $t_1, \ldots, t_n$) of heap graphs such that $\forall 1 \leq i \leq n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

*P. Habermehl, L. Holik, J. Simacek, 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.

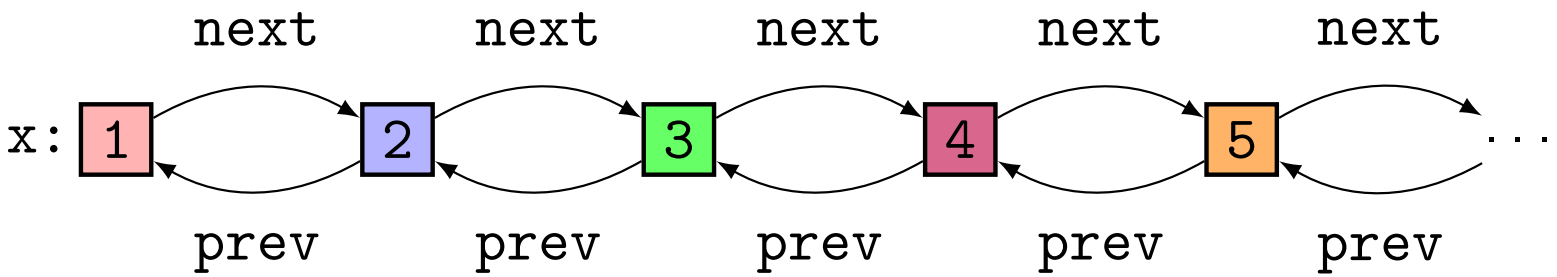


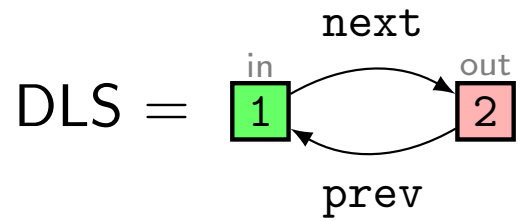


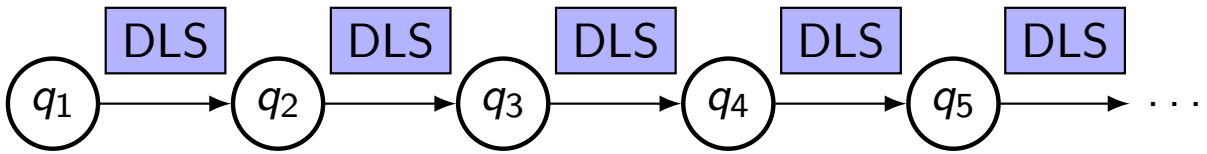
P. Habermehl, L. Holik, J. Simacek, A. Rogalewicz, and T. Vojnar. Forest Automata for Verification of Heap Manipulation. FMSD, 41(1):83–106, Springer, 2012.

Boxes

- We use boxes 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.

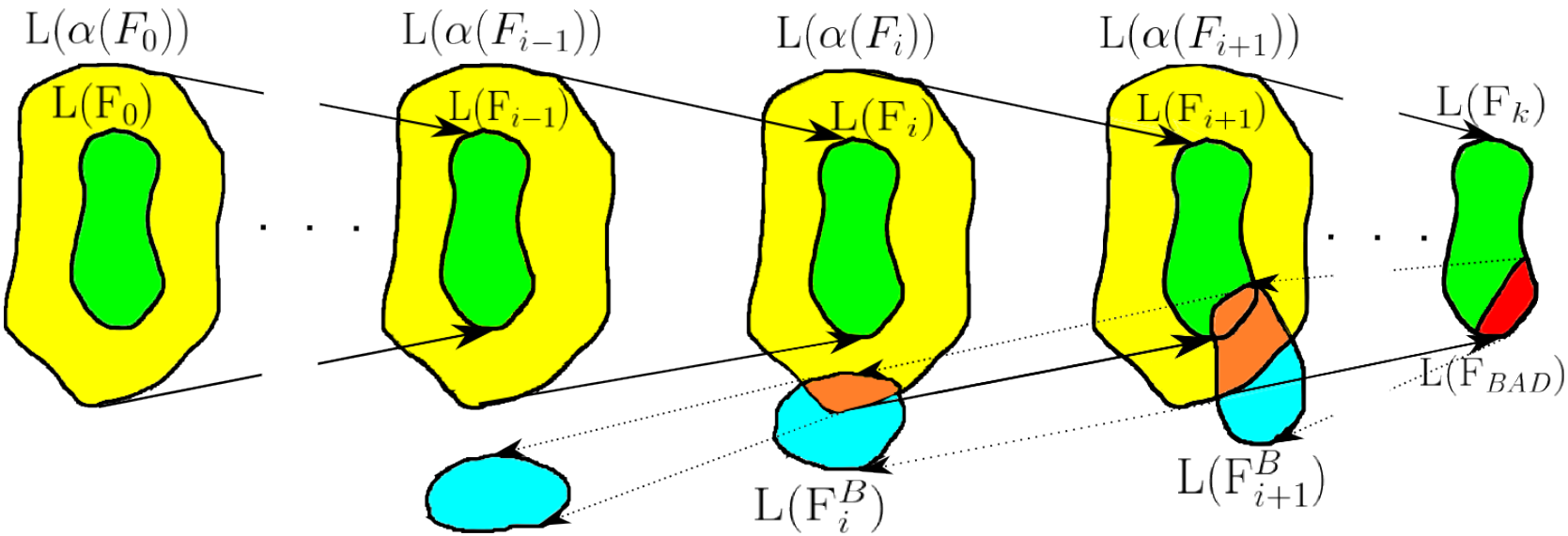






An Overview of Verification Method

- 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)*

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

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

    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

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

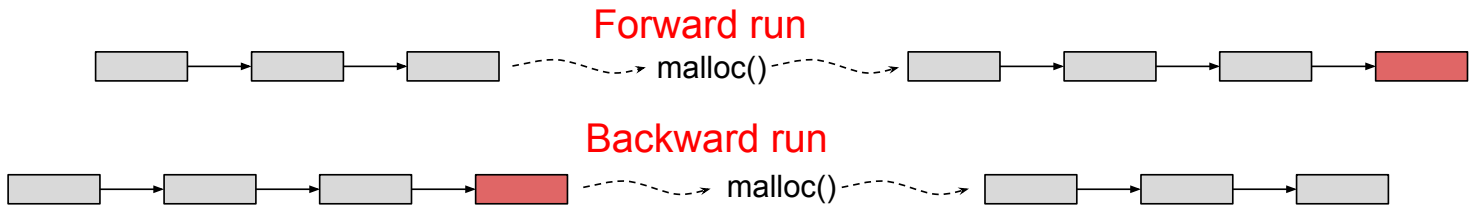
    where $L^n(q)$ is the language of prefixes of trees accepted from $L(q)$ with height up to n
  - Refinement: By increasing the height
  - Not informed refinement $\Rightarrow$ state explosion
- Predicate Language Abstraction
  - Given a set of predicate languages $P = \{p_1, \ldots, p_n\}$, equivalence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

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

  - Refinement: Interpolating languages from counterexamples
  - Informed refinement

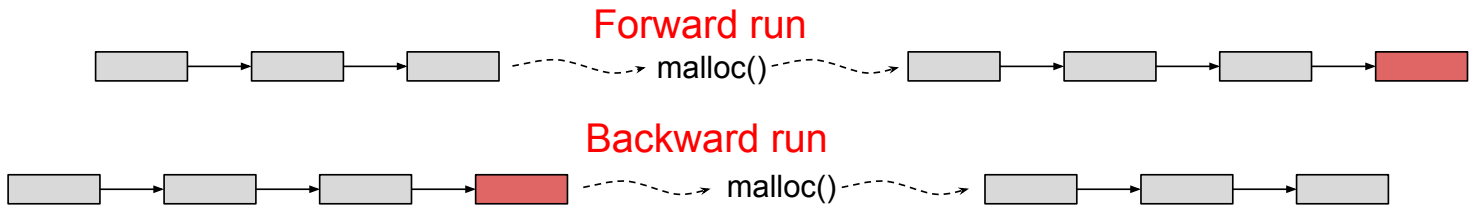
Backward Run and Abstraction for Forest Automata

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



Backward Run and Abstraction for Forest Automata

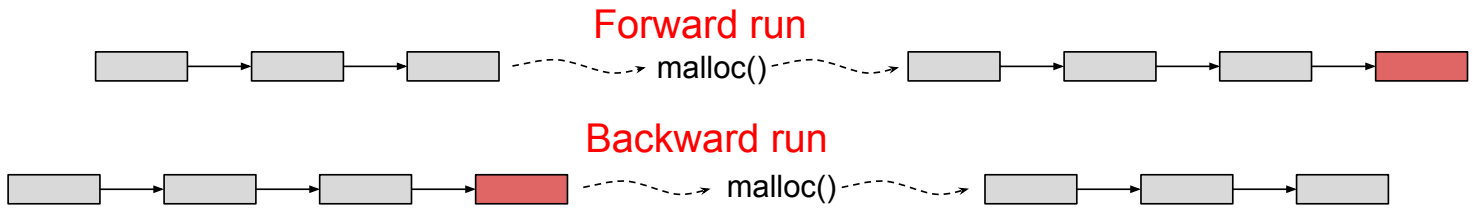
- Counterexample validation via backward run.
- Ingredients for backward run:
  - Reversion of abstract transformations.



  - Reversion of abstraction $\rightarrow$ 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.

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

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.

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018.
- 2015
  - Using VATA as backend.
  - Counterxample witness generation.

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018.
- 2015
  - Using VATA as backend.
  - Counterxample witness generation.
- 2016
  - Added support to run Forester using Benchexec.
  - Counterexample analysis and abstraction analysis for basic forest automata.

Forester in SV-COMP

- Forester competed in SV-COMP in years 2015-2018.
- 2015
  - Using VATA as backend.
  - Counterxample witness generation.
- 2016
  - Added support to run Forester using Benchexec.
  - Counterexample analysis and abstraction analysis for basic forest automata.
- 2017
  - Generating correctness witness. In our case, printing forest automaton representing shape invariant for each line.
  - Counterexample analysis and abstraction analysis for hierarchical forest automata.

Towards Efficient Shape Analysis with Tree Automata

Holík, L., Hruška, M., Towards Efficient Shape Analysis with Tree Automata. NETYS'21.

Towards Efficient Shape Analysis with Tree Automata

- Forest Automata have their limits:
  - Can't represent data structures such as grid.
  - Are not closed under union (which complicates their manipulation during verification procedure, e.g., a fixpoint computing).

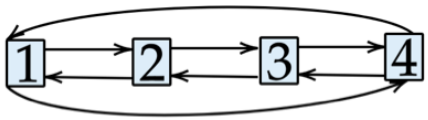
Towards Efficient Shape Analysis with Tree Automata

- Forest Automata have their limits:
  - Can't represent data structures such as grid.
  - Are not closed under union (which complicates their manipulation during verification procedure, e.g., a fixpoint computing).
- We proposed a new automata capable of representing graphs with bound tree-width.
  - Particularly, their langauges consist of tree decompositions of graphs with bounded tree width.

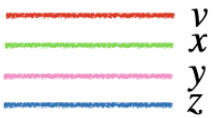
Towards Efficient Shape Analysis with Tree Automata

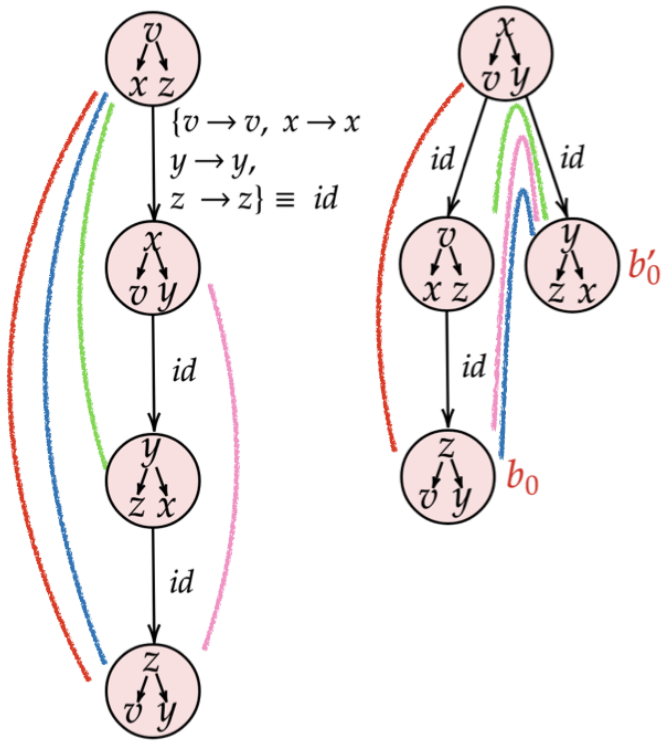
- A *tree decomposition* of a labeled graph g over a finite set of variables $\mathbb{X}$ and alphabet Σ is a tree $d = (B, E)$. Nodes B of t are Σ-labeled graphs called *bags*. Nodes of a bag are variables from $\mathbb{X}$. Edges of d are labelled by partial mappings $\rho : \mathbb{X} \to \mathbb{X}$ called *parameter assignments*.

Graph of circular DLL



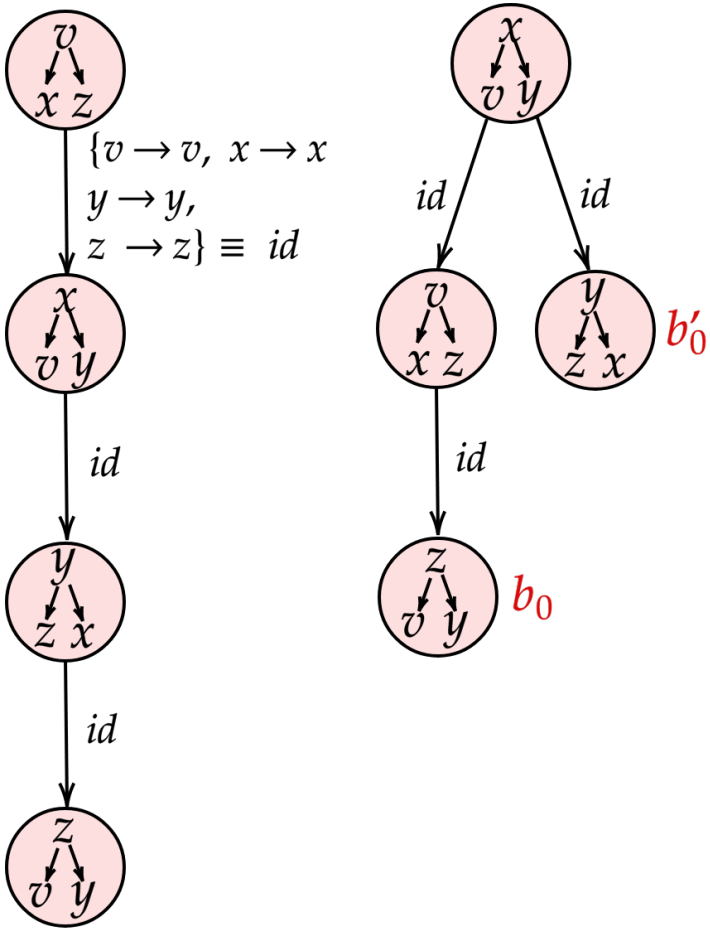
Tree decompositions:

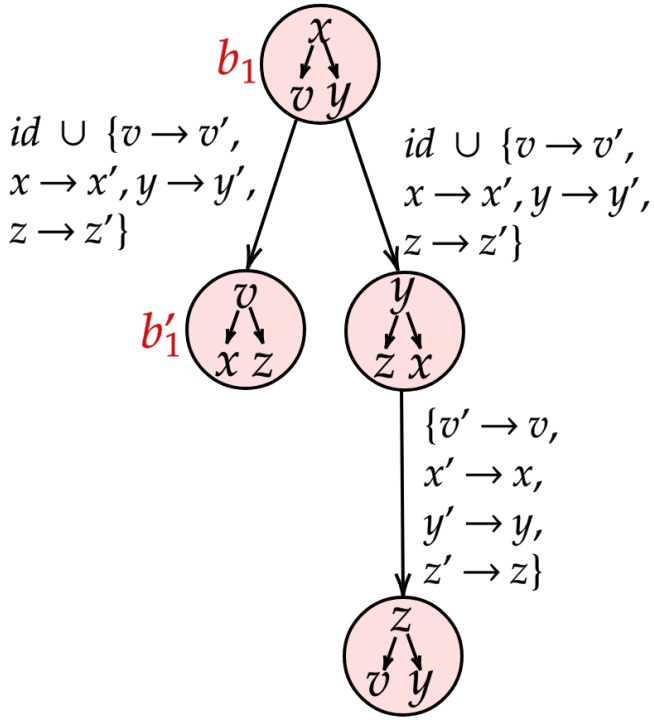




Automata over Graphs with Bounded Tree-width

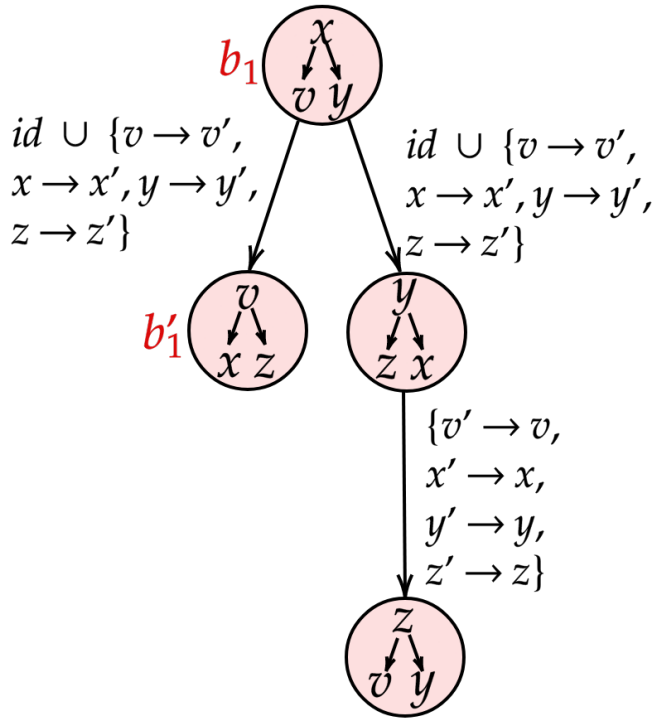
- Reconnection

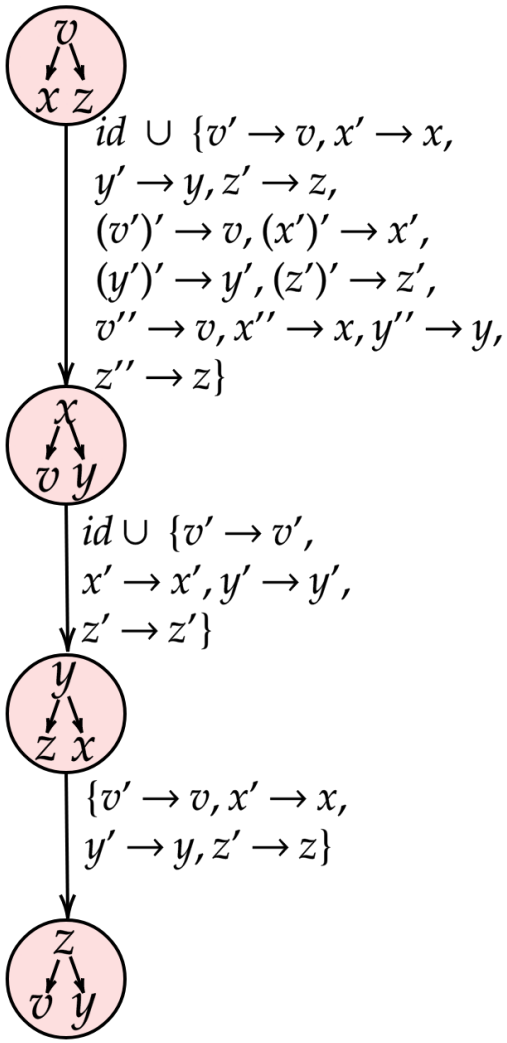




Automata over Graphs with Bounded Tree-width

- Rotation





Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).

Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).
- One phase at most doubles the number of variables.

Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).
- One phase at most doubles the number of variables.
- An equivalent decomposition t can be obtained from t' in a number of phases that depends only on $max(tw(t), tw(t))$ where tw is a tree width of decomposition.

Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).
- One phase at most doubles the number of variables.
- An equivalent decomposition t can be obtained from t' in a number of phases that depends only on $max(tw(t), tw(t))$ where tw is a tree width of decomposition.

- Tree decompositions can be represented as language of tree automaton.

Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).
- One phase at most doubles the number of variables.
- An equivalent decomposition t can be obtained from t' in a number of phases that depends only on $max(tw(t), tw(t))$ where tw is a tree width of decomposition.

- Tree decompositions can be represented as language of tree automaton.
- The implementation of a tree automata phase at most doubles the number of variables and leads to an automaton that is of a polynomial size assuming a fixed tree-width of the original automaton.

Towards entailment

- Phase is a set of all operations that could be performed at once (they don't interfere with each other).
- One phase at most doubles the number of variables.
- An equivalent decomposition t can be obtained from t' in a number of phases that depends only on $max(tw(t), tw(t))$ where tw is a tree width of decomposition.

- Tree decompositions can be represented as language of tree automaton.
- The implementation of a tree automata phase at most doubles the number of variables and leads to an automaton 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., Template-Based Verification of Heap-Manipulating Programs. 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 memory dynamically allocated (represented by abstract objects).
- Domain designed for representation of linked-lists (therefore more straightforward than the automata-based approaches).
- Implemented within 2LS framework for program analysis.
  - Combination with other domains in this Framework, e.g., the numerical domain.
- Lacks generality of automata based approaches.

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.
  - A set of invariants based on predefined templates (proposed for various domains of data).
  - The property of interest.

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.
  - A set of invariants based on predefined templates (proposed for various domains of data).
  - The property of interest.
- We verify that there is no reachable counterexample violating the invariants.
- Technically, i.e., unsatisfaibility of this formula:
  - $\exists \vec{x}, \vec{x}'. \neg(\mathit{Init}(\vec{x}) \implies \mathcal{T}(\vec{x}, \vec{d})) \vee \neg(\mathcal{T}(\vec{x}, \vec{d}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \implies \mathcal{T}(\vec{x}', \vec{d}))$
  - where $\mathit{Init}(\vec{x})$ is an initial state of variables, $\mathcal{T}$ represents program properities of interest and negation of their invariants, *fmtrans* formula representing program

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.
  - A set of invariants based on predefined templates (proposed for various domains of data).
  - The property of interest.
- We verify that there is no reachable counterexample violating the invariants.
- Technically, i.e., unsatisfaibility of this formula:
  - $\exists \vec{x}, \vec{x}'.\ \neg(\mathit{Init}(\vec{x}) \implies \mathcal{T}(\vec{x}, \vec{d})) \vee \neg(\mathcal{T}(\vec{x}, \vec{d}) \wedge \mathit{Trans}(\vec{x}, \vec{x}') \implies \mathcal{T}(\vec{x}', \vec{d}))$
  - where $\mathit{Init}(\vec{x})$ is an initial state of variables, $\mathcal{T}$ represents program properities of interest and negation of their invariants, *fmtrans* 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 (abstract value $d_{p_i^{lb}}$) that the pointer may point to.

Shape Analysis based on SMT Solving in 2LS Framework

Table: Comparison of 2LS with other tools on examples combining 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 Sotware Testing (with Automata)

Fiedor, T., Hruška, M., Smrčka, A., Generating Scenarios for Digital Twins of Distributed Manufacturing Execution Systems. EUROCAST'22.

Introduction

- Manufacturing Execution System (MES)
  - Software managing production in manufactory.
  - MES tasks are:
    - Communication with Enterprise Resource Planning (ERP) systems such as SAP, e.g., checking quantity of material in stocks.
    - Scheduling production over the machines.
    - Controlling and communicating with machines during production.
    - Processes inputs by human workers operating machines.

Introduction

- Manufacturing Execution System (MES)
  - Software managing production in manufactory.
  - MES tasks are:
    - Communication with Enterprise Resource Planning (ERP) systems such as SAP, e.g., checking quantity of material in stocks.
    - Scheduling production over the machines.
    - Controlling and communicating with machines during production.
    - Processes inputs by human workers operating machines.
  - Testing of MES is difficult because:
    - Distributed nature of manufacturing — different components (ERP, MES, Machines) run it own software and communicate asynchronously.
    - Different communication protocols between different components.
    - Different formats structured data in messages.
    - Nondeterministic factors — human workers and their interactions with machines, physical aspects of machines.

Introduction — Digital twin

- 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.

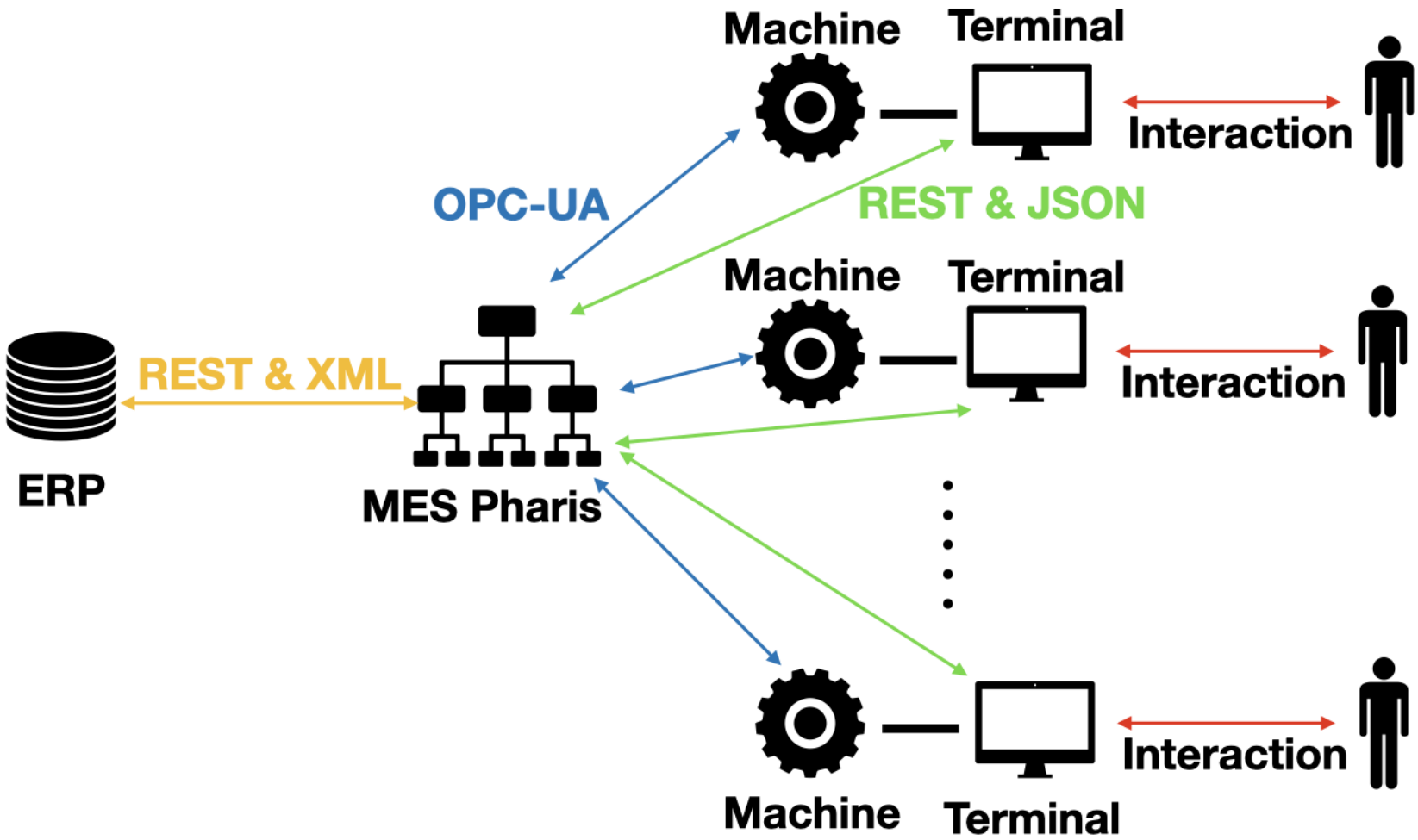
Introduction — Digital twin

- 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.
  - In our case, digital copy of a manufactory simulated in command line.
  - Particularly, machines, ERP, or human workers are simulated, MES is executed natively.
  - Can be orchestrated (deployed and controlled) by an input scenario (step by step).

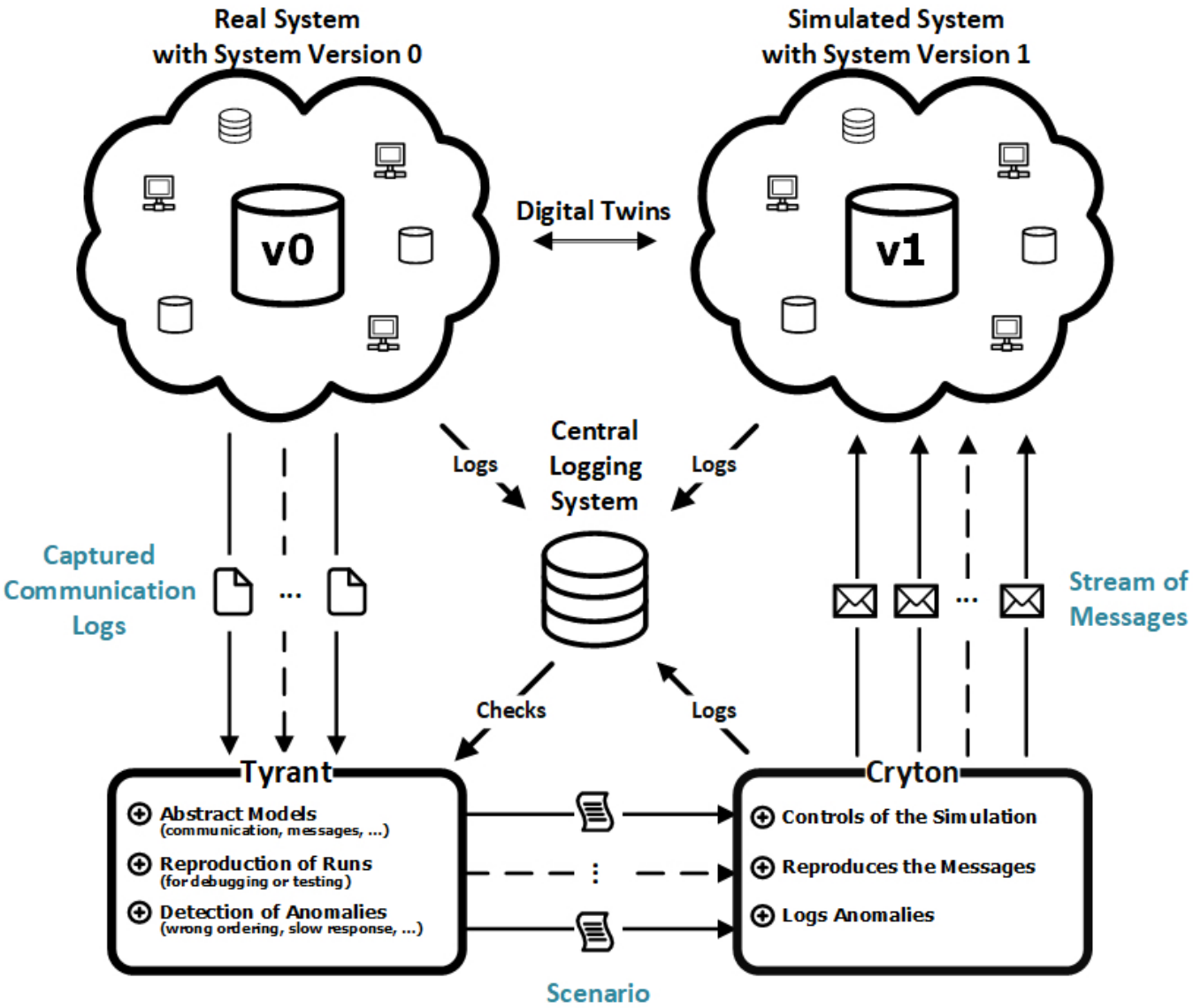
Context of project

- In collaboration with Masaryk University and UNIS, a.s. company
- The goal of project was to develop digital twin of systems controlled by MES Pharis developed by UNIS company
- The presented talk is about learning models of communication and generating scenarios for orchestrating digital twin
- Digital twin was implemented by Masaryk University

Real system


Machine
Terminal
Interaction
OPC-UA
REST & JSON
Machine
Terminal
Interaction
REST & XML
ERP
MES Pharis
Interaction
Machine
Terminal

Our solution



Our solution

- Representing messages with structured data
  - Different formats of messages — JSON and XML
  - Generic representation and unified manipulation using Trie structure (prefix tree)
  - We classify messages based on structural similarity to different classes and create an abstract representation of each class.
  - Can automatically generate new messages from abstract representation.

Our solution

- Representing messages with structured data
  - Different formats of messages — JSON and XML
  - Generic representation and unified manipulation using Trie structure (prefix tree)
  - We classify messages based on structural similarity to different classes and create an abstract representation of each class.
  - Can automatically generate new messages from abstract representation.
- Representing model of communication:
  - Representing communication in system using event calendar.
  - There is created an event in calendar for each message (containing sender, receiver, type of message, etc.).
  - Later serialized to a YAML scenario used for orchestrating digital twin.

Our solution

- Representing messages with structured data
  - Different formats of messages — JSON and XML
  - Generic representation and unified manipulation using Trie structure (prefix tree)
  - We classify messages based on structural similarity to different classes and create an abstract representation of each class.
  - Can automatically generate new messages from abstract representation.
- Representing model of communication:
  - Representing communication in system using event calendar.
  - There is created an event in calendar for each message (containing sender, receiver, type of message, etc.).
  - Later serialized to a YAML scenario used for orchestrating digital twin.
- Implemented in the Tyrant tool.

Automata abstraction to generate new tests cases

- Function for transforming event calendar to finite automaton — set of events is used as alphabet of automaton
- Runs of modeled system (which are basically series of events) are strings in language of automaton

Automata abstraction to generate new tests cases

- Function for transforming event calendar to finite automaton — set of events is used as alphabet of automaton
- Runs of modeled system (which are basically series of events) are strings in language of automaton

- Length abstraction of automaton — merges states of automaton with the same prefix language w.r.t. a given length
- Abstraction overapproximates language of automaton $\rightarrow$ language of abstracted automaton may contain strings which are not in the original one
- Such new strings could be converted back to series of events $\rightarrow$ new scenario $\rightarrow$ new test case

Evaluation

- We test and run our tools in UNIS company.
- We automatically generated a valid scenario for Cryton (so far, the scenarios were created manually by a human tester only).
- We made experiments with automatic new scenarios generation:
  - Abstraction only in certain types of communication (Primarly, between MES and machines).
  - Communication between terminals and MES is not suitable for abstraction (need to preserve sequences of messages).

Evaluation

- We test and run our tools in UNIS company.
- We automatically generated a valid scenario for Cryton (so far, the scenarios were created manually by a human tester only).
- We made experiments with automatic new scenarios generation:
  - Abstraction only in certain types of communication (Primarly, between MES and machines).
  - Communication between terminals and MES is not suitable for abstraction (need to preserve sequences of messages).
- Future Work
  - Deploy our tools in industry settings of UNIS company.
  - Tune automata abstraction and new messages generation for Pharis.

Conclusion

Conclusion

- Work presented in the thesis:
  - Counterexample analysis and abstraction refinement for Forest automata.
  - Competing in SV-COMP, edition 2015-2018.
  - 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.
  - Competing in SV-COMP, edition 2015-2018.
  - 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 (MATA'24).

Publications

- Chocholaty D., Fiedor T., Havlena V., Holik L., Hruska M., Lengal, O., Sic J.: *Mata: A Fast and Simple Finite Automata Library.* In Proc. of TACAS'24.
- L Holik, T. Fiedor, A. Rogalewicz, P. Vargovcik, M. Hruska, J. Sic.: *Reasoning about Regular Properties: A Comparative Study.* In Proc. of CADE'23.
- P. Habermehl, L. Holik, M. Hruska, O. Lengal, A. Rogalewicz, J. Simacek, T. Vojnar. *Forester: Tree Automata in Shape Analysis*. Accepted for publication in the book ASV on the State-of-the-art Tools for Software Verification.
- T. Fiedor, M. Hruska, A. Smrcka. *Orchestrating Digital Twins for Distributed Manufacturing Execution Systems.* In Proc. of EUROCAST'22.
- L. Holik, M. Hruska. *Towards Efficient Shape Analysis with Tree Automata.* In Proc. of NETYS'21.
- P. Ayaziova, M. Hruska, M. Chalupa, M. Jasek, J. Strejcek, V. Sokova, L. Tomovic. *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.
- Participation at SV-COMP'19 with the 2LS tool (in collaboration with Viktor Malík)
- 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

- Chocholaty D., Fiedor T., Havlena V., Holik L., Hruska M., Lengal, O., Sic J.: *Mata: A Finite Automata Library.* 2024.
- T. Fiedor, M. Hruska, A. Smrcka, M. Sveda, T. Hradsky. *Analyser of Metrics Measured in Monitoring Center.* 2022.
- T. Fiedor, M. Hruska, A. Smrcka. *Software for measurement and evaluation of performance parameters.* 2021.
- T. Fiedor, M. Hruska, A. Smrcka, S. Panov, M. Rozsival, D. Turecek, P. Celeda, L. Pospisil. *Tyrant: The Ruler of Digital Twins.* 2021.

The Definition of Graph Tree Width

- A tree decompostion of a graph $G = (V_G, E_G)$ is a tree $T = (V_T, E_T)$, such that:
  - $V_T \subseteq 2^{V_G}$
  - $\forall v_G \in V_G \, \exists v_T \in V_T : v_G \in v_T$
  - $\forall (u, v) \in E_G \, \exists v_T \in V_T : u, v \in v_T$
  - $\forall v_G \in V_G$: The set $\{v_t \in V_T \mid v_G \in v_T\}$ induces connected subtree
- A width of the given decomposition is the maximal cardinality of the nodes of the decomposition minus one.
- A tree width of the given graph is a minimal width of all possible tree decompositions of the given graph.