Automata-based LTL$_f$ Satisfiability Checking via ASP

Andrea Cuteri[1], Giuseppe Mazzotta[1], Rafael Peñaloza[2] and Francesco Ricca[1]

[1]University of Calabria, Italy
[2]University of Milano-Bicocca, Italy

Abstract

Linear Temporal Logic over finite traces (LTL$_f$) stands as a prominent and highly valuable formalism with application in various areas, including AI, process mining, and model checking among many others. The key reasoning task for LTL$_f$ is satisfiability checking, which amounts to verifying whether an input formula admits temporal models. In this paper we propose a novel approach to satisfiability checking based on Answer Set Programming (ASP). The idea is to encode in an ASP program the search for a finite state automaton that recognizes (a subset of) the language of the LTL$_f$ formula given in input. An experimental analysis demonstrates the viability of our approach.

Keywords

Linear Temporal Logic, Satisfiability Checking, and Answer Set Programming

1. Introduction

Linear Temporal Logic over finite traces (LTL$_f$) [2] stands as a prominent formalism with application in various areas, including AI [3, 4, 5, 6], process mining [7, 8], model checking [9], and many others. The main reasoning task in LTL$_f$ is satisfiability checking, which involves determining if there exist temporal models that satisfy the formula. Traditionally, this problem is approached through automata-based methods [10] or by reducing it to SAT, i.e., the satisfiability problem over Boolean formulas [2] which is supported by highly efficient implementations [11]. Another promising formalism in this context is Answer Set Programming (ASP) [12], a well-established logic-based framework for non-monotonic reasoning, which combines a declarative language based on stable model semantics [13] with efficient implementations [14], and academic and industrial applications [15, 16, 17].

In this paper we propose an approach to LTL$_f$ satisfiability checking based on ASP. We build on a different method for deciding satisfiability of LTL$_f$ formulas based on the classical automata-based construction originally developed for LTL (over infinite traces) [18]. In a nutshell, the approach builds an automaton [19]—for LTL a Büchi automaton—whose accepting runs correspond to temporal models satisfying the formula. The construction for LTL$_f$ uses non-deterministic finite automata (NFA) to deal with finite traces only. We encode the search for (a subset of) the so-called φ-automaton, which can be used to test satisfiability, into an ASP program. Indeed, if one can establish a connection between an initial state and a final state in the NFA corresponding to a given formula φ, then φ is satisfiable (and *vice versa*) [20].

The resulting ASP-based formulation has several features that makes it interesting. First of all, from a pure knowledge representation perspective, it is a declarative solution based on a direct and uniform encoding of LTL$_f$ satisfiability in the sense that the non-ground ASP program is a general solution that can be used over different instances, whereas the SAT-based ones have to resort to a procedural step that generates a specific propositional formula for the given input. Moreover, the answer sets of the program have a direct association with some of the models of the original formula, and provide as a byproduct a witness of satisfiability. Another property, which is interesting from a computational point

AI4CC-IPS-RCRA-SPIRIT 2024: International Workshop on Artificial Intelligence for Climate Change, Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy. November 25-28th, 2024, Bolzano, Italy [1].
cuteri.andrea@gmail.com (A. Cuteri); giuseppe.mazzotta@unical.it (G. Mazzotta); rafael.penaloza@unimib.it (R. Peñaloza); francesco.ricca@unical.it (F. Ricca)
0000-0003-0125-0477 (G. Mazzotta); 0000-0002-2693-5790 (R. Peñaloza); 0000-0001-8218-3178 (F. Ricca)

of view is that our approach does not require an explicit construction of the entire φ-automaton (which is exponential in the size of the input) to check satisfiability. Indeed, the need for such a construction is generally seen as detrimental to the use of automata-based decision processes, by requiring an exponential consumption of resources before any actual "reasoning" is performed. An additional benefit for our encoding is that it allows for structure sharing of sub-formulas; that is, when a sub-formula ψ appears repeatedly in the formula φ, our encoding represents only one copy of ψ within the encoding. This allows us to transfer some of the benefits of propositional formula compilation [21] to the LTL_f setting and thus improve the overall performance.

On the practical side, a thorough empirical analysis reveals that our approach is highly competitive against state of the art approaches on benchmarks available in the literature and demonstrated a significant scalability on hard instances. These findings confirm the viability of our approach, and justify further work in this direction.

2. Linear Time Temporal Logic on Finite Traces

We briefly introduce the notions of linear temporal logic over finite traces (LTL_f) which are necessary for understanding our work. LTL_f [20] is a temporal logic over linear, discrete timepoints which is characterised by considering only *finite* traces; that is, models must have a final timepoint. Syntactically, it is equivalent to the well known LTL (over infinite time) [18]; that is, LTL_f formulas are built, starting from a set $\mathscr{P}$ of propositional formulas, through the grammar

$$\varphi ::= x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathcal{X}\varphi \mid \varphi \,\mathcal{U}\, \varphi$$

where $x \in \mathscr{P}$. Briefly, LTL_f extends propositional logic with the "next" ($\mathcal{X}$) and "until" ($\mathcal{U}$) operators. The semantics is based on finite temporal models, which are just finite sequences of propositional valuations. Formally, a *valuation* is a set $\mathscr{V} \subseteq \mathscr{P}$ that intuitively describes which variables are true. A *temporal model* is a finite sequence $\mathscr{M}$ of valuations.

Definition 1 (satisfiability). *Let $\mathscr{M} = \mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_n$ be a temporal model.* Satisfaction *of a formula φ at time k, $1 \leq k \leq n$ (denoted by $\mathscr{M}, k \vDash \varphi$) is defined inductively as follows:*

- *if $x \in \mathscr{P}$, then $\mathscr{M}, k \vDash x$ iff $x \in \mathscr{V}_k$;*
- *$\mathscr{M}, k \vDash \neg\varphi$ iff $\mathscr{M}, k \nvDash \varphi$;*
- *$\mathscr{M}, k \vDash \varphi \wedge \psi$ iff $\mathscr{M}, k \vDash \varphi$ and $\mathscr{M}, k \vDash \psi$;*
- *$\mathscr{M}, k \vDash \mathcal{X}\varphi$ iff $k < n$ and $\mathscr{M}, k+1 \vDash \varphi$; and*
- *$\mathscr{M}, k \vDash \varphi \,\mathcal{U}\, \psi$ iff there exists $\ell, k \leq \ell \leq n$ such that (i) $\mathscr{M}, \ell \vDash \psi$ and (ii) for all $k \leq j < \ell$ $\mathscr{M}, j \vDash \varphi$.*

$\mathscr{M}$ satisfies φ *(denoted as $\mathscr{M} \vDash \varphi$) iff $\mathscr{M}, 1 \vDash \varphi$. In that case, we say that φ is* satisfiable.

Satisfiability of a formula of the form $\mathcal{X}\varphi$ requires that there is at least one successive timepoint. By this semantics, $\varphi \,\mathcal{U}\, \psi$ is equivalent to $\psi \vee (\varphi \wedge \mathcal{X}(\varphi \,\mathcal{U}\, \psi))$; in other words, we can decide to satisfy an until formula now, or wait until the next point in time.

In the literature one can find other temporal constructors like the *weak next*, which is true also in cases that no next timepoint exists, or the *eventually* and *always* in the future operators. They can all be expressed in terms of the constructors that we use [22]. We choose to preserve the limited syntax to handle reasoning more effectively. Yet, we will use the disjunction $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ and the tautology $\top := p \vee \neg p$, where p is any propositional variable for brevity.

3. Automata for LTL_f satisfiability

A well-known method for deciding the satisfiability of an LTL_f formula is based on the construction of an automaton which, intuitively, accepts the temporal models that satisfy it. Thus, an emptiness

test of the automaton yields a decision procedure for satisfiability. Since our method is based on this construction, we briefly recall it here assuming a basic knowledge of finite automata (for the full details, we refer the interested reader to [23]).

Given an LTL_f formula φ, let $\mathrm{sub}(\varphi)$ denote the set of all its sub-formulas. The *sub-formula closure* of φ is the smallest set $\mathcal{C}(\varphi)$ that contains $\mathrm{sub}(\varphi)$; is closed under negation; and such that if $\varphi \mathcal{U} \psi \in \mathcal{C}(\varphi)$ then $\mathcal{X}(\varphi \mathcal{U} \psi) \in \mathcal{C}(\varphi)$ too. The formulas in $\mathcal{C}(\varphi)$ are sufficient to verify satisfiability of the formula φ.

A temporal model, which is formally defined only by the propositional variables satisfied at each timepoint, can also be characterised by the formulas (in $\mathcal{C}(\varphi)$) that it makes true at each point in time. Under this view, each timepoint can be associated to a *type*, which is a maximal consistent subset of $\mathcal{C}(\varphi)$ which also preserves the semantics of $\mathcal{U}$. Formally, a set $\tau \subseteq \mathcal{C}(\varphi)$ is a type iff the following conditions hold:

- for every $\psi \in \mathcal{C}(\varphi)$, $\psi \in \tau$ iff $\neg\psi \notin \tau$;
- for every $\psi_1 \wedge \psi_2 \in \mathcal{C}(\varphi)$, $\psi_1 \wedge \psi_2 \in \tau$ iff $\{\psi_1, \psi_2\} \subseteq \tau$;
- for every $\psi_1 \vee \psi_2 \in \mathcal{C}(\varphi)$, $\psi_1 \vee \psi_2 \in \tau$ iff $\{\psi_1, \psi_2\} \cap \tau \neq \emptyset$;
- for every $\psi_1 \mathcal{U} \psi_2 \in \mathcal{C}(\varphi)$, $\psi_1 \mathcal{U} \psi_2 \in \tau$ iff either $\psi_2 \in \tau$ or $\{\psi_1, \mathcal{X}(\psi_1 \mathcal{U} \psi_2)\} \subseteq \tau$.

Note that this latter condition uses the equivalence of $\mathcal{U}$ as described in the previous section.

As mentioned, types check for *local* consistency within a model, but one must still take into account the temporal semantics of the $\mathcal{X}$ operator. Two types τ_1 and τ_2 are *compatible* iff for every formula of the form $\mathcal{X}\psi$ in $\mathcal{C}(\varphi)$ it holds that $\mathcal{X}\psi \in \tau_1$ iff $\psi \in \tau_2$. Notice that this in particular means (by the maximality of types) that $\neg\mathcal{X}\psi \in \tau_1$ iff $\neg\psi \in \tau_2$. We say that a type is *terminal* if it does not contain any formula of the form $\mathcal{X}\psi$.

Definition 2 (φ-automaton). *The LTL_f formula φ defines the unlabelled NFA $\mathcal{A}_\varphi := (Q, \delta, I, F)$ where*

- *Q is the set of all types for φ;*
- *$\delta := \{(\tau_1, \tau_2) \in Q^2 \mid \tau_1$ and τ_2 are compatible$\}$;*
- *$I := \{\tau \in Q \mid \varphi \in \tau\}$; and*
- *$F := \{\tau \in Q \mid \tau$ is terminal$\}$.*

Intuitively, $\mathcal{A}_\varphi$ is a reachability graph, where an edge $(\tau_1, \tau_2) \in \delta$ states that one can have a temporal model with two successive timepoints satisfying the formulas in τ_1 and in τ_2, respectively. To construct a model of φ, we need to find a path that goes from an *initial* type (in I) to a *final* type (in F). Note that the condition for a type to belong to F—that is, the lack of formulas of the form $\mathcal{X}\psi$—means that it is safe to stop at that point, as no successive timepoint is needed to satisfy the constraints in the last observed type. Yet, one is not *required* to stop (other, longer models may exist as well).

Proposition 3 ([20]). *The LTL_f formula φ is satisfiable iff $\mathcal{A}_\varphi$ is non-empty.*

In Section 5, we present our approach which takes advantage of an ASP reasoner to decide satisfiability of LTL_f formulas by encoding (implicitly) the execution of a run of the automaton $\mathcal{A}_\varphi$. In a nutshell, the approach guesses the types satisfied at each timepoint, but in a way that satisfies all the constraints of the automaton. An important property of our encoding is that it allows for structure sharing, hence potentially reducing the encoding length of a formula. Before that, it is important to understand the main features of ASP.

4. Answer Set Programming

In this section we recall syntax and semantics of ASP. We refer the interested reader to the specific literature for a more detailed account [12, 13].

ASP syntax. A *variable* is a string starting with uppercase letter. A *constant* is an integer number or a string starting with lowercase letter. An *atom* is an expression of the form $p(t_1, \cdots, t_n)$ where p is a predicate of arity n and $t_1, \cdots, t_n$ are terms. An atom is *ground* if it contains no variable. A *literal* is a atom a or its negation *not* a where *not* denotes negation as failure. A literal l is *negative* if it is of the form *not* a, otherwise it is *positive*. The complement of positive (resp. negative) literal $l = a$ (resp. $l =$ *not* a), denoted by $\bar{l}$, is the literal *not* a (resp. a). A (normal) *rule* is an expression of the form $h \leftarrow b_1, \cdots, b_n$ where $b_1, \cdots, b_n$ is a conjunction of literals, referred to as body, $n \geq 0$, and h is an atom. All variables in a rule must occur in some positive literal of the body (i.e., are safe cfr. [24]). A *fact* is a rule with an empty body (i.e. $n = 0$). A *constraint* is a rule with an empty head that is a shorthand for rule $p \leftarrow b_1, \cdots, b_n, not\ p$, where p is a standard ground atom not occurring anywhere else. A *program* is a finite set of rules.

Stable Models Semantics. Given a program P, the *Herbrand Universe* U_P denotes the set of constants in P; the *Herbrand Base* B_P denotes the set of standard ground atoms that can be obtained from predicates in P and constants in U_P. Given a rule $r \in P$, $ground(r)$ denotes the set of possible rule instantiations that can be obtained by replacing variables in r with constants in U_P. The ground instantiation of the program P, denoted by $ground(P)$, is the union of ground instantiations of rules in P. An *interpretation* I is a subset of B_P. Given an interpretation I, a positive (resp. negative) literal l is true w.r.t. I, if $l \in I$ (resp. $\bar{l} \notin I$); it is false if $l \notin I$ (resp. $\bar{l} \in I$). A conjunction of literals is true w.r.t. I if all the literals are true w.r.t. I. An interpretation I is a *model* of P if for each $r \in ground(P)$, the head of r is true whenever the body of r is true. Given a program P and an interpretation I, the (Gelfond-Lifschitz) reduct [13], P^I, is the program obtained from $ground(P)$ by (i) removing all those rules having in the body a false negative literal w.r.t. I, and (ii) removing negative literals from the body of remaining rules. Given a program P and a model I, I is a *stable model* or *answer set* of P if there is no $I' \subset I$ such that I' is a model of P^I. P is coherent if it admits at least one answer set, otherwise it is incoherent.

5. ASP-based Satisfiability Checking

In this section we present an ASP-encoding that models the LTL_f satisfiability checking problem. In particular, we first describe how to encode the input and then we describe the satisfiability checking encoding.

5.1. Input Data

Given an input LTL_f formula φ, it is encoded in ASP by means of facts over predicates: $sub/1$, $isPhi/1$, $state/1$, $neg/2$, $next/2$, $and/3$, $or/3$, and $until/3$.

In particular, each distinct sub-formula has to be uniquely identified by an *id*, which is declared by facts of the form $sub(id)$. Atoms of the form $and(id, id_1, id_2)$ and $or(id, id_1, id_2)$, encode sub-formulas defined as the conjunction or disjunction, respectively of the two sub-formulas identified by id_1, and id_2. Atoms of the form $neg(id, id_1)$ models the negation of a sub-formula identified by id_1, by means of a sub-formula identified by *id*. We assume that both $neg(id_1, id_2)$ and $neg(id_2, id_1)$ are part of the input whenever id_1 is the negation of id_2. Atoms of the form $next(id, id_1)$ encode sub-formulas of the form $\mathcal{X} f_1$, where the sub-formula f_1 is identified by id_1, and atoms of the form $until(id, id_1, id_2)$ denote sub-formulas of the form $f_1 \mathcal{U} f_2$, where id_1, and id_2 are the identifiers of f_1 and f_2, respectively. The atom $isPhi(id)$ encodes the input formula φ, where *id* is the identifier of the outermost sub-formula. Runs of the automaton of length n are represented using atoms of the form $state(t)$, where $1 \leq t \leq n$. Given a formula φ, all these facts can be obtained by decomposing, bottom-up, φ into different sub-formulas, going from propositional variables to the outermost operator. Thus, an input formula φ can be encoded as a set of facts of linear size w.r.t. the number of subformulas of φ.

Example 4. *The formula* $\varphi = \neg(x) \wedge ((\mathcal{X}\, y)\, \mathcal{U}\, x)$ *is encoded as:*

$sub(x)$	$sub(y)$	$sub(1)$	$sub(2)$
$sub(3)$	$sub(4)$	$sub(5)$	$sub(6)$
$sub(7)$	$sub(8)$	$sub(9)$	$sub(10)$
$neg(1, x)$	$next(2, y)$	$until(3, 2, x)$	$and(4, 1, 3)$
$next(9, 3)$	$neg(5, y)$	$neg(6, 2)$	$neg(7, 3)$
$neg(8, 4)$	$neg(10, 9)$		

$sub(x)$ and $sub(y)$ encodes the two propositional variables x and y. Each $sub(i)$, $1 \leq i \leq 10$ instead encodes a composite sub-formula. Specifically:

- $sub(1)$, $neg(1, x)$ *encode the sub-formula* $f_1 : \neg x$
- $sub(2)$, $next(2, y)$ *encode the sub-formula* $f_2 : \mathcal{X}\, y$
- $sub(3)$, $until(3, 2, x)$ *encode the sub-formula* $f_3 : f_2\, \mathcal{U}\, x$
- $sub(4)$, $and(4, 1, 3)$ *encoded the sub-formula* $f_4 : f_1 \wedge f_3$
- $sub(5)$, $neg(5, y)$ *encode the sub-formula* $f_5 : \neg y$
- $sub(6)$, $neg(6, 2)$ *encode the sub-formula* $f_6 : \neg f_2$
- $sub(7)$, $neg(7, 3)$ *encode the sub-formula* $f_7 : \neg f_3$
- $sub(8)$, $neg(8, 4)$ *encode the sub-formula* $f_8 : \neg f_4$
- $sub(9)$, $next(9, 3)$ *encode the sub-formula* $f_9 : \mathcal{X}\, f_3$
- $sub(10)$, $neg(10, 9)$ *encode the sub-formula* $f_{10} : \neg f_9$

Note that f_4 is the outermost sub-formula after reconstructing the formula φ and so $isPhi(4)$ encodes φ. In addition, note that f_9 and f_{10} (along with most of the negations) are not strictly sub-formulas of φ but they must be included to guarantee the construction of a type from the set (φ) as defined in Section 3.

5.2. Encoding

In this section we present the encoding that models the satisfiability check of an LTL_f formula. Notably, this encoding is highly general and does not depend on the specific formula being evaluated. Therefore, given any formula φ and its corresponding encoding as ASP facts (as outlined in the previous section), the encoding can be directly reused to verify the satisfiability of φ. In order to better present the proposed encoding, in what follows we refer to sub-formulas by means of their identifier. The proposed encoding exploits the well-known Guess&Check paradigm [25] in which the Guess part tries to guess maximal subset of formulae, by assigning to each state either a formula F or its negation $F1$.

$$g_1 : \quad type(S, F) \leftarrow \quad state(S),\ sub(F),\ neg(F1, F),\ not\, type(S, F1)$$
$$g_2 : \quad type(S, F1) \leftarrow \quad state(S),\ sub(F),\ neg(F1, F),\ not\, type(S, F)$$

Next, the definition of the different elements of the automaton is verified by means of normal rules and constraints.

Initial State. To ensure that state 1 (the first state in the run) contains the the outermost sub-formula denoted by $isPhi(F)$ we use the rule:

$$type(1, F) \leftarrow isPhi(F).$$

Final and active states. A final state, which contains no $\mathcal{X}$-formula, should be observed within the first n states of the run. This is enforced through the rules:

$$r_1 : \quad hasNext(S) \leftarrow \quad type(S, F), next(F, F1)$$
$$r_2 : \quad final(S) \leftarrow \quad state(S),\ not\, hasNext(S)$$
$$r_3 : \quad found\_final \leftarrow \quad final(L)$$
$$r_4 : \quad \leftarrow \quad not\ found\_final$$
$$r_5 : \quad active(S) \leftarrow \quad state(S), final(L), L >= S$$

Rule r_1 defines atoms of the form $hasNext(S)$. These denote all those states containing at least one sub-formula, $F : \mathcal{X}F1$. Rule r_2 encodes the final states as atoms of the form $final(S)$. A state S is final if does not contain sub-formula of the form $\mathcal{X}F1$ (i.e. *not* $hasNext(S)$). Finally, rules r_3 and r_4 impose that at least one final state exists. Moreover, since the proposed encoding search for a run of at most n states, then the required conditions must be verified for all the states up to the actual final one. Thus, we mark as *active*, by means of rule r_5 that derives an atom $active(S)$, for each state S that is less or equal than the actual last state L (i.e. $final(L)$).

Type Condition. We must still impose that each state $1 \leq S \leq \ell$ (where ℓ is the first observed final state) is a type. A state S is a type iff the following conditions hold for all formulas in (φ)

1. S contains the formula F if and only if S does not contain $\neg F$
2. S contains the formula $X \wedge Y$ if and only if S contains both X and Y
3. S contains the formula $X \vee Y$ if and only if S contains at least one between X and Y
4. S contains the formula $F = X\mathcal{U}Y$ if and only if at least one of the following conditions holds:
   a) S contains Y or
   b) S contains both X and $\mathcal{X}F$

Condition 1 is ensured by the guess rules g_1, g_2 that for each state S and each sub-formula F assign either F or $F1 : \neg F$ to S.

Condition 2 is ensured by the constraints:

$$\begin{aligned} a_1 &: \leftarrow type(S,F),\ and(F,X,Y),\ not\ type(S,X),\ active(S) \\ a_2 &: \leftarrow type(S,F),\ and(F,X,Y),\ not\ type(S,Y),\ active(S) \end{aligned}$$

$$a_3 : \leftarrow type(S,X),\ type(S,Y),\ and(F,X,Y),\ not\ type(S,F), active(S)$$

Intuitively, for each formula $F : X \wedge Y$, constraint a_1 (resp. a_2) imposes that it is not possible that a state $S \leq \ell$ contains F and does not contain X (resp. Y). Constraint a_3 discards those answer sets in which a state $S \leq \ell$ contains both X and Y, and does not contain F.

Condition 3 is ensured by the constraints:

$$o_1 : \leftarrow type(S,F),\ or(F,X,Y),\ not\ type(S,X),\ not\ type(S,Y), active(S)$$

$$\begin{aligned} o_2 &: \leftarrow type(S,X),\ or(F,X,Y),\ not\ type(S,F),\ active(S) \\ o_3 &: \leftarrow type(S,Y),\ or(F,X,Y),\ not\ type(S,F),\ active(S) \end{aligned}$$

Here, for each formula $F : X \vee Y$, the constraint o_1 expresses that it is not possible for a state $S \leq \ell$ to contain F, but contain neither X nor Y. Constraint o_2 (resp. o_3) imposes that it is not possible that a state $S \leq \ell$ contains X (resp. Y) and does not contain F.

In order to verify the conditions 4a and 4b the predicate $sat\_until/4$ is used. Atoms of the form $sat\_until(S,F,X,Y)$ denote that at least one between conditions 4a and 4b holds at state S for the formula $F : X\mathcal{U}Y$ and so F can be added to the state S. These atoms can be derived by the rules:

$$\begin{aligned} su_1 &: sat\_until(S,F,X,Y) \leftarrow until(F,X,Y),\ type(S,Y) \\ su_2 &: sat\_until(S,F,X,Y) \leftarrow until(F,X,Y),\ type(S,X), next(F1,F),\ type(S,F1) \end{aligned}$$

Roughly, for each formula $F : X\mathcal{U}Y$, if there exists a state S containing Y then condition 4a is satisfied; thus, rule su_1 derives $sat\_until(S,F,X,Y)$. On the other hand, if there exists a state S containing both X and a formula $F1 : \mathcal{X}F$ then condition 4b is satisfied and so rule su_2 derives $sat\_until(S,F,X,Y)$. Thus, condition 4 is ensured by the constraints:

$$\begin{aligned} u_1 &: \leftarrow type(S,F),\ until(F,X,Y),\ not\ sat\_until(S,F,X,Y), active(S) \\ u_2 &: \leftarrow not\ type(S,F),\ sat\_until(S,F,X,Y),\ active(S) \end{aligned}$$

Intuitively, for each formula $F : X\mathcal{U}Y$, constraint u_1 imposes that a state $S \leq \ell$ cannot contain F if F cannot be added to S, while, constraint u_2, imposes that it is not possible that F can be added to the state S but S does not contain F.

Connectedness. The guessed automaton is connected if the following conditions hold:

1. a state S contains a formula $F1 : \mathscr{X} F$ iff S is not the final state and the state $S+1$ contains F
2. a state S contains a formula $F1 : \neg(\mathscr{X} F)$ iff S is not the final state and the state $S+1$ contains $\neg F$

Condition 1 is ensured by the constraints:

$$c_1 : \leftarrow type(S,F),\ next(F,F1),\ S1 = S+1,\ active(S1), not\, type(S1,F1),\ active(S)$$

$$c_2 : \leftarrow not\, type(S,F),\ next(F,F1),\ S1 = S+1,\ active(S1),\ type(S1,F1),\ active(S)$$

Basically, for every formula $F : \mathscr{X} F1$, c_1 imposes that it is not possible for a state S to contain F if $F1$ is not assigned at state $S+1$. On the other hand, c_2 imposes that it is not possible that there exists a state $S+1$ containing a formula $F1$ such that the state S does not contain the formula $F : \mathscr{X} F1$. Condition 2 is ensured by the constraints:

$$\begin{aligned} c_3 : \leftarrow\ & neg(F,F1),\ next(F1,F2),\ neg(F3,F2), type(S,F), \\ & S1 = S+1,\ active(S1),\ not\, type(S1,F3),\ active(S) \end{aligned}$$

$$\begin{aligned} c_4 : \leftarrow\ & neg(F,F1),\ next(F1,F2),\ neg(F3,F2), type(S1,F3), \\ & S1 = S+1,\ active(S),\ not\, type(S,F),\ active(S) \end{aligned}$$

Intuitively, let $F : \neg F1$, $F1 : \mathscr{X} F2$, and $F3 : \neg F2$, c_3 imposes that it is not possible that there exists a state S containing the formula F (i.e. $\neg(\mathscr{X} F2)$), and the formula $F3$ (i.e. $\neg F2$) is not assigned at state $S+1$. On the other hand, c_4 imposes that it is not possible that there exists a state $S+1$ containing the formula $F3$ (i.e. $\neg F2$), and the state S does not contain the formula F (i.e. $\neg(\mathscr{X} F2)$).

Correctness. Given the LTL$_f$ formula φ, let $F(\varphi, n)$ denotes the input obtained from φ as described in Section 5.1, where n indicates the maximum number of states in the run to be considered; and let Π denote the ASP program described in Section 5.2.

Proposition 5. *An LTL$_f$ formula φ is satisfiable iff there exists an integer $n > 0$ such that $\Pi \cup F(\varphi, n)$ is coherent.*

Intuitively, the results follows from Proposition 3 observing that the choice rule in Π guesses a type for each state, and the constraints in Π enforce the remaining conditions of Definition 2.

Implementation. By exploiting the encoding proposed in this section, it is possible to construct an Algorithm for deciding the satisfiability of an LTL$_f$ formula. More precisely, Algorithm 1 reports the pseudo-code of such a procedure.

Intuitively, starting with $n = 1$ it is possible to incremetally search for an accepting run that uses at most n timepoints (each tagged with a state). Thus, at each iteration if $P = \Pi \cup F(\varphi, n)$ is coherent

Algorithm 1 Decide-LTL$_f$

```
   Input : An LTL_f formula φ
1  begin
2      n = 1
3      while n ≤ 2^(‖φ‖+1) do
4          P := Π ∪ F(φ, n)
5          if P is coherent then
6              return SAT
7          n := n + 1
8      return UNSAT
```

LTL$_f$ pattern	#Inst	Formula
Alternate Response ($AR(k)$)	40	$\neg(\top\ \mathcal{U}\ ((\neg x_0) \vee (\mathcal{X}\ ((\neg x_0)\ \mathcal{U}\ (\bigvee_{i=1}^{k} x_i)))))$
Chain Response ($CR(k)$)	40	$\neg(\top\ \mathcal{U}\ ((\neg x_0) \vee (\mathcal{X}\ (\bigvee_{i=1}^{k} x_i))))$
Response ($R(k)$)	40	$\neg((\neg x_0) \wedge (\top\ \mathcal{U}\ (\bigvee_{i=1}^{k} x_i)))$
RespondedExistence ($RE(k)$)	40	$(\neg(\top\ \mathcal{U}\ x_0)) \vee (\top\ \mathcal{U}\ (\bigvee_{i=1}^{k} x_i))$
$E(k)$	40	$\bigwedge_{i=1}^{k} \top\ \mathcal{U}\ x_i$
$S(k)$	40	$\bigwedge_{i=1}^{k} \top\ \mathcal{U}\ \neg x_i$
$EL(k, z)$	80	$(\bigwedge_{i=1}^{k} \top\ \mathcal{U}\ \neg x_i) \wedge (\mathcal{X}\ (\mathcal{X}\ (\cdots (\mathcal{X}\ x_0) \cdots)))$, where z is number of nested "next" operators.

Table 1: Considered LTL$_f$-specific benchmarks

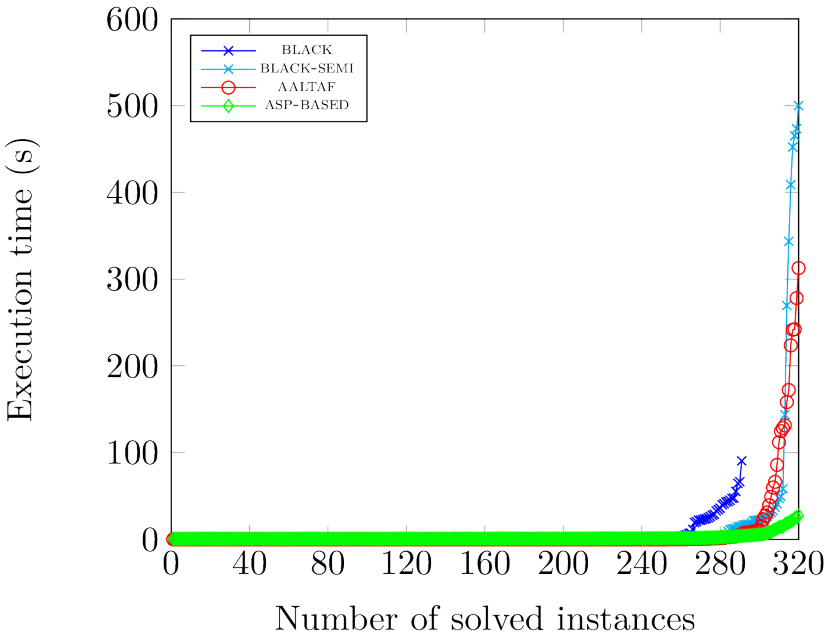
then we found a witness of satisfiability of φ. Conversely, if we were not able to found such a witness with n up to $2^{\|\varphi\|+1}$ where $\|\varphi\|$ denotes the number of symbols, excluding parenthesis, appearing in φ, then φ is unsatisfiability. This upper bound is an easy consequence of the definition of a type. Indeed, from Section 3 it can be seen that the number of types is bounded by $2^{\|\varphi\|+1}$ and that in the worst case, a successful run from an initial to a final type will traverse all types once. In practice, many state-of-the-art proposals [22, 26, 27] operate following a similar strategy (i.e. incremental expanding horizon), and are effective whenever the formula admits temporal models with a reasonably short length. We implemented such an approach into a Python prototype that takes as input a formula φ, at each iteration computes $F(\varphi, n)$, and uses the ASP solver *clingo* [28] for verifying the coherence $\Pi \cup F(\varphi, n)$.

6. Experiments

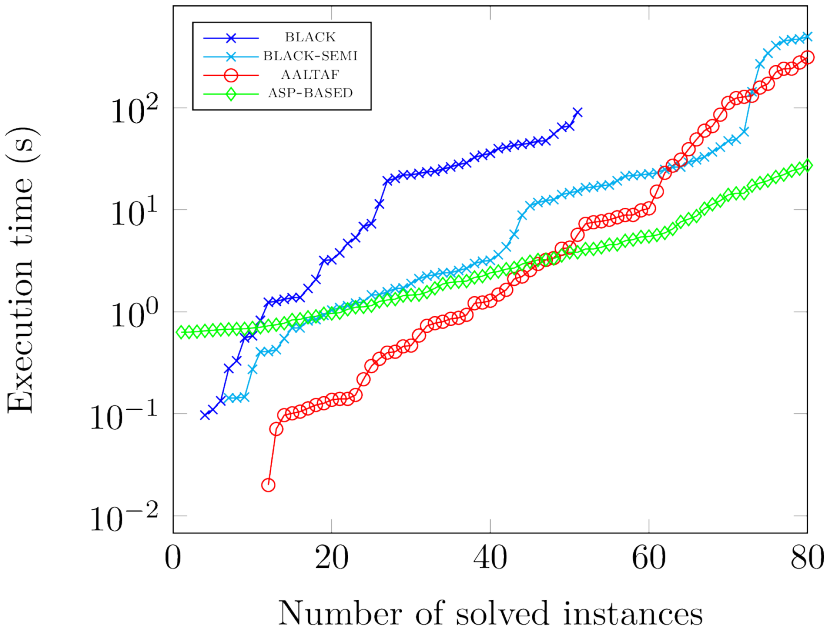
We now present an empirical evaluation aimed at investigating the viability of the approach which we have proposed in practical settings.

Benchmark suite. In this experiment, we used a benchmark suite consisting of LTL$_f$-specific benchmarks commonly used in the literature to evaluate systems for LTL$_f$ satisfiability checking [22]. Specifically, each benchmark corresponds to a distinct LTL$_f$ pattern formula as listed in Table 1. It is worth noting that these formulas are expressed in the native syntax of LTL$_f$, which differs from the extended syntax in [22], but the formulas are equivalent.

The first four, namely Alternate Response, Chain Response, Response, and RespondedExistence, are LTL$_f$ encodings of Declare constraints [29]. Declare is the *de facto* standard in declarative process mining, thus these can be considered as representative of a concrete application of LTL$_f$. The remaining three, instead, are specific patterns of formulas suitable for a scalability analysis on a synthetic benchmark. Concerning the instances, for each pattern we generated formulas with increasing values of k, from 5 to 200. For benchmark $EL(k, z)$ we considered two values of z for each value of k: $k/2$ and $(3 \times k)/2$, respectively. As a result we obtained a benchmark suite comprising 320 formulas.



(a) Overall Comparison



(b) Comparison on $EL(k, z)$

Figure 1: Systems Comparison

Compared methods. For our evaluation we consider four different methods: (i) our ASP-BASED[1] approach, implemented by running the ASP solver *clingo* [28]; (ii) the AALTAF system implementing (Conflict-Driven LTL$_f$ Satisfiability Checking) [22]; (iii) the *black* system [27], in two different settings: (a) the standard version denoted as BLACK; (b) the semi-decisional one denoted as BLACK-SEMI.

Hardware and software resources. All the experiments were executed on a machine equipped with Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz, running macOS 14.5 (23F79) (Darwin Kernel Version 23.5.0). Memory and time were limited to 2GB and 600s respectively.

[1]https://github.com/MazzottaG/LTLToAutomata.git

Results. The results obtained are summarized by the cactus plots shown in Figure 1, which report the execution time for each system ordered from best to worst. Specifically, in a cactus plot, instances are sorted by their execution time, and a point (i, j) indicates that a solver can solve the i-th instance within j seconds.

As it is evident from Figure (1a), the ASP-BASED approach is the fastest system among compared ones, solving every instance within few seconds. Specifically, ASP-BASED and the other compared systems can solve all instances of the benchmarks referring to Declare patterns (i.e., $AR(k)$, $CR(k)$, $R(k)$, and $RE(k)$) almost instantaneously (less than 0.5 seconds per instance) even for larger values of k (up to 1000), demonstrating a significant positive outcome for declarative process mining, which is an important application for LTL_f reasoning.

Regarding the scalability of our approach on synthetic formulas, we note that for the first two benchmarks (i.e., $E(k)$ and $S(k)$), all the compared systems are instantaneous, similarly to the Declare patterns. However, the last benchmark, $EL(k, z)$, proved to be more challenging than the previous ones. Figure (1b) reports the execution times of the compared systems on such benchmarks. Our approach exhibits a negligible overhead on very simple instances (less than 0.5 seconds) due to the Python interpreter. The strength of the ASP-BASED approach is clear, in the evaluation of hard LTL_f formulas, where the execution is orders of magnitude smaller than compared systems.

7. Related Work

The primary task in LTL_f is satisfiability checking, for which various approaches have been proposed. Some of these rely on translation of LTL_f formulas to symbolic Deterministic Finite Automata (DFA)[10, 30]. These translations convert LTL_f semantics into First Order Logic and Monadic Second Order logic, and then uses tools like MONA [31] to obtain the symbolic DFA. However, these approaches require the full materialization of the automata, which is exponential w.r.t. the size of the input formula. On the contrary, our approach incrementally searches for an accepting run of the underlying automata without materializing it.

Actual implementations [22, 26, 27] are based on translations to SAT [32, 33]. The AALTAF system [22] uses SAT-solving to create a transition system for an LTL_f formula, turning satisfiability checking into a path-search problem. Additionally, the introduced CDLSC (Conflict-Driven LTLf Satisfiability Checking) algorithm uses SAT solver data for both satisfiable and unsatisfiable results. Another strategy for transforming an LTL_f formula into propositional formulas (cfr. [26]) can be used to verify the existence of a temporal model of length at most n, until the theoretical upper bound for n is reached. The approach used in BLACK [27] encodes the one-pass and tree-shaped tableau by Reynolds [34]. In this approach the underlying tableau tree is built in a breadth-first way, by means of Boolean formulae encoding the possible tableau branches up to a given depth k, which is increased at each step. All the SAT-based approaches generate a specific SAT encoding for each formula. Conversely our approach provides a general uniform ASP encoding based on φ-automata that can be used for evaluating any LTL_f formula, that is also efficient.

Thanks to its declarative nature, ASP found also various applications in the field of LTL_f satisfiability. Among such works, an ASP-based method for reasoning over weighted LTL_f formulas has been proposed [35]. In this approach a satisfiability checker [26] is used to verify the satisfiability of the input formula, and then an ASP solver searches for optimal models. While our first-order constraints resemble the variable-free ones in their model-checker program, our goal differs as we directly determine the satisfiability of an LTL_f formula, whereas their approach relies on an initial satisfiability checker. Furthermore, ASP has been applied to compute minimal unsatisfiable cores of LTL_f formulae [36, 37]. The approach proposed in [38, 39] leverages algorithms for minimal unsatisfiable subprogram enumeration [40] applied to an ASP encoding for bounded satisfiability [41].

ASP has been also used for bounded model checking [42], where it is demonstrated that a 1-safe Petri net and its behavioral requirements (in LTL) can be translated into a logic program, so bounded model checking amounts to computing its stable models. Moreover, recent applications of ASP to declarative

process mining have been proposed [43, 44, 45]. However, in these works the authors do not address the problem of satisfiabilty of LTL_f but targeted related tasks such as Conformance Checking that indeed falls in lower complexity classes.

The extensions of ASP for modeling temporal properties are also related. The system *telingo* [46] extends CLINGO by adding to ASP future and past temporal operators and incrementally solving the corresponding temporal logic programs using CLINGO's multi-shot solving interface. Roughly, LTL relates to SAT in the same way the temporal extension supported by *telingo* relates to ASP. Thus, the formalism implemented by *telingo* is more expressive than LTL_f, and so, one could even envision modeling our task in *telingo*. However, the goal of this paper is to provide an efficient encoding of LTL_f in plain ASP. Finally, we also mention methods for modeling temporal constraints in ASP that are based on alternating automata [47].

8. Conclusion

A core challenge in the LTL_f literature is the development of methods for satisfiability checking. This area is being dominated by logic-based approaches [22, 26, 27], which are based on propositional SAT solving.

This paper presents a fresh perspective for tackling this reasoning task, by presenting an automata-based approach implemented in ASP. The resulting approach is based on a more direct modeling of the problem that does not require the input formula to be highly pre-processed or rendered a normal form; moreover, it gives as a byproduct a meaningful witness of the satisfiability outcome. A first experimental analysis demonstrates that our approach is also viable in practice, delivering good performance, comparable to the state-of-the-art CDSLC approach on declarative process mining benchmarks.

Our proposal paves the way for new research and development opportunities in the field. On the one hand, the generation of witnesses of satisfiability in our approach lays a foundation for the development of explainability techniques in LTL_f reasoning. On the other hand, this initial encoding in ASP presents potential for optimizations at both the encoding and system levels. Indeed, the usage of novel grounding-less evaluation techniques [48, 49, 50] could beneficial for obtaining even better performances.

Acknowledgments

This work was partially supported by MISE under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, and MUR under projects: PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006 and H45E21000210001. Finally, we mention that Francesco Ricca is member of Gruppo Nazionale Calcolo Scentifico of INDAM (GNCS-INDAM).

References

[1] D. Aineto, R. De Benedictis, M. Maratea, M. Mittelmann, G. Monaco, E. Scala, L. Serafini, I. Serina, F. Spegni, E. Tosello, A. Umbrico, M. Vallati (Eds.), Proceedings of the International Workshop on Artificial Intelligence for Climate Change, the Italian workshop on Planning and Scheduling, the RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and the Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (AI4CC-IPS-RCRA-SPIRIT 2024), co-located with 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), CEUR Workshop Proceedings, CEUR-WS.org, 2024.

[2] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013, pp. 854–860.

[3] G. De Giacomo, F. M. Maggi, A. Marrella, S. Sardiña, Computing trace alignment against declarative process models through planning, in: ICAPS, 2016, pp. 367–375.
[4] G. De Giacomo, M. Y. Vardi, Automata-theoretic approach to planning for temporally extended goals, in: ECP, volume 1809 of *LNCS*, 1999, pp. 226–238.
[5] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. Artif. Intell. 22 (1998) 5–27.
[6] D. Calvanese, G. De Giacomo, M. Y. Vardi, Reasoning about actions and planning in LTL action theories, in: KR, 2002, pp. 593–602.
[7] G. De Giacomo, M. Dumas, F. M. Maggi, M. Montali, Declarative process modeling in BPMN, in: CAiSE, volume 9097 of *LNCS*, 2015, pp. 84–100.
[8] A. Cecconi, G. De Giacomo, C. D. Ciccio, F. M. Maggi, J. Mendling, Measuring the interestingness of temporal logic behavioral specifications in process mining, Inf. Syst. 107 (2022) 101920.
[9] Y. Tsay, M. Y. Vardi, From linear temporal logics to Büchi automata: The early and simple principle, in: Model Checking, Synthesis, and Learning, volume 13030 of *LNCS*, 2021, pp. 8–40.
[10] S. Zhu, G. Pu, M. Y. Vardi, First-order vs. second-order encodings for \textsc ltl_f -to-automata translation, in: Theory and Applications of Models of Computation - 15th Annual Conference, TAMC 2019, Kitakyushu, Japan, April 13-16, 2019, Proceedings, volume 11436 of *LNCS*, 2019, pp. 684–705.
[11] Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2021.
[12] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Com. ACM 54 (2011) 92–103.
[13] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Comput. 9 (1991) 365–386.
[14] Y. Lierler, M. Maratea, F. Ricca, Systems, engineering environments, and competitions, AI Magazine 37 (2016) 45–52.
[15] M. Alviano, C. Dodaro, M. Maratea, Nurse (re)scheduling via answer set programming, Intelligenza Artificiale 12 (2018) 109–124.
[16] G. Grasso, S. Iiritano, N. Leone, V. Lio, F. Ricca, F. Scalise, An asp-based system for team-building in the gioia-tauro seaport, in: PADL, volume 5937 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 40–42.
[17] V. Barbara, M. Guarascio, N. Leone, G. Manco, A. Quarta, F. Ricca, E. Ritacco, Neuro-symbolic AI for compliance checking of electrical control panels, Theory Pract. Log. Program. 23 (2023) 748–764.
[18] A. Pnueli, The temporal logic of programs, in: Proc. of 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 46–57.
[19] J. R. Büchi, On a Decision Method in Restricted Second Order Arithmetic, New York, NY, 1990, pp. 425–435.
[20] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, 2013, pp. 854–860.
[21] C. Y. Lee, Representation of switching circuits by binary-decision programs, The Bell System Technical Journal 38 (1959) 985–999.
[22] J. Li, G. Pu, Y. Zhang, M. Y. Vardi, K. Y. Rozier, SAT-based explicit LTLf satisfiability checking, Artificial Intelligence 289 (2020) 103369.
[23] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, 1979.
[24] S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Surveys in computer science, 1990.
[25] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (1995) 289–323.
[26] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623.

[27] L. Geatti, N. Gigante, A. Montanari, A sat-based encoding of the one-pass and tree-shaped tableau system for LTL, in: Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings, volume 11714 of *LNCS*, 2019, pp. 3–20.
[28] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, TPLP 19 (2019) 27–82.
[29] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for loosely-structured processes, in: IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2007, pp. 287–300.
[30] S. Zhu, L. M. Tabajara, J. Li, G. Pu, M. Y. Vardi, Symbolic ltlf synthesis, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, 2017, pp. 1362–1369.
[31] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, A. Sandholm, Mona: Monadic second-order logic in practice, in: Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings, volume 1019 of *LNCS*, 1995, pp. 89–110.
[32] J. Li, G. Pu, L. Zhang, M. Y. Vardi, J. He, Accelerating LTL satisfiability checking by SAT solvers, J. Log. Comput. 28 (2018) 1011–1030.
[33] K. Y. Rozier, M. Y. Vardi, A multi-encoding approach for LTL symbolic satisfiability checking, in: FM, volume 6664 of *LNCS*, 2011, pp. 417–431.
[34] M. Reynolds, A new rule for LTL tableaux, in: Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016, volume 226 of *EPTCS*, 2016, pp. 287–301.
[35] C. Dodaro, V. Fionda, G. Greco, LTL on weighted finite traces: Formal foundations and algorithms, in: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, 2022, pp. 2606–2612.
[36] T. Niu, S. Xiao, X. Zhang, J. Li, Y. Huang, J. Shi, Computing minimal unsatisfiable core for ltl over finite traces, Journal of Logic and Computation 34 (2024) 1274–1294.
[37] M. Roveri, C. D. Ciccio, C. D. Francescomarino, C. Ghidini, Computing unsatisfiable cores for ltlf specifications, J. Artif. Intell. Res. 80 (2024) 517–558.
[38] A. Ielo, G. Mazzotta, R. Peñaloza, F. Ricca, Towards asp-based minimal unsatisfiable cores enumeration for ltlf, in: OVERLAY, volume To appear of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024.
[39] A. Ielo, G. Mazzotta, R. Peñaloza, F. Ricca, Enumerating minimal unsatisfiable cores of ltlf formulas, CoRR abs/2409.09485 (2024).
[40] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, ASP and subset minimality: Enumeration, cautious reasoning and muses, Artif. Intell. 320 (2023) 103931.
[41] V. Fionda, A. Ielo, F. Ricca, Ltlf2asp: Ltlf bounded satisfiability in ASP, in: LPNMR, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 373–386.
[42] K. Heljanko, I. Niemelä, Bounded LTL model checking with stable models, TPLP 3 (2003) 519–550.
[43] F. Chiariello, F. M. Maggi, F. Patrizi, Asp-based declarative process mining, in: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, AAAI Press, 2022, pp. 5539–5547. URL: https://doi.org/10.1609/aaai.v36i5.20493. doi:10.1609/AAAI.V36I5.20493.
[44] I. Kuhlmann, C. Corea, J. Grant, Non-automata based conformance checking of declarative process specifications based on ASP, in: J. D. Weerdt, L. Pufahl (Eds.), Business Process Management Workshops - BPM 2023 International Workshops, Utrecht, The Netherlands, September 11-15, 2023, Revised Selected Papers, volume 492 of *Lecture Notes in Business Information Processing*, Springer, 2023, pp. 396–408. URL: https://doi.org/10.1007/978-3-031-50974-2_30. doi:10.1007/978-3-031-50974-2_30.
[45] F. Chiariello, V. Fionda, A. Ielo, F. Ricca, A direct ASP encoding for declare, in: PADL, volume 14512 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 116–133.
[46] P. Cabalar, R. Kaminski, P. Morkisch, T. Schaub, telingo = ASP + time, in: Logic Programming and

Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings, volume 11481 of *LNCS*, 2019, pp. 256–269.

[47] P. Cabalar, M. Diéguez, S. Hahn, T. Schaub, Automata for dynamic answer set solving: Preliminary report, in: Proceedings of the International Conference on Logic Programming 2021 Workshops co-located with the 37th International Conference on Logic Programming (ICLP 2021), Porto, Portugal (virtual), September 20th-21st, 2021, volume 2970 of *CEUR Workshop Proceedings*, 2021.

[48] G. Mazzotta, F. Ricca, C. Dodaro, Compilation of aggregates in ASP systems, in: AAAI, AAAI Press, 2022, pp. 5834–5841.

[49] C. Dodaro, G. Mazzotta, F. Ricca, Compilation of tight ASP programs, in: ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023), volume 372 of *Frontiers in Artificial Intelligence and Applications*, 2023, pp. 557–564.

[50] C. Dodaro, G. Mazzotta, F. Ricca, Blending Grounding and Compilation for Efficient ASP Solving, in: Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, 2024, pp. 317–328. URL: https://doi.org/10.24963/kr.2024/30. doi:10.24963/kr.2024/30.