Jan Mendling and Stefanie Rinderle-Ma, eds.: *Proceedings of EMISA 2016*,
Gesellschaft für Informatik, Bonn 2016

Plausibility Checking of Formal Business Process Specifications in Linear Temporal Logic
(Extended Abstract)[1]

Christoph Czepa, Huy Tran, Uwe Zdun[2]
Thanh Tran Thi Kim, Erhard Weiss, Christoph Ruhsam[3]

Abstract: Many approaches for keeping business processes in line with constraints stemming from various sources (such as laws, standards, internal policies, best practices, etc.) are based on Linear Temporal Logic (LTL). Creating LTL specifications is an error-prone task, which entails the risk that the formula does not match the intention of its creator. Manual testing is time-consuming and usually limited to a small amount of test cases. This paper proposes a semi-automatic *plausibility checking* approach for LTL-based specifications. Additionally to the LTL formula, the user specifies the desired behavior of the LTL specification in several smaller parts, namely by an initial truth value and one or more temporal queries (TQs). TQs change the truth value once a specific pattern of events occurred in an event trace. By this approach, a large set of test cases for the LTL specification can be created automatically. The work summarized in this extended abstract has been published in [Cz16].

Keywords: Linear Temporal Logic, Testing, Plausibility Check, Business Process Management

1 Introduction

Linear Temporal Logic (LTL) has become a *de facto* standard for defining system specifications due to its extensive use in model checking (cf. e.g., [Ro11]) and the possibility to automatically translate LTL formulas to nondeterministic finite automata (NFA) for runtime verification on finite traces (cf. e.g., [DGDMM14]). In business process management, LTL plays an important role in the verification of business processes. Elgammal et al. [El14] propose a *compliance request language (CRL)* with underlying LTL representations. Pesic & van der Aalst [PvdA06] suggest *ConDec*, a graphical language for the definition of declarative workflows with mappings to LTL formulas.

The creation of LTL formulas is a challenging and error-prone task that requires considerable knowledge and experience. It is hardly surprising that higher levels of abstraction, such as CRL and ConDec, are often preferred to creating LTL formulas from scratch. However, there are two major issues when solely trying to rely on a pattern-based approach. Firstly, formal patterns that precisely match the intention of the user might not be available. Hence, manually defining a new formula by modifying or combining existing

[1] The research leading to these results has received funding from the FFG project CACAO, no. 843461 and the WWTF, Grant No. ICT12-001. This paper contains an image licensed under CC [Wi]
[2] University of Vienna, Faculty of Computer Science, Software Architecture Research Group, Währingerstraße 29, 1090 Vienna, Austria, firstname.lastname@univie.ac.at
[3] Isis Papyrus Europe AG, Alter Wienerweg 12, 2344 Maria Enzersdorf, Austria, firstname.lastname@isis-papyrus.com

patterns or by creating a new specialized LTL formula might be required. Secondly, if an existing candidate pattern has been identified, it remains unclear whether the intention of the creator is really met. Either the meaning of the pattern could be misinterpreted or the LTL formula might contain errors.

To the best of our knowledge, finding errors in LTL formulas has not yet been investigated sufficiently. Salamah et al. [Sa05] propose to use a set of manually created test cases to check the plausibility of pattern-generated LTL formulas. However, this involves the user in the generation process of all the sample traces and the expected truth values at the end of these traces. This results usually in a small number of test cases since the manual specification of test cases is time-consuming.

2 Approach

The proposed *plausibility checking* approach aims at supporting users during the creation of LTL formulas. Figure 1 provides an approach overview.

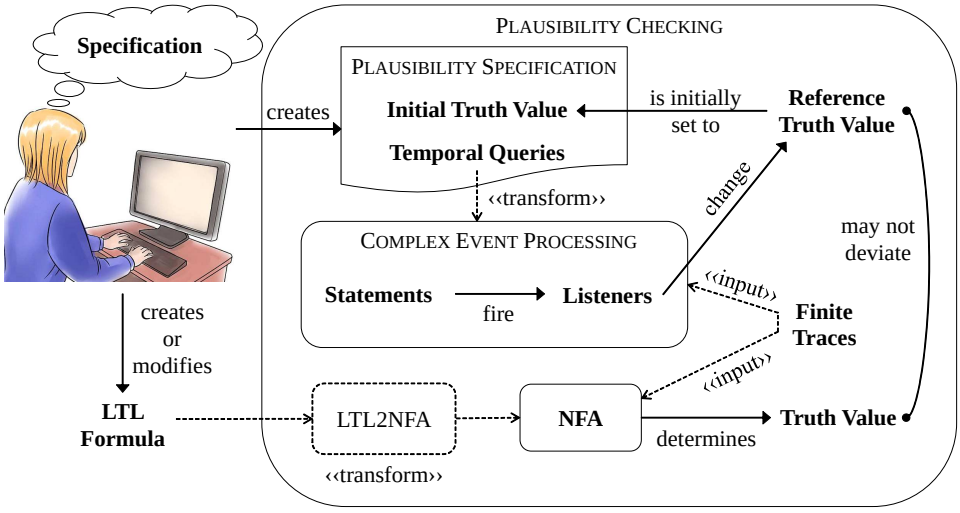

Fig. 1: Approach Overview

Whenever an LTL formula is created or modified, the user also creates a *plausibility specification* which is used to encode the desired behavior of the LTL formula. The plausibility specification consists of an initial truth value (either *temporarily satisfied* or *temporarily violated*) and one or more temporal queries (TQs) which describe truth value changes. A TQ consists of a temporal expression that uses a subset of *EPL (Event Processing Language)* [Esa], more specifically the operators *every* ($\forall$), *until* ($\mathscr{U}$), *leads-to* ($\rightsquigarrow$), *not* ($\neg$), *and* ($\wedge$), *or* ($\vee$), and a truth value to which the reference truth value is changed to once the temporal expression is matched by a given trace. A temporal query is of the form $e \Rrightarrow r$ where e is an temporal expression and r is a truth value. The expression formed by the operator $\Rrightarrow$ implies that there is a change of the truth value caused by the temporal ex-

pression e and the resulting truth value is r ($\bot$ for *temporarily violated*, $\top$ for *temporarily satisfied*, $\bot_{\mathscr{P}}$ for *permanently violated*, $\top_{\mathscr{P}}$ for *permanently satisfied*).

The temporal expression of a TQ is enacted as a statement by the *CEP (Complex Event Processing)* engine *Esper* [Esb] which fires a *listener* that turns the *reference truth value* to the defined truth value once the statement is matched. The LTL formula is transformed into a *nondeterministic finite automaton (NFA)* by the `LTL2NFA` algorithm [DGDMM14]. Both the NFA and CEP receive as inputs the elements of finite traces. These inputs lead to changes of both the *truth value*, which reflects the current state of the automaton, and the *reference truth value* determined by the CEP engine. In order to achieve a positive plausibility checking result, there must not be any deviation between the *truth value* (determined by NFA) and the *reference truth value* (determined by CEP) for all inputs. In case of a deviation, a counterexample trace and the truth values of both the LTL formula and the plausibility specification are being made available as a starting point for correction.

Since the reference truth value is determined by the plausibility specification, manually assigning a truth value to a trace is no longer necessary. Consequently, a large number of test cases can be generated automatically based on the plausibility specification.

3 Discussion of Applicability in the Context of Specification Patterns

In 1999, Dwyer et al. publish a paper entitled "Patterns in property specifications for finite-state verification" [DAC99] alongside with a constraint pattern collection called "Property Specification Patterns" which is available online[3]. In the FAQs, the following information is stated: *"Mappings were validated primarily by peer review amongst the project members, with assistance from several other people on selected pattern mappings. Some of the mappings also underwent testing by running existing FSV*[4] *tools to analyze small finite-state transition systems which encode (un)satisfying sequences of states/events."*. Consequently, we cannot assume the correctness of a pattern representation. As an example for an LTL formula that does not match our understanding of the corresponding pattern, we are now going to discuss the *Precedence After* pattern (*after a: b precedes c*) and its LTL representation $\mathscr{G}(\neg a) \vee \mathscr{F}(a \wedge (\neg c \mathscr{W} b))$. We formulate a single TQ $a \rightsquigarrow \neg b \mathscr{U} c \Rrightarrow \bot_{\mathscr{P}}$ which causes the truth value to switch to permanently violated once there has been an a but thereafter no b until c occurs. Plausibility checking notifies us of the counterexample $[a,c,a]$ where the LTL formula is satisfied. According to the pattern scopes defined by Dwyer et al. [DAC99], the after scope *after a* starts at the first occurrence of a. Thus, with the occurrence of the trace $[a,c]$ the pattern becomes permanently violated because b should have happened in between the first occurrence of a and the occurrence of c. Consequently, every suffix of $[a,c]$ must not cause any further change of the truth value of the pattern, so there must be something wrong with the LTL formula. The reason why the LTL formula is incorrect becomes obvious when we substitute the *weak until* by one of its equivalences. Then the modified formula is given as $\mathscr{G}(\neg a) \vee \mathscr{F}(a \wedge ((\neg c \mathscr{U} b) \vee \mathscr{G}(\neg c)))$. The trace $[a,c,a]$ meets the LTL formula by satisfying the subformula $\mathscr{F}(a \wedge \mathscr{G}(\neg c))$ through the

[3] `http://patterns.projects.cis.ksu.edu`
[4] Finite State Verification

second occurrence of a because the trace ends there and c is not present after this a. From our point of view, a correct LTL formula for this pattern is $(\mathcal{G}\neg a) \vee (\neg a \,\mathcal{U}\, (a \wedge (\neg c \,\mathcal{W}\, b)))$ because here it is ensured that only the first a starts the scope.

4 Conclusion

This papers discusses a plausibility checking approach which requires the user to specify the desired behavior of the LTL specification in several smaller parts, namely by an initial truth value and temporal queries. This enables the evaluation of LTL formulas against a large set of automatically generated test cases that are based on the provided plausibility specification. Existing pattern-based approaches, such as CRL and ConDec, may benefit from this approach whenever it becomes necessary to extend the set of supported constraints.

References

[Cz16] Czepa, Christoph; Tran, Huy; Zdun, Uwe; Tran, Thanh; Weiss, Erhard; Ruhsam, Christoph: Plausibility Checking of Formal Business Process Specifications in Linear Temporal Logic. In: 28th International Conference on Advanced Information Systems Engineering (CAiSE'16), Forum Track. June 2016.

[DAC99] Dwyer, Matthew B.; Avrunin, George S.; Corbett, James C.: Patterns in Property Specifications for Finite-state Verification. In: 21st International Conference on Software Engineering (ICSE). ACM, pp. 411–420, 1999.

[DGDMM14] De Giacomo, Giuseppe; De Masellis, Riccardo; Montali, Marco: Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In: AAAI. AAAI Press, pp. 1027–1033, 2014.

[El14] Elgammal, Amal; Turetken, Oktay; van den Heuvel, Willem-Jan; Papazoglou, Mike: Formalizing and applying compliance patterns for business process compliance. Software & Systems Modeling, pp. 1–28, 2014.

[Esa] EsperTech Inc.: , EPL Reference. `http://www.espertech.com/esper/release-5.1.0/esper-reference/html/event_patterns.html`. Last accessed: July 25, 2016.

[Esb] EsperTech Inc.: , Esper. `http://www.espertech.com/esper/`. Last accessed: July 25, 2016.

[PvdA06] Pesic, M.; van der Aalst, W. M. P.: A Declarative Approach for Flexible Business Processes Management. In: BPM Workshops. Springer, pp. 169–180, 2006.

[Ro11] Rozier, Kristin Y.: Survey: Linear Temporal Logic Symbolic Model Checking. Comput. Sci. Rev., 5(2):163–203, May 2011.

[Sa05] Salamah, Salamah; Gates, Ann; Roach, Steve; Mondragon, Oscar: Verifying Pattern-Generated LTL Formulas: A Case Study. In: Model Checking Software, volume 3639 of LNCS, pp. 200–220. Springer, 2005.

[Wi] Wikihow: . `http://de.wikihow.com/Ein-Software-Ingenieur-werden`. Licensed under: `http://creativecommons.org/licenses/by-nc-sa/3.0/` Last accessed: July 25, 2016.