An ASP-Based Framework for Solving Problems Related to Declarative Process Specifications

Isabelle Kuhlmann[1], Carl Corea[2] and John Grant[3]

[1]*Artificial Intelligence Group, University of Hagen, Germany*
[2]*Institute for IS Research, University of Koblenz, Germany*
[3]*University of Maryland, College Park, USA*

Abstract
We present a framework of answer set programming-based solutions for various problems related to declarative process specifications. Specifically, the framework offers implementations for conformance checking, satisfiability checking, and two different inconsistency measures. Since the aforementioned problems are represented in a fragment of linear temporal logic, the framework could also prove useful for a broader range of applications beyond process specifications.

Keywords
answer set programming, declarative process specifications, conformance checking, satisfiability checking, inconsistency measurement, linear temporal logic

1. Introduction

Declarative process specifications are a crucial concept for modeling business processes. Such specifications can be expressed by means of linear temporal logic (LTL) [1], i.e., a specification can be formulated as a set of LTL formulas. The intuition behind this is that a business process is a (temporal) sequence of actions. Problems in this area include *conformance checking* (does a given sequence of actions conform to a specification?), *satisfiability checking* (is a given specification free of conflicts?), as well as *inconsistency measurement* (if a specification contains conflicts, how severe are they?) [2, 3].

For conformance checking, existing algorithmic approaches (see [4] for an overview) are often geared towards a specific modeling language, such as *Declare*, which means that it might not be possible to use them for arbitrary LTL formulas. Also, many existing approaches (also some based on answer set programming [5, 6]) represent specifications with finite state automata. As this transformation and related automata operations may introduce a computational burden, in this work, our presented framework encodes the LTL semantics in a native manner, such that no transformation is needed.

Regarding the analysis of inconsistency in declarative specifications, virtually all existing approaches also rely on the mentioned automata representation (and verify inconsistency by checking whether the automata product is empty) [7]. Also, our framework is the first to offer practical solutions for the actual measurement of inconsistency in such specifications.

In summary, we present a framework based on answer set programming (ASP) featuring the following qualities:

1. It combines implementations for different problems in the area of declarative process specifications, in particular *conformance checking*, *satisfiability checking*, and *inconsistency measurement* (w.r.t. two distinct inconsistency measures).
2. The encodings are based on LTL directly, meaning no preprocessing, such as the transformation to automata [5], is required. It is sufficient to provide a set of LTL formulas (and, for conformance checking, a set of sequences of actions).
3. Since our implementations work directly on LTL formulas, they can be used in other fields as well.
4. Due to the declarative nature of ASP our framework delivers interpretable solutions. E.g., in the case of inconsistency measurement, we can identify which parts of a set of formulas representing a specification are involved in a conflict.

We present some preliminaries on LTL and ASP in Section 2 and present an overview of our framework in Section 3.

2. Preliminaries

In the following, we define the specific variant of LTL our work is based on, as well as the problems we address in our framework, and provide a basic overview of ASP.

Linear Temporal Logic on Fixed Traces Let At be a set of propositional symbols, and $t_0, \ldots, t_m$ a linear

21st International Workshop on Nonmonotonic Reasoning, September 2-4, 2023, Rhodes, Greece
isabelle.kuhlmann@fernuni-hagen.de (I. Kuhlmann); ccorea@uni-koblenz.de (C. Corea); grant@cs.umd.edu (J. Grant)
0000-0001-9636-122X (I. Kuhlmann); 0000-0001-7503-7703 (J. Grant)

sequence of temporal states. In this work we consider sequences of finite [8] and fixed length, and refer to the corresponding fragment of LTL as *linear temporal logic on fixed traces* ($\mathrm{LTL_{ff}}$) [3]. Formulas are closed under the unary operator $\mathbf{X}$ (*next*) and the binary operator $\mathbf{U}$ (*until*), in addition to the classical operators $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation). We also consider the auxiliary operators $\mathbf{F}$ (*eventually*) and $\mathbf{G}$ (*globally*), with $\mathbf{F}\varphi$ being defined as $\top\mathbf{U}\varphi$, and $\mathbf{G}\varphi$ being defined as $\neg\mathbf{F}\neg\varphi$.

An $\mathrm{LTL_{ff}}$-interpretation $\hat{\omega}$ w.r.t. At is a function mapping each state t and proposition a to 1 (true) or 0 (false), i.e., $\hat{\omega}(t,a) = 1$ if a is assigned 1 at t. A formula φ is *satisfied* by an interpretation $\hat{\omega}$, denoted $\hat{\omega} \models \varphi$, iff $\hat{\omega}, t_0 \models \varphi$. Further, $\hat{\omega}, t_i \models \varphi$ is inductively defined as

$$\hat{\omega}, t_i \models a \text{ iff } \hat{\omega}(t_i, a) = 1 \text{ for } a \in \mathsf{At}$$
$$\hat{\omega}, t_i \models \neg\varphi \text{ iff } \hat{\omega}, t_i \not\models \varphi$$
$$\hat{\omega}, t_i \models \varphi_1 \wedge \varphi_2 \text{ iff } \hat{\omega}, t_i \models \varphi_1 \text{ and } \hat{\omega}, t_i \models \varphi_2$$
$$\hat{\omega}, t_i \models \varphi_1 \vee \varphi_2 \text{ iff } \hat{\omega}, t_i \models \varphi_1 \text{ or } \hat{\omega}, t_i \models \varphi_2$$
$$\hat{\omega}, t_i \models \mathbf{X}\varphi \text{ iff } i < m \text{ and } \hat{\omega}, t_{i+1} \models \varphi$$
$$\hat{\omega}, t_i \models \varphi_1\mathbf{U}\varphi_2 \text{ iff } \hat{\omega}, t_j \models \varphi_2 \text{ with } j \in \{i+1, \ldots, m\} \text{ and } \hat{\omega}, t_k \models \varphi_1 \, \forall k \in \{i, \ldots, j-1\}$$

w.r.t. any interpretation $\hat{\omega}$, and $t_i \in \{t_0, \ldots, t_m\}$. An interpretation $\hat{\omega}$ satisfies a set of formulas $\mathcal{K}$ iff $\forall\varphi \in \mathcal{K} : \hat{\omega} \models \varphi$. We also refer to a satisfiable set of formulas as *consistent*, and to an unsatisfiable set of formulas as *inconsistent*.

Problems Related to Declarative Process Specifications *Conformance checking* (CC) is the task of deciding whether a given *trace* $s = \langle s_0, \ldots, s_m \rangle$, consisting of a sequence of activities or actions, conforms to a given specification $\mathcal{K}$. Note that, following the literature in business process modeling, only the atom indicated by s_i ($i \in \{0, \ldots, m\}$) is set to 1 in state t_i, while all other atoms are set to 0 in t_i [8]. We denote the resulting interpretation as $\hat{\omega}^s$. Thus, formally, we check whether $\hat{\omega}^s \models \mathcal{K}$. In addition, we allow individual formulas φ to be *vacuously* satisfied if none of the atoms occurring in φ are contained in s.

As a generalization of CC, we define the problem of *satisfiability checking* (SC) as the task of deciding whether there exists a trace s of length m, and a corresponding interpretation $\hat{\omega}^s$, s.t. $\hat{\omega}^s \models \mathcal{K}$, for a given specification $\mathcal{K}$. Again, only atom s_i is set to true in state t_i, and formulas can be vacuously satisfied.

To allow for our framework to be used beyond declarative process specifications, we additionally provide implementations for the "general" problems of CC and SC in $\mathrm{LTL_{ff}}$. To be precise, we lift the restriction that only atom s_i is allowed to be true in state t_i, and we prohibit formulas to be vacuously satisfied. We denote the corresponding problems as MC (since the "general" variant of CC is actually closer to the definition of the problem of *model checking*), and $\mathrm{SC_{LTL_{ff}}}$, respectively.

In *inconsistency measurement* (IM) [9], the goal is to quantitatively assess the level of inconsistency in a given knowledge base. Hence, we aim to map a given specification $\mathcal{K}$ to a numerical value. Although applied in various other domains, it is a relatively new concept in the area of declarative process specifications. In [3], the authors first introduced two inconsistency measures for $\mathrm{LTL_{ff}}$ based on paraconsistent semantics, which are implemented in the framework.

Answer Set Programming Answer set programming (ASP) [10] is a declarative programming paradigm, where the objective is to represent a given problem in a logical format (an *extended logic program*) s.t. the models of this representation (the *answer sets*) express solutions of the initial problem. An extended logic program is comprised of *rules* of the form "$r = a_0 \texttt{:-}\ a_1, \ldots, a_n, \texttt{not}\, a_{n+1}, \ldots, \texttt{not}\, a_m.$" with a_i ($0 \leq i \leq n \leq m$) being atoms, and "not" indicating default negation [11]. An atom is a predicate $p(v_1, \ldots, v_k)$ with $k \geq 0$, with each $v_1, \ldots, v_k$ being either a constant or a variable[1]. Further, ":-" can be interpreted as "if", a "," as "and", and a "." marks the end of a rule. If an atom/rule/program does not contain any variables, it is referred to as *ground*.

An ASP rule r (as illustrated above) is comprised of a *head* $H(r) = a_0$ and a *body* $B(r) = \{a_1, \ldots, a_n, \texttt{not}\, a_{n+1}, \ldots, \texttt{not}\, a_m\}$. If $H(r)$ is empty, r is called a *constraint*, and if $B(r) = \emptyset$, r is called a *fact*. We further divide the elements of $B(r)$ into $B^+(r) = \{a_1, \ldots, a_n\}$ and $B^-(r) = \{a_{n+1}, \ldots, a_m\}$. A set X of ground atoms is a *model* of a ground logic program P if for all $r \in P$, $H(r) \in X$ whenever $B^-(r) \cap X = \emptyset$ and $B^+(r) \subseteq X$. The *reduct* [13] of a program P w.r.t. X is defined as $P^X = \{H(r) \texttt{:-}\ B^+(r) \mid B^-(r) \cap X = \emptyset, r \in P\}$. If X is a subset-minimal model of P^X, then X is called an *answer set* of P.

3. Framework

The framework[2] we present combines ASP-based implementations of all problems defined in the preceding section (namely, CC, MC, SC, $\mathrm{SC_{LTL_{ff}}}$, and IM w.r.t. the two measures defined in [3]). The implementations are done in C++, and the ASP solver we use is Clingo [12]. Note that the implementations for CC and SC have been introduced in a recent work by the authors [14], and

[1]Following the Clingo [12] syntax, we denote constants by strings starting with a lowercase letter, and variables by strings starting with an uppercase letter. Anonymous variables (which do not recur within a rule) are denoted by "_".

[2]https://github.com/aig-hagen/ASP_for_LTL

the implementations for the two inconsistency measures have been proposed in a work that is currently under review [15].

Since due to page limitations, we cannot explain the ASP encodings for each problem in detail, we take CC as an example to demonstrate our overall approach. Our framework takes as input for CC (i) a file containing a specification $\mathcal{K}$ (i.e., a set of LTL formulas), and (ii) a file containing a set of traces S. For each trace $s \in S$, we encode CC in ASP, and if an answer set can be derived, the given problem instance $(\mathcal{K}, s)$ is satisfiable, otherwise it is unsatisfiable.

In order to encode a given problem instance $(\mathcal{K}, s)$ in ASP, we begin by representing each atom $a \in \mathsf{At}(\mathcal{K})$ as a fact "atom(a).". In the same manner, each formula $\varphi \in \mathcal{K}$ is modeled as kbElement(φ). Next, we represent the "type" of each (sub-)formula; e.g., a conjunction $\varphi = \varphi_1 \wedge \varphi_2$ is represented as conjunction($\varphi, \varphi_1, \varphi_2$). The remaining operators (disjunction, negation, next, until, eventually, and globally) are modeled analogously. If a formula φ consists of a single atom a, it is represented by formulaIsAtom(φ, a). To represent a trace s in ASP, we first define $|s|$ states, where the final state $m = |s| - 1$ is represented as finalState(m). The states $\{t_0, \ldots, t_m\}$ are then modeled by adding the rule "state(0..M) :- finalState(M)." to the encoding. Further, we model that s_i ($i \in \{0, \ldots, m\}$) is true in state t_i by adding "true(s_i, i).".

An overview of our encoding of logical entailment is given in Listing 1. Essentially, we can directly follow the definition of each operator (see Section 2)—e.g., a formula $\mathbf{X}\varphi$ is true in t_i if $i < m$ and φ is true in t_{i+1} (line 5). Moreover, lines 2–4 describe the classical operators, lines 6–8 the remaining LTL-specific ones, and line 1 the case if a formula is in fact an atom. Finally, we add an integrity constraint (line 9) which ensures that no answer set can be derived if any formula $\varphi \in \mathcal{K}$ evaluates to "not true" in state t_0, i.e., every formula must be satisfied.

```
1 true(F,S):- formulaIsAtom(F,A), state(S), true(A,S).
2 true(F,S):- conjunction(F,G,H), state(S), true(G,S),
      true(H,S).
3 true(F,S):- disjunction(F,G,H), state(S),
      1{true(G,S); true(H,S)}.
4 true(F,S):- negation(F,G), state(S), not true(G,S).
5 true(F,Si):- next(F,G), state(Si), Sj=Si+1, Si<M,
      finalState(M), true(G,Sj).
6 true(F,Si):- until(F,G,H), state(Si), state(Sj),
      Sj>Si, Sj<=M, finalState(M), X{true(G,S):
      state(S), S>=Si, S<Sj}X, X=Sj-Si, true(H,Sj).
7 true(F,Si):- globally(F,G), state(Si), X{true(G,S):
      state(S), S>Si}X, finalState(M), X=M-Si.
8 true(F,Si):- eventually(F,G), state(Si), true(G,Sj),
      state(Sj), Sj>Si.
9 :- not true(F,0), kbElement(F), state(0).
```

Listing 1: Encoding of logical entailment.

The approach described above can directly be used to solve MC as well[3]—the framework simply requires a different input format which specifies a set of atoms to be true per state (instead of a single atom). Furthermore, the approach can be easily modified to solve SC and $\mathrm{SC}_{\mathrm{LTL_{ff}}}$. Intuitively, for $\mathrm{SC}_{\mathrm{LTL_{ff}}}$ we merely need to add a rule that "guesses" an interpretation which is then checked to be satisfiable, and for SC we additionally need to handle vacuous satisfiability directly within ASP.

The ASP encodings for the two inconsistency measures from [3] are a bit more intricate, since we need to model paraconsistent semantics. E.g., it is not sufficient to model only the "true" case w.r.t. the different operators. Instead, we use truthValue(φ, t_i, θ) to explicitly represent a truth value θ for a formula φ in t_i. However, the overall approach is still the same—we directly model logical entailment.

4. Conclusions and Future Work

We introduced a framework that provides ASP-based solutions for multiple problems related to LTL, targeted at the field of declarative process specifications. Our approach does not require any preprocessing (such as the computation of automata), and can be used in other LTL-related domains as well. In terms of future work, we aim to extend our set of implementations, e.g., by considering culpability measures (which indicate a level of blame for the inconsistency of a knowledge base w.r.t. a given formula or atom). Also, we aim to apply our framework to more non-monotonic settings, e.g., inconsistency measurement in the presence of superiority relations between LTL formulas.

References

[1] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57.

[2] J. Carmona, B. van Dongen, M. Weidlich, Conformance checking: foundations, milestones and challenges, in: Process Mining Handbook, Springer, 2022, pp. 155–190.

[3] C. Corea, J. Grant, M. Thimm, Measuring inconsistency in declarative process specifications, in: 20th International Conference on Business Process Management, Springer, 2022, pp. 289–306.

[4] S. Dunzer, M. Stierle, M. Matzner, S. Baier, Conformance checking: a state-of-the-art literature review, in: Proceedings of the 11th international conference on subject-oriented business process management, 2019, pp. 1–10.

[3]Note that in the case of CC, vacuously satisfied formulas are simply not encoded in ASP; in the case of MC, every formula is encoded.

[5] F. Chiariello, F. M. Maggi, F. Patrizi, Asp-based declarative process mining, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2022, pp. 5539–5547.

[6] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann, C. Vidal, Linear-time temporal answer set programming, Theory and Practice of Logic Programming 23 (2023) 2–56.

[7] C. Di Ciccio, F. M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in decl. process models, Information Systems 64 (2017) 425–446.

[8] G. D. Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proceedings of the 23rd IJCAI, Beijing, AAAI, 2013, pp. 854–860.

[9] J. Grant, Classifications for inconsistent theories, Notre Dame Journal of Formal Logic 19 (1978) 435–444.

[10] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synthesis Lectures on AI and Machine Learning 6 (2012) 1–238.

[11] R. Reiter, A logic for default reasoning, 1980.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with Clingo 5, in: Technical Communications of ICLP, OASICS, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:15.

[13] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. ICLP/SLP, MIT Press, 1988, pp. 1070–1080.

[14] I. Kuhlmann, C. Corea, J. Grant, Non-automata based conformance checking of declarative process specifications based on ASP, in: First International Workshop on Formal Methods for Business Process Management, 2023. In Press.

[15] C. Corea, I. Kuhlmann, M. Thimm, J. Grant, Paraconsistent reasoning and inconsistency measurement in declarative process specifications, Under review.