O pewnym logicznym podejściu do analizy wydobytych sieci aktywności

Radosław Klimek

John William WATERHOUSE: *Odyseusz i syreny*

Table of Contents

Table of Contents

Abstract

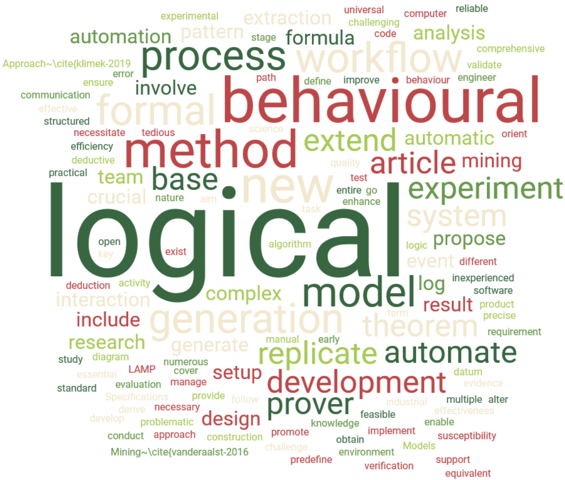

experimental
extraction
universal
computer
reliable
challenging
automation
pattern
stage
formula
code
analysis
process
workflow
comprehensive
Approach~\cite{klimek-2019
error
validate
ensure
path
define
improve
behaviour
engineer
involve
communication
formal
behavioural
effective
structured
science
necessitate
tedious
orient
efficiency
method
extend
automatic
deductive
quality
article
mining
practical
team
base
test
crucial
nature
aim
new
entire
go
experiment
enhance
task
system
logical
open
key
exist
algorithm
logic
propose
event
different
deduction
activity
inexperienced
interaction
complex
model
log
software
result
product
term
include
generation
theorem
precise
requirement
research
generate
manual
early
study
diagram
replicate
automate
numerous
LAMP
cover
setup
datum
essential
evaluation
manage
development
evidence
standard
Specifications
provide
follow
industrial
multiple
alter
derive
necessary
prover
effectiveness
develop
problematic
design
feasible
knowledge
enable
conduct
approach
construction
promote
implement
susceptibility
obtain
challenge
environment
Models
Mining~\cite{vanderaalst-2016
predefine
support
verification
equivalent

Table of Contents

Background

There is no better way to **bore** the audience than to talk about **logic**!

"Logic has simple, unambiguous syntax and semantics. It is thus ideally suited to the task of specifying information systems" – **Chomicki & Saake**.

"Logic is the glue that binds together methods of reasoning, in all domains" – **Galton**.

"We need a style of logic that can be used as a tool in every-day work" – **Gries & Schneider**.

Logic and algebra

According to Manfred Broy*, two dominant approaches in the field of software engineering are:

(1) the logical approach, (2) the algebraic approach.

* Manfred Broy, "On the Role of Logic and Algebra in Software Engineering". In: "Mathematics, Computer Science and Logic – A Never Ending Story" 2013, pp. 51–68, Springer.

Logic and algebra

According to Manfred Broy*, two dominant approaches in the field of software engineering are:

(1) the logical approach, (2) the algebraic approach.

A few interesting quotes from the paper:

"...stakeholders tend to be contradictory and inconsistent" "**Logic can help to define what consistency means, to analyse inconsistencies and give hints how they may be resolved**" "**each component behaviour is captured by a logical formula**" "Software development is an art and a craft; it proceeds by esoteric lore, by stepwise improvement, by trial and error – software is an artifact, immeasurable, unreliable, and unpredictable" "**Logic provides a unifying frame!**"

* Manfred Broy, "On the Role of Logic and Algebra in Software Engineering". In: "Mathematics, Computer Science and Logic – A Never Ending Story" 2013, pp. 51–68, Springer.

Formal Methods – Two Approaches

Two approaches* to the system specification and verification:

1. model checking – state exploration and reachability,
2. logical inference – a highly developed approach.

Meanwhile, formal logical reasoning, both deductive and abductive, is closely aligned with the natural human approach to understanding inference. It is also applied in our everyday activities.

* Clarke, Wing et al.: Formal methods: State of the art and future directions. In: *ACM Computing Surveys*, 1996, vol. 28, no. 4, pp. 626-643.

Model checking

Definition

Model checking MC is a decision problem whether a Kripke structure K and temporal formula P is valid in K, i.e. $K \models P$?

- Algorithmic verification whether a program satisfies a logical specification;
- efficient algorithms, significant progress in the past, and so on;
- however, it is a form of simulation, being more operational than analytical.

Turing Award (2007) for Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis for "developing model checking into a highly effective verification technology, widely adopted in the hardware and software industries".

Model checking

Definition

Model checking MC is a decision problem whether a Kripke structure K and temporal formula P is valid in K, i.e. $K \models P$?

- Algorithmic verification whether a program satisfies a logical specification;
- efficient algorithms, significant progress in the past, and so on;
- however, it is a form of simulation, being more operational than analytical.

Turing Award (2007) for Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis for "developing model checking into a highly effective verification technology, widely adopted in the hardware and software industries".

No, it's not like that!

Deductive reasoning

Definition

Logical reasoning is a kind of valid reasoning when set of statements p_i preserve true, i.e.
$p_1 \wedge \ldots \wedge p_n \models P$?

- The process of moving from general statements, based on what is known, to logically certain conclusions;
- from true premises to true conclusions;
- **logical reasoning is "symbolic" and can address infinite computations!**
- the deductive approach is currently undergoing rapid development.

Deductive reasoning

Definition

Logical reasoning is a kind of valid reasoning when set of statements p_i preserve true, i.e.
$p_1 \wedge \ldots \wedge p_n \models P$?

- The process of moving from general statements, based on what is known, to logically certain conclusions;
- from true premises to true conclusions;
- **logical reasoning is "symbolic" and can address infinite computations!**
- the deductive approach is currently undergoing rapid development.

Yes, something like that!

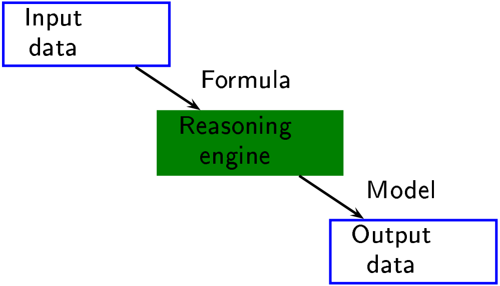
Deductive approach with theorem provers

Prominent theorem provers/SAT solvers:

- Prover Vampire is an advanced theorem prover used for verifying statements in first-order logic and formal program verification.
- Prover E is a theorem prover that utilizes connection calculus, term indexing, and built-in strategies to handle first-order logic.
- Prover9 is a theorem prover that employs resolution and paramodulation to derive proofs in first-order and equational logic.
- Z3 operates as an SMT solver across various theories enabling efficient reasoning in first-order logic and beyond.
- InKreSAT is an incremental SAT solver that efficiently handles the satisfiability updating solutions as new constraints are added.
- And many others.

Deductive approach with theorem provers (cont.)

Input formats: TPTP, TPTP-FOF, FOL, Mizar etc. – it is easy to translate between them. Output – quite different, depending on the prover's operational principles; sometimes the results need to be interpreted.



Motivations

- Necessity of Formal Analysis: Logical specifications are critical for the formal analysis and precise design of complex systems, ensuring reliability and managing industrial requirements.
- Automation in Software Development: Automating the generation of these logical specifications is essential to promoting logical and deductive methods in software development.
- Challenges: Manual creation of formal specifications is difficult, error-prone, and time-consuming, particularly for engineers lacking experience, motivating the need for automation.
- Workflow Mining Expansion: Extending automation to include workflow mining processes addresses more complex, real-world applications, broadening the approach's applicability.

Objectives and aims

- Replicate and Extend: To replicate and extend previous methods for automating logical specification generation, that is moving from activity diagrams to event logs using workflow mining.
- Address New Challenges: To handle more complex tasks that require new behavioral patterns in generating specifications from event logs.
- Validate with Theorem Provers: To validate the approach through interactions with theorem provers, demonstrating its effectiveness.
- Explore New Research Areas: To explore new research paths in logic-based nowledge extraction from workflow mining.

Table of Contents

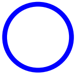
Processes and event data

The general area of interest or the roadmap

Processes and event data

The general area of interest or the roadmap

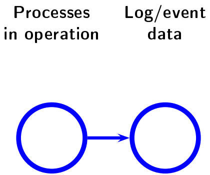
Processes in operation



Plenty of processes: industrial, system, application, security, network, administrative, resource management diagnostic, audit, etc

Processes and event data

The general area of interest or the roadmap

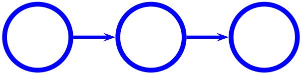


Plenty of formats:
Plain Text, CSV,
JSON, XML, Syslog,
Key-Value Pairs, etc.

Processes and event data

The general area of interest or the roadmap

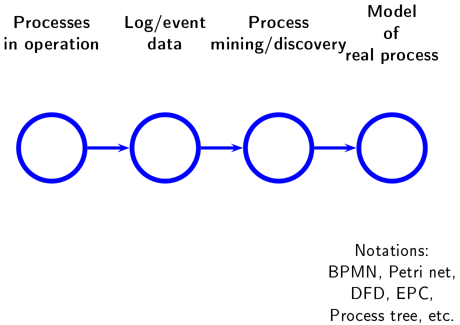
Processes in operation | Log/event data | Process mining/discovery



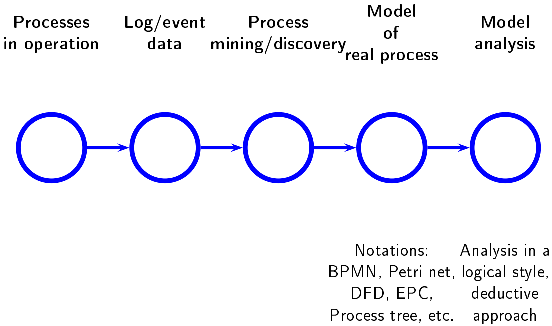
Algorithms:
Alpha Miner
Heuristics Miner
Inductive Miner

Processes and event data

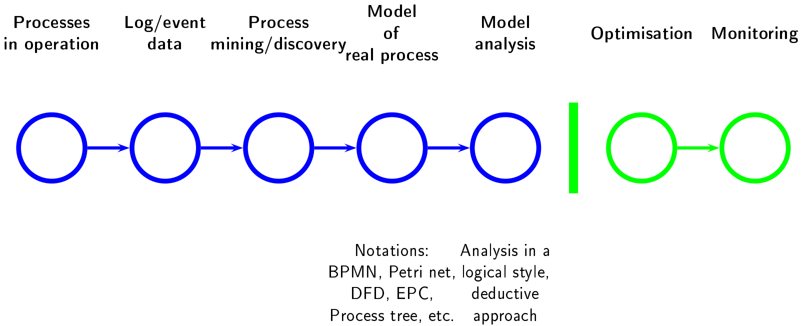
The general area of interest or the roadmap



The general area of interest or the roadmap

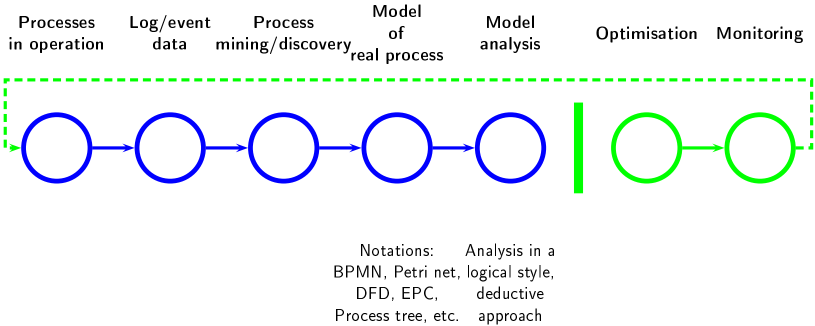


The general area of interest or the roadmap

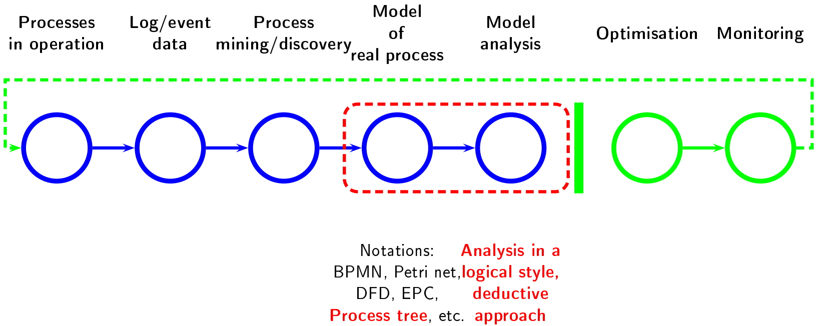


Processes and event data

The general area of interest or the roadmap



The general area of interest or the roadmap



Why workflow mining?

See: Wil van der Aalst: **How process mining improves the things you do not see**

https://www.youtube.com/watch?v=uP1stpUAv3c

Workflow mining algorithms

Prominent *workflow mining* algorithms include:

- **Alpha Miner**: Discovers process models by identifying direct succession relations between activities;
- **Heuristics Miner**: Uses heuristics to handle noise and complexity in event logs;
- **Inductive Miner**: Systematically infers process models by partitioning event logs into trace subsets.
- others: Fuzzy Miner, Split Miner, etc.

Inductive Miner – key features

- **Divide-and-conquer**: The algorithm splits the event log into smaller parts, which are analyzed to create models based on the identified segments.
- **Determinism**: Unlike many other algorithms, Inductive Miner always produces the same model for a given dataset, making it stable and predictable.
- **Precision**: Models generated by Inductive Miner align closely with the event logs and accurately represent the actual flow of processes.
- **Handling complex structures**: The algorithm is capable of recognizing structures such as parallelism, sequences, alternatives, and loops in processes.

Inductive Miner – output data

Inductive Miner generates a **process tree** as the base model. A process tree is a hierarchical structure that represents the process using nested logical operators and actions. Each node in the tree represents a specific operation or process structure, such as:

1. Sequence – actions executed in a specific order.
2. Parallelism – actions executed simultaneously.
3. Alternative (XOR) – a choice between several possible paths.
4. Loop – the possibility of repeating certain steps.

Although Inductive Miner always generates a process tree, in practice, the output can be presented in different formats like Petri nets, BPMN, or DFG. This is because the process tree is easily convertible into these formats.

Generating logical specifications – algorithm ΠBCD

- Aim – automation of workflow specification generation for behavioural models;
- Pattern-based Composition-driven approach – algorithm ΠBCD (or ΠC shortly);
- Published a few years ago as a journal paper.

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Pattern-based and composition-driven automatic generation of logical specifications for workflow-oriented software models

Radosław Klimek

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

ARTICLE INFO

Article history:
Received 18 June 2018
Received in revised form 14 January 2019
Accepted 12 February 2019
Available online 19 February 2019

Keywords:

ABSTRACT

This work relates to the automatic generation of logical specifications extracted directly from workflow-oriented behavioural models of software. The aim is to present a unified framework, which gives formal foundations and an algorithm for the logical specification generation process, allowing for further implementation works. Logical specifications are considered as sets of temporal logic formulas. The extraction process relies on the assumption that the entire developed model is structured purely by predefined workflow

ΠBCD – how does it work?

Assumptions:

- the workflow is structured and compositional,
- only predefined patterns (primitives) can be used, and
- we can freely nest the workflow structure.

ΠBCD – how does it work?

Assumptions:

- the workflow is structured and compositional,
- only predefined patterns (primitives) can be used, and
- we can freely nest the workflow structure.

---

1. The pattern expression W is a literal representation of the tree:

$$\Pi extract(Tree, \Pi) = W \quad (1)$$

Here, *Tree* represents the tree as a result of workflow mining, while Π denotes the set of approved patterns.

2. Algorithm ΠC generates the logical specification:

$$\Pi C(W, \Sigma) = L \quad (2)$$

where Σ signifies the logical definitions of patterns, and L is the resulting logical specification.

Reproducibility studies (RENE)

- Replicate and extend previous research on automating the generation of logical specifications, shifting from activity diagrams to event logs using workflow mining;
- A different team successfully re-implemented the algorithms in a new setup with fresh code and data, ensuring replicability;
- Experiments were conducted using real-world event logs, replacing simplified diagrams, leading to more complex and realistic scenarios.
- The approach was validated through multiple experiments, with logical specifications tested via theorem provers, confirming its effectiveness.

Table of Contents

Approved pattern structures

$$\Pi = \{Seq2, Seq3, Seq4, Seq5, Xor2, Xor3, And2, And3, Loop\}$$

Approved pattern structures

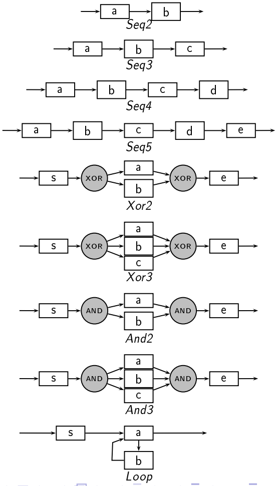
$\Pi = \{Seq2, Seq3, Seq4, Seq5, Xor2, Xor3, And2, And3, Loop\}$

- $Seq2(a, b) \equiv a; b$
- $Seq3(a, b, c) \equiv a; b; c$
- $Seq4(a, b, c, d) \equiv a; b; c; d$
- $Seq5(a, b, c, d, e) \equiv a; b; c; d; e$
- $Xor2(s, a, b, e) \equiv s; (a \otimes b); e$
- $Xor3(s, a, b, c, e) \equiv s; (a \otimes b \otimes c); e$
- $And2(s, a, b, e) \equiv s; (a \| b); e$
- $And3(s, a, b, c, e) \equiv s; (a \| b \| c); e$
- $Loop(s, a, b) \equiv$
  $s; a$ or $s; a; b; a$ or $s; a; b; a; b; a$ etc.

Approved pattern structures

$\Pi = \{Seq2, Seq3, Seq4, Seq5, Xor2, Xor3, And2, And3, Loop\}$

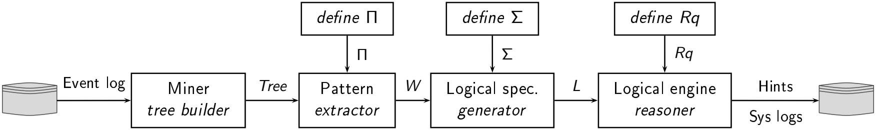
- $Seq2(a, b) \equiv a; b$
- $Seq3(a, b, c) \equiv a; b; c$
- $Seq4(a, b, c, d) \equiv a; b; c; d$
- $Seq5(a, b, c, d, e) \equiv a; b; c; d; e$
- $Xor2(s, a, b, e) \equiv s; (a \otimes b); e$
- $Xor3(s, a, b, c, e) \equiv s; (a \otimes b \otimes c); e$
- $And2(s, a, b, e) \equiv s; (a || b); e$
- $And3(s, a, b, c, e) \equiv s; (a || b || c); e$
- $Loop(s, a, b) \equiv$
  $s; a$ or $s; a; b; a$ or $s; a; b; a; b; a$ etc.



Fixed logical properties in PLTL for approved patterns

$$\Sigma = \{ \quad \mathsf{Seq2}(a,b) = \langle a, b, \Diamond a, \Box(a \Rightarrow \Diamond b), \Box\neg(a \wedge b)\rangle,$$
$$\mathsf{Seq3}(a,b,c) = \langle a, c, \Diamond a, \Box(a \Rightarrow \Diamond b), \Box(b \Rightarrow \Diamond c), \Box\neg(a \wedge b), \Box\neg(a \wedge c), \Box\neg(b \wedge c)\rangle,$$
$$\mathsf{Seq4}(a,b,c,d) = \langle a, d, \Diamond a, \Box(a \Rightarrow \Diamond b), \Box(b \Rightarrow \Diamond c), \Box(c \Rightarrow \Diamond d), \Box\neg(a \wedge b), \Box\neg(a \wedge c), \Box\neg(a \wedge d), \Box\neg(b \wedge c), \Box\neg(b \wedge d), \Box\neg(c \wedge d)\rangle,$$
$$\mathsf{Seq5}(a,b,c,s,e) = \langle a, e, \Diamond a, \Box(a \Rightarrow \Diamond b), \Box(b \Rightarrow \Diamond c), \Box(c \Rightarrow \Diamond d), \Box(d \Rightarrow \Diamond e), \Box\neg(a \wedge b), \Box\neg(a \wedge c), \Box\neg(a \wedge d), \Box\neg(a \wedge e), \Box\neg(b \wedge c), \Box\neg(b \wedge d), \Box\neg(b \wedge e), \Box\neg(c \wedge d), \Box\neg(c \wedge e), \Box\neg(d \wedge e)\rangle,$$
$$\mathsf{Xor2}(s,a,b,e) = \langle s, e, \Diamond s, \Box(s \Rightarrow (\Diamond a \wedge \neg\Diamond b) \vee (\neg\Diamond a \wedge \Diamond b)), \Box((a \vee b) \Rightarrow \Diamond e), \Box\neg(s \wedge a), \Box\neg(s \wedge b), \Box\neg(s \wedge e), \Box\neg(a \wedge b), \Box\neg(a \wedge e), \Box\neg(b \wedge e)\rangle,$$
$$\mathsf{Xor3}(s,a,b,c,e) = \langle s, e, \Diamond s, \Box(s \Rightarrow (\Diamond a \wedge \neg\Diamond b \wedge \neg\Diamond c) \vee (\neg\Diamond a \wedge \Diamond b \wedge \neg\Diamond c) \vee (\neg\Diamond a \wedge \neg\Diamond b \wedge \Diamond c)), \Box((a \vee b \vee c) \Rightarrow \Diamond e), \Box\neg(s \wedge a), \Box\neg(s \wedge b), \Box\neg(s \wedge c), \Box\neg(s \wedge e), \Box\neg(a \wedge b), \Box\neg(a \wedge c), \Box\neg(a \wedge e), \Box\neg(b \wedge c), \Box\neg(b \wedge e), \Box\neg(c \wedge e)\rangle,$$
$$\mathsf{And2}(s,a,b,e) = \langle s, e, \Diamond s, \Box(s \Rightarrow \Diamond a \wedge \Diamond b), \Box(a \Rightarrow \Diamond e), \Box(b \Rightarrow \Diamond e), \Box\neg(s \wedge (a \vee b)), \Box\neg((a \vee b) \wedge e)\rangle,$$
$$\mathsf{And3}(s,a,b,c,e) = \langle s, e, \Diamond s, \Box(s \Rightarrow \Diamond a \wedge \Diamond b \wedge \Diamond c), \Box(a \Rightarrow \Diamond e), \Box(b \Rightarrow \Diamond e), \Box(c \Rightarrow \Diamond e), \Box\neg(s \wedge (a \vee b \vee c)), \Box\neg((a \vee b \vee c) \wedge e)\rangle,$$
$$\mathsf{Loop}(s,a,b) = \langle s, a, \Diamond s, \Box(s \Rightarrow \Diamond a), \Box(a \Rightarrow (\Diamond b \wedge \Diamond a) \vee \neg\Diamond b), \Box(b \Rightarrow \Diamond a), \Box\neg(s \wedge a), \Box\neg(s \wedge b), \Box\neg(a \wedge b)\rangle \ \}$$

General scheme of data processing and flows



Event log – raw data collection;

Tree – discovered process tree;

W – pattern expression, intermediate representation;

L – logical specification, set of formulas/clauses;

Hints – results to interpret.

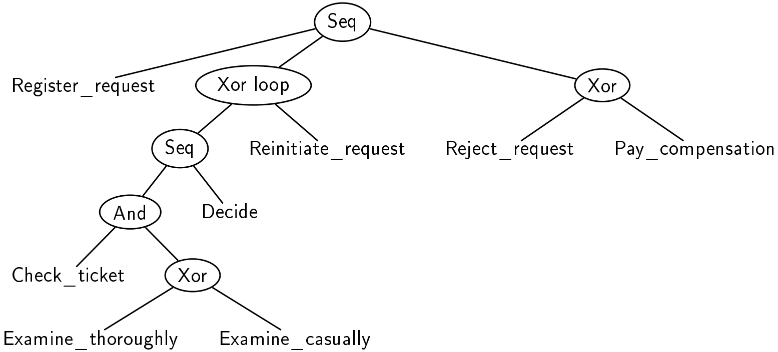
Table of Contents

Two event logs

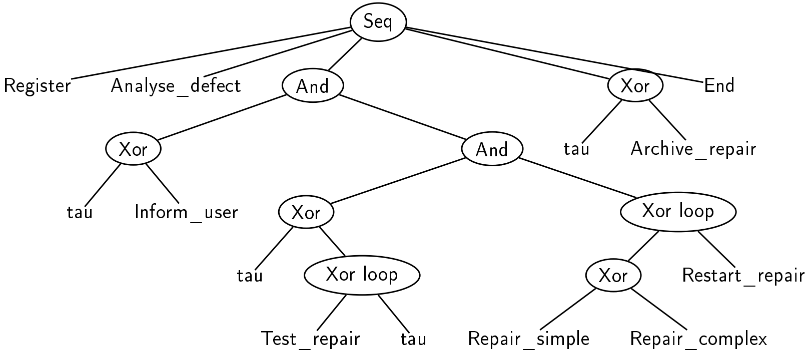
The following two event logs are considered:

1. the event log `running-example.xes`
   `https://pm4py.fit.fraunhofer.de/getting-started-page`
   simple customer service system;
2. the event log `repair-example.xes`
   `https://ai.ia.agh.edu.pl/pl:dydaktyka:dss:lab02`
   another simple customer service system.

The process tree for "running-example"



The process tree for "repair-example"



Pattern expressions

Literal representation of the both process trees:

$$W_1 = Seq3(Register\_request, Loop(Seq2(And2 (Check\_ticket, Xor2(Examine\_thoroughly, Examine\_casually)), Decide), Reinitiate\_request), Xor2(Reject\_request, Pay\_compensation)) \tag{3}$$

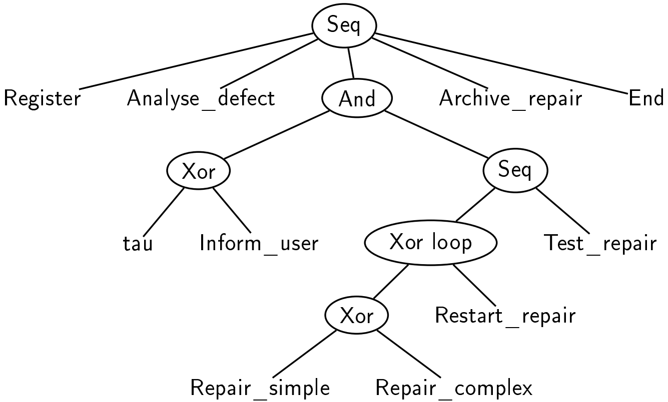
$$W_2 = Seq5(Register, Analyse\_defect, And2 (Xor2(tau1, Inform\_user), And2(Xor2(tau2, Loop(Test\_repair, tau3)), Loop(Xor2 (Repair\_complex, Repair\_simple), Restart\_repair))), Xor2(tau4, Archive\_repair), End) \tag{4}$$

Three groups of experiments

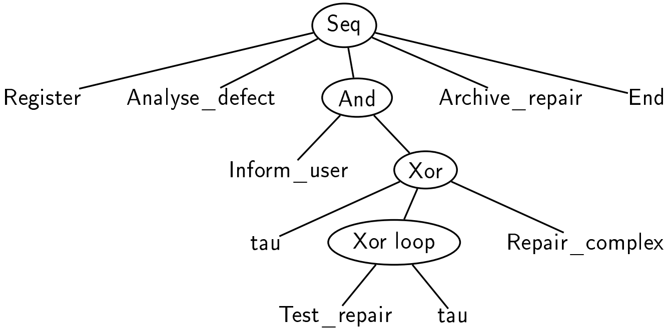
The conducted experiments are categorised into three groups:

1. testing the satisfiability of the obtained logical specifications – this process assesses the logical soundness of the derived formulas;
2. testing the logical relations between the obtained logical specifications – this method evaluates the logical consistency and interrelations among the derived sets of formulas;
3. testing the fulfilment of logical properties, expressed by new and separate formulas, with respect to the obtained logical specifications – this process introduces new requirement formulas and evaluates their relationship with those automatically derived from event logs.

Process tree for the event log `repair-example.xes` with a noise parameter value of 0.5



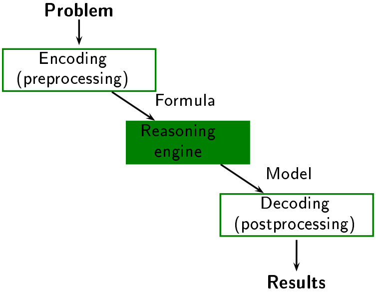
Process tree for the event log repair-example.xes with a noise parameter value of 1



Problem as formulas

Encoding a problem in a logical formula is the process of transforming a verbal description into formal logical expressions to clearly define conditions and relationships, enabling analysis and solution using deductive methods with **theorem provers**.

Features: precision, automation, verifiability, generalization, consistency.



Satisfability testing

- **Problem1.p** – the logical problem generated for the event log `running-example.xes` at the default noise threshold level;
- **Problem2.p** – the logical problem generated for the event log `repair-example.xes` at the default noise threshold level;
- **Problem3.p** – the logical problem generated for the event log `repair-example.xes` with the noise threshold set to 0.5;
- **Problem4.p** – the logical problem generated for the event log `repair-example.xes` with the noise threshold set to 1.

Satisfability testing – results

File	Vampire's output Comment	E's output Comment
Problem1.p	% SZS status Satisfiable for problem1 % Termination reason: Satisfiable Specification satisfied	# No proof found! # SZS staus Satisfiable Specification satisfied
Problem2.p	% Refutation found % Termination reason: Refutation Specification unsatisfied	# Proof found! # SZS status Unsatisfiable Specification unsatisfied
Problem3.p	% SZS status Satisfiable for problem3 % Termination reason: Satisfiable Specification satisfied	# No proof found! # SZS status Satisfiable Specification satisfied
Problem4.p	% SZS status Satisfiable for problem4 % Termination reason: Satisfiable Specification satisfied	# No proof found! # SZS status Satisfiable Specification satisfied

Logical relationships

- **Problem5.p** – the question is whether the specification for Problem 2 logically entails the specification for Problem 3;
- **Problem6.p** – the question is whether the specification for Problem 3 logically entails the specification for Problem 4;
- **Problem7.p** – the question is whether the specification for Problem 1 logically entails the specification for Problem 4;
- **Problem8.p** – the question is whether the specification for Problem 2 is logically equivalent to the specification for Problem 3;
- **Problem9.p** – the question is whether the specification for Problem 3 is logically equivalent to the specification for Problem 4;
- **Problem10.p** – the question is whether the specification for Problem 1 is logically equivalent to the specification for Problem 4.

Logical relationships – results

File	Vampire's output / Comment	E's output / Comment
Problem5.p	% SZS status Refutation % Termination reason: Refutation	# Proof found! # SZS status Contradictory Axioms
	The proof refutation	Contradiction between axioms
Problem6.p	% SZS status Counter Satisfiable for problem6 % Termination reason: Satisfiable	# No proof found! # SZS status Counter Satisfiable
	Satisfiability of axioms conjunction and the proof refutation	Satisfiability of axioms conjunction and the proof refutation
Problem7.p	% SZS status Counter Satisfiable for problem7 % Termination reason: Satisfiable	# No proof found! # SZS status Counter Satisfiable
	Satisfiability of axioms conjunction and the proof refutation	Satisfiability of axioms conjunction and the proof refutation
Problem8.p	% SZS status Refutation % Termination reason: Refutation	# Proof found! # SZS status Contradictory Axioms
	The proof refutation	Contradiction between axioms
Problem9.p	% SZS status Counter Satisfiable for problem9 % Termination reason: Satisfiable	# No proof found! # SZS status Counter Satisfiable
	Satisfiability of axioms conjunction and the proof refutation	Satisfiability of axioms conjunction and the proof refutation
Problem10.p	% SZS status Counter Satisfiable for problem10 % Termination reason: Satisfiable	# No proof found! # SZS status Counter Satisfiable
	Satisfiability of axioms conjunction and the proof refutation	Satisfiability of axioms conjunction and the proof refutation

Logical properties – examples

For the specification of Problem 1, two requirements have been defined, namely liveness:

$$\Box(Register\_request \Rightarrow \Diamond(Reject\_request \vee Pay\_compensation)) \quad (5)$$

and safety:

$$\Box\neg(Reject\_request \wedge Pay\_compensation) \quad (6)$$

In turn, for the specification of Problems 3 and 4, two requirements have been defined, that is liveness:

$$\Box(Register \Rightarrow \Diamond(Repair\_simple \vee Repair\_complex)) \quad (7)$$

and safety:

$$\Box\neg(Inform\_user \wedge null) \quad (8)$$

We expect these properties to be fulfilled.

Logical properties

- **Problem11.p** – the question is whether the specification for Problem 1 implies Formula (5);
- **Problem12.p** – the question is whether the specification for Problem 1 implies Formula (6);
- **Problem13.p** – the question is whether the specification for Problem 3 implies Formula (7);
- **Problem14.p** – the question is whether the specification for Problem 3 implies Formula (8);
- **Problem15.p** – the question is whether the specification for Problem 4 implies Formula (7);
- **Problem16.p** – the question is whether the specification for Problem 4 implies Formula (8).

Logical properties – results

File	Vampire's output Comment	E's output Comment
Problem11.p *liveness formula*	% SZS status Theorem for problem11 Proof confirmed	# Proof found! # SZS status Theorem Proof confirmed
Problem12.p *safety formula*	% SZS status Theorem for problem12 Proof confirmed	# Proof found! # SZS status Theorem Proof confirmed
Problem13.p *liveness formula*	% SZS status Theorem for problem13 Proof confirmed	# Proof found! # SZS status Theorem Proof confirmed
Problem14.p *safety formula*	% SZS status Theorem for problem14 Proof confirmed	# Proof found! # SZS status Theorem Proof confirmed
Problem15.p *liveness formula*	% SZS status Counter Satifsiable for problem15 Proof refuted	# No proof found! # SZS status Counter Satisfiable Proof refuted
Problem16.p *safety formula*	% SZS status Theorem for problem16 Proof confirmed	# Proof found! # SZS status Theorem Proof confirmed

Table of Contents

Evaluation

- **Validation**: The approach was validated through interactions with theorem provers, confirming the effectiveness of the generated specifications.
- **Research Expansion**: This work enters the significant area of business models and workflow mining, engaging a larger research community and extending the original research.
- **Impact on Software Engineering**: The ease of generating formal specifications and integrating them with theorem provers promotes a logical, deductive approach in automated software engineering.

Conclusions and further works

- **Replication**: Previous work was replicated by conducting experiments in new setups, shifting from simplified activity diagrams to real-world event log-based processes.
- **New Perspective**: The research now generates logical specifications from event logs, with extensive experimentation across multiple perspectives, providing strong empirical evidence.
- **Further Experiments**: Further experiments on a larger scale with larger data sets;
- **New Research Areas**: integration with optimization and monitoring.