# A Declarative Framework for Temporal Reasoning in Green-Aware Applications

Giuseppe De Giacomo[1], Valeria Fionda[2], Nicola Gigante[3], Antonio Ielo[2], Francesco Ricca[2] and Alessandra Russo[4]

[1]*University of Oxford, UK*
[2]*University of Calabria, IT*
[3]*Free University of Bozen-Bolzano, IT*
[4]*Imperial College London, UK*

**Abstract**

This paper presents a novel framework that integrates Linear Temporal Logic over Finite Traces ($\text{LTL}_\text{f}$) with the expressive capabilities of Answer Set Programming (ASP). Combining these two powerful formalisms enables efficient reasoning in complex, temporally dynamic, and knowledge-rich environments, making it well-suited for green-aware applications that demand sustainable resource management and reduced environmental impact. The proposed framework supports the querying of evolving ASP knowledge bases, allowing both brave and cautious inferences aligned with temporal constraints. We detail the foundational principles of the framework and explore its potential applications in environmentally conscious scenarios.

**Keywords**

Linear Temporal Logic, Answer Set Programming, Green-Aware reasoning

## 1. Introduction

In recent years, the growing emphasis on sustainability and environmental consciousness has stimulated interest in developing green-aware applications [1]. These systems require advanced reasoning capabilities to manage resources efficiently while satisfying temporal and logical constraints. Linear Temporal Logic (LTL) [2] has been a cornerstone in Computer Science for reasoning about sequences of events and has found application in many domains [3, 4, 5]. The variant LTL over finite traces ($\text{LTL}_\text{f}$) [6, 7, 8] has proven particularly effective in scenarios where operations are naturally representable by finite sequences of actions [9, 10]. However, due to its propositional nature, $\text{LTL}_\text{f}$ is cumbersome when dealing with complex, knowledge-intensive domains. For instance, modeling dynamic systems such as networks of sensors or smart buildings often involves reasoning about properties like graph connectivity or node-to-node reachability, or even more complex properties such as guaranteeing a cover or domination of a subnetwork, which are challenging to express solely within $\text{LTL}_\text{f}$.

**Example 1.** *In an urban traffic network, roads (edges) may be closed for maintenance, environmental reasons, or congestion control. However, at each point in time, it must be ensured that traffic flow remains functional while reducing emissions by dynamically shutting down certain routes, that is ensuring that all locations (nodes) remain connected despite closing certain roads. This involve reasoning about relationships between nodes and edges over time. Encoding such relationships directly in $LTL_f$ requires extensive formulae that become unmanageable, inefficient, and unintuitive, especially as the network complexity increases.*

To address these limitations, we propose a new framework, called $\text{LTL}_\text{f}^\text{ASP}$, that extends $\text{LTL}_\text{f}$ by integrating it with Answer Set Programming (ASP) [11, 12]. ASP is a declarative programming paradigm well-suited in combinatorial optimization and knowledge representation and reasoning. Our framework

*1st Workshop on Green-Aware Artificial Intelligence, 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), November 25–28, 2024, Bolzano, Italy*

giuseppe.degiacomo@cs.ox.ac.uk (G. D. Giacomo); valeria.fionda@unical.it (V. Fionda); Nicola.Gigante@unibz.it (N. Gigante); antonio.ielo@unical.it (A. Ielo); francesco.ricca@unical.it (F. Ricca); a.russo@imperial.ac.uk (A. Russo)
0000-0001-9680-7658 (G. D. Giacomo); 0000-0002-7018-1324 (V. Fionda); 0000-0002-2254-4821 (N. Gigante); 0009-0006-9644-7975 (A. Ielo); 0000-0001-8218-3178 (F. Ricca); 0000-0002-3318-8711 (A. Russo)

provides a way to meld ASP's rich, expressive language with $\mathrm{LTL_f}$ temporal reasoning features, easing the development of sophisticated green-aware applications. Indeed, ASP enables reasoning about individual states induced by the temporal specification in a high-level, declarative, expressive and concise way.

**Example 2.** *Using $LTL_f^{ASP}$, the traffic network problem can be encoded more naturally and efficiently. While $LTL_f$ handles the temporal aspects of the problem (e.g., ensuring specific routes are open or closed at different times), ASP can handle graph properties such as connectivity and connectedness between nodes as the system evolves. This avoids the need for propositional encodings in $LTL_f$ alone, making the encoding significantly more concise and manageable, even as the network complexity increases.*

This paper defines the $\mathrm{LTL_f^{ASP}}$ framework and instantiates it in several green-aware application scenarios. The rest of the paper is organized as follows. Section 2 reviews related literature. Section 3 provides a detailed overview of Linear Temporal Logic over Finite Traces and Answer Set Programming. In Section 4, we formally define the syntax and semantics of the $\mathrm{LTL_f^{ASP}}$ framework, explaining how it integrates the temporal reasoning capabilities of $\mathrm{LTL_f}$ with the declarative power of ASP. Section 5 illustrates the application of $\mathrm{LTL_f^{ASP}}$ in some green-aware domains. Finally, Section 6 concludes the paper.

## 2. Related Work

Linear Temporal Logic (LTL) has been extensively used for reasoning about event sequences in various domains, such as planning, verification, and process management [2, 3, 4]. Its finite variant, $\mathrm{LTL_f}$, is particularly suitable for scenarios bounded by a finite time horizon, which is crucial in many practical applications [6, 7]. While $\mathrm{LTL_f}$ is a powerful formalism, its propositional nature limits the ability to capture complex temporal properties in knowledge- and data-intensive domains. To overcome these limitations, extensions such as $\mathrm{LTL_f^{MT}}$ ($\mathrm{LTL_f}$ Modulo Theories) [13] have been developed, which integrate additional theories like linear integer arithmetic and uninterpreted functions to increase expressiveness [14, 15, 16], as well as atomic constraints over concrete domains [17]. Similarly, first-order extensions of $\mathrm{LTL_f}$ enhance its knowledge representation capabilities by allowing richer interactions with domain-specific data [18, 19].

The integration of logic programming with temporal reasoning has been explored to address limitations in modeling dynamic systems. Notably, Eiter et al. [20] proposed combining a declarative planning language with Answer Set Programming (ASP), enhancing expressiveness and problem-solving capabilities in dynamic domains. Although successful, these approaches do not fully leverage ASP *as a theory* within temporal logic.

From the Answer Set Programming perspective, the strand of research on Temporal Answer Set Programming (TASP) [21] extends standard ASP with LTL-like temporal constructs. Our approach is not an extension of ASP with temporal constructs, but rather the *embedding* of ASP into $\mathrm{LTL_f}$ reasoning, facilitating complex queries over execution traces.

Another strand of related research is stream reasoning, which deals with continuous query answering over data streams [22], which often relies on logic programming as its foundation. ASP-based stream reasoning systems typically focus on window-based approaches, processing segments of data streams to perform temporal reasoning. Our approach reasons over finite traces, making it ideal for green-aware applications requiring full temporal context [23].

There have been in the literature also some proposals for ASP *Modulo Theories* [24] that extends ASP by integrating external theories, akin to how SAT is extended by Satisfiability Modulo Theories (SMT) [25, 26]. This extension allows ASP to interface with additional reasoning modules. Our departure point is entirely different since we aim to use ASP as a theory within $\mathrm{LTL_f}$, enabling advanced reasoning capabilities tailored specifically for knowledge-rich, temporal environments. Thus, our proposal relates more to first-order extensions of $\mathrm{LTL_f}$ and $\mathrm{LTL_f^{MT}}$ than TASP.

Green-aware reasoning has gained significant attention recently as sustainability has become a critical concern. Some proposals that leverage logic-based approaches have been used to model and manage energy consumption, optimize resource allocation, and reduce environmental impacts in various applications. A recent proposal [27] introduced a formal framework using temporal logic and simulation tools to model and verify the energy management of buildings. In [28] the authors present a fuzzy temporal logic-based approach to optimize energy-efficient routing in wireless sensor networks (WSNs) integrating fuzzy logic with temporal reasoning to enhance sustainability in network operations. These approaches demonstrate the potential of logic-based formalisms in promoting sustainability. However, existing systems often focus on specific application domains and lack the generality and expressive power needed to address broader green-aware scenarios, a gap that our proposed $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ framework aims to fill.

## 3. Preliminaries

This section recaps basic notions of Linear Temporal Logic over Finite Traces ($\mathrm{LTL_f}$) and Answer Set Programming (ASP) that will be used in the rest of the paper.

### 3.1. Linear Temporal Logic over Finite Traces

Linear Temporal Logic (LTL) is a propositional modal logic, that allows one to reason about the *temporal properties* of *traces*. A trace is an infinite sequence of sets of propositional symbols. LTL handles time in an abstract manner, focusing on the *relative order of events* within a trace rather than on the *duration* of those events. Despite its abstract treatment of time, LTL has been widely and successfully applied in various computer science fields, including planning [29, 3, 30], robotics and control theory [31, 32], and business process management [33, 9]. Linear Temporal Logic over Finite Traces ($\mathrm{LTL_f}$) [6] keeps the same syntax as LTL but is interpreted over *finite* traces.

Let $\mathcal{A}$ be a finite set of propositional symbols. An $\mathrm{LTL_f}$ formula is defined according to the following grammar:

$$\varphi := a \in \mathcal{A} \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi$$

Standard shorthand for propositional logic applies. Moreover, we define the *eventually* operator $\mathsf{F}\,\varphi \equiv \top\,\mathsf{U}\,\varphi$, the *always* operator $\mathsf{G}\,\varphi \equiv \neg\mathsf{F}\,\neg\varphi$, and the *release* operator $\varphi_1\,\mathsf{R}\,\varphi_2 \equiv \neg(\neg\varphi_1\,\mathsf{U}\,\neg\varphi_2)$.

A trace $\pi$ is a sequence $\pi = \pi_0, \ldots, \pi_n - 1$, where $\pi_i \subseteq \mathcal{A}$ for each $0 < i < n$; we say that $n$ is the *length* of the trace, denoted by $|\pi|$. A trace $\pi$ *satisfies* an $\mathrm{LTL_f}$ formula $\varphi$ at time $i$, denoted by $\pi, i \models \varphi$, according to the following inductive rules:

- $\pi, i \models \top$ for all $i \in \{0, \ldots, |\pi|\}$
- $\pi, I \models \alpha$ for $\alpha \in \mathcal{A}$ if $\alpha \in \pi_i$
- $\pi, i \models \neg\varphi$ if $\pi, i \not\models \varphi$
- $\pi, i \models \varphi_1 \wedge \varphi_2$ if $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \mathsf{X}\,\varphi_1$ if $i < |\pi| - 1$ and $\pi, i+1 \models \varphi_1$;
- $\pi, i \models \varphi_1\,\mathsf{U}\,\varphi_2$ if $\exists j$ with $i \leq j \leq |\pi|$ s.t. $\pi, j \models \varphi_2$ and $\forall k$ with $i \leq k < j$.

In practical applications, states often represent the system's status at discrete moments, with traces capturing the system's evolution over time. Furthermore, we say that $\pi$ is a model of $\varphi$ if $\pi, 0 \models \varphi$.

**Example 3.** *Consider a green-aware energy management system for smart buildings, where $\mathcal{A} = \{lightOff, lowPowerMode\}$. The formula $\varphi = G\ (lightOff\ U\ \neg(lowPowerMode))$ specifies that the lights must remain off whenever the system is in low power mode, and this condition must hold at all times. This constraint ensures continuous energy saving by keeping lights off during periods of reduced power consumption. An example trace that satisfies $\varphi$ is $\pi = \{lightOff, lowPowerMode\}, \{lightOff, lowPowerMode\}, \{lightOff\}, \{\}$, maintaining the lights off consistently until low power mode is exited.*

## 3.2. Answer Set Programming

Answer Set Programming (ASP) [11, 12] is a declarative knowledge representation formalism rooted in the answer set semantics of logic programs that can solve problems up to $\Sigma_2^P$ in the polynomial hierarchy. This section briefly summarizes basic notions about its syntax, semantics, and reasoning tasks. The examples in the rest of the paper will use the CLINGO input language. Interested readers can refer to [34] for a reference on the modeling capabilities of the language.

### 3.2.1. Syntax.

A normal logic program $P$ is a set of *(normal) rules*, expressions $h \leftarrow a_1, \ldots, a_k, \neg b_1, \ldots, \neg b_k.$, where $h, a_i, b_j$ are *atoms*. An atom is an expression $p(t_1, \ldots, t_k)$ where $p$ is a predicate name, $t_i$ are terms built over a set of *constants* $\mathcal{C}$ that appear in $P$. The symbol $\neg$ denotes *negation-as-failure*, typeset as `not` in code examples. A *literal* is either an atom $a$ or a negated atom $\neg a$, where we say that $\neg a$ is the *opposite literal* of $a$. Given the rule $\rho = h \leftarrow a_1, \ldots, a_k, \neg b_1, \ldots \neg b_k$ we denote by $H(\rho) = \{h\}$ its *head*, $B(\rho) = \{a_1, \ldots, a_k, \neg b_1, \ldots, \neg b_k\}$ its *body*, which can be partitioned into $B^+(\rho) = \{a_1, \ldots, a_k\}$ (*"positive body"*) and $B^-(\rho) = \{b_1, \ldots, b_k\}$ (*"negative body"*). Modern ASP systems' input language accepts many syntactic shortcuts, which ease modeling tasks without increasing the language expressiveness from a complexity point of view, as for example *weight rules*, *cardinality constraints*, and *choice rules* - that can all be rewritten into sets of normal rules. In particular, the disjunctive extension of the language, which allows more atoms in the head of the rule, increases its expressivity up to $\Sigma_2^P$, while normal logic programs can model up to NP problems.

### 3.2.2. Semantics.

Let $\mathcal{B}(P)$ denote the *Herbrand base* of a logic program $P$, which is the set of all possible atoms that can be formed using the predicates and constants present in $P$. A logic program with variables is considered syntactic sugar for its *ground* version, which consists of all the rules that can be generated by substituting variables with atoms from $\mathcal{B}(P)$. An *interpretation* $\mathcal{I}$ is a subset of $\mathcal{B}(P)$, and it *satisfies* a rule $\rho$ if $B^+(\rho) \subseteq \mathcal{I}$ and $B^-(\rho) \cap \mathcal{I} = \emptyset$, where $B^+(\rho)$ and $B^-(\rho)$ represent the positive and negative body of the rule, respectively. If $\mathcal{I}$ satisfies all rules of $P$, then $\mathcal{I}$ is a *model* of $P$. The *reduct* of $P$ with respect to $\mathcal{I}$, denoted $P^{\mathcal{I}}$, is obtained by removing from $P$ all rules $\rho$ where either (i) $B^+(\rho) \nsubseteq \mathcal{I}$ or (ii) $B^-(\rho) \cap \mathcal{I} \neq \emptyset$. A subset-minimal model of the reduct $P^{\mathcal{I}}$ is called an *answer set* (or *stable model*) of $P$. A logic program can have zero, one, or multiple answer sets; it is termed *coherent* if it has at least one answer set, and *incoherent* if it has none. The collection of all answer sets of $P$ is denoted by $AS(P)$.

### 3.2.3. Solving Problems via ASP

The standard approach to solve problems using ASP is to write a (typically non-ground) logic program $P$ such that, given a problem instance encoded by a set of facts $F$, the answer sets of the logic program $P \cup F$ result in solutions to the problem instance at hand. The logic program $P$ is usually written according to the *guess & check* programming technique, and consists of two subprograms: a *guess* program that generates potential answer sets (which typically includes choice rules or disjunctive rules); and a *check* component that asserts solution's properties (e.g., discarding unfit candidate solutions) by means of constraints. An illustrative example of this approach is the *subgraph isomorphism problem.*

**Example 4 (Green-Aware Subgraph Isomorphism).** *Subgraph isomorphism is a classic NP-complete problem [35]. Given two graphs, $G(V, E)$ and $H(V', E')$, the aim is to determine whether there exists a subgraph of $G$ that is isomorphic to $H$. This involves finding a bijection $\sigma : V' \to V_0$, where $V_0 \subseteq V$, such that an edge $(x, y) \in E'$ corresponds precisely to an edge $(\sigma(x), \sigma(y)) \in E$. Subgraph isomorphism can be applied to check the efficient energy distribution in smart grids. Consider two graphs: $G(V, E)$ representing the full power grid and $H(V', E')$ representing an energy-efficient subnetwork configuration. The task is to determine whether a subgraph of $G$ exists that is isomorphic to $H$, reflecting an optimal* *way to manage energy flow. In this context, finding a subgraph isomorphism corresponds to identifying an energy-efficient setup within the larger grid. The bijection $\sigma : V' \to V_0$, where $V_0 \subseteq V$, maps nodes (e.g., energy sources, transformers, and consumers) of the desired configuration $H$ onto nodes in the larger grid $G$, such that every connection $(x, y) \in E'$ matches a corresponding connection $(\sigma(x), \sigma(y)) \in E$.*

*The problem can be modeled using the following logic program. An atom $match(x, y)$ encodes that* node $x \in V'$ is matched to node $y \in V$. *The background knowledge is the set of facts $node/1$, $edge/2$ that encode $V$ and $E$, plus $hnode/1$, $hedge/2$ that encode $V'$ and $E'$.*

```
{ match(HX,X): node(X) } = 1 :- hnode(HX).
:- match(HX,X), match(HY,Y), hedge(HX,HY), not edge(X,Y).
```

*The choice rule generates graph matching between the energy-efficient subnetwork and the full power grid, where the $match(HX, X)$ atom models that the node $HX$ in $H$ is matched onto the active node $X$ in $G$. Finally, a constraint discards candidate solutions that violate the definition of graph isomorphism. The answer sets of the above program encode possible $\sigma$ that yield a valid matching between $H$ and $G$.*

Let $P$ be a logic program. The *cautious consequences* of $P$, denoted by Cautious$(P)$, are the atoms that are true in all answer sets of $P$, while the *brave consequences*, denoted by Brave$(P)$, are the atoms that are true in at least one answer set. If $P$ has a unique answer set, as is often the case when $P$ is a Datalog program, then Cautious$(P)$ = Brave$(P)$. For an atom $\alpha$ and a set of positive literals $M$, we say $\neg\alpha \in M$ if $\alpha \notin M$. An atom $\alpha$ *bravely holds* in $P$, denoted $P \models_b \alpha$, if there exists an answer set $M \in AS(P)$ such that $\alpha \in M$. Conversely, $\alpha$ *cautiously holds* in $P$, denoted $P \models_c \alpha$, if $\alpha$ is present in all answer sets of $P$. By definition, $P \models_b \alpha$ if and only if $P \not\models_c \neg\alpha$.

In a green-aware context, consider a scenario where we use these reasoning modes to optimize energy distribution in a smart grid by finding energy-efficient subgraph configurations. Let $\Pi_{SGI}$ be the program reported in Example 4, $F(G)$ and $F(H)$ be the set of facts encoding the graphs $G$ and $H$, respectively.

**Example 5 (Brave Reasoning in Energy Optimization).** *Suppose we want to determine if there is an energy-efficient subgraph configuration between two power grid structures, $G$ and $H$, that does not use a specific node $x \in V$ (e.g., a transformer station). This question can be addressed using a brave query on the atom $not\_used(x)$, which indicates whether node $x$ is part of any feasible configuration:*

```
used(X) :- match(_,X).
not_used(X) :- not used(X), node(X).
```

*This query checks if there exists at least one subgraph isomorphism where the node $x$ is not used, aligning with a potential energy-saving configuration.*

**Example 6 (Cautious Reasoning in Grid Stability).** *To ensure grid stability, we might need to verify if a specific mapping $\sigma(x) = y$ holds in all valid energy-efficient subgraph configurations. This can be formulated as a cautious query, $\Pi_{SGI} \cup F(G) \cup F(H) \models_c match(y, x)$, ensuring that $x$ consistently maps to $y$ across all subgraph isomorphisms, thus confirming its necessity for stable and sustainable grid operations.*

## 4. The $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ Framework

In this section, we introduce the syntax and semantics of $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$, an extension of $\mathrm{LTL}_\mathrm{f}$ within the $\mathrm{LTL}_\mathrm{f}^\mathrm{MT}$ framework that incorporates queries over logic programs with answer set semantics as an external theory. The core idea, detailed further in the following, is to specify how system properties, represented as cautious and brave queries over a logic program, should evolve over time. In this framework, traces capture the temporal evolution of the system's state in a relational form. Logic programs enrich this relational representation by defining additional domain-specific concepts through rules. Queries over these programs allow us to verify whether certain properties hold at given points in time. Standard linear temporal logic operators are then used to impose constraints on how these properties evolve, providing a powerful mechanism for temporal reasoning within knowledge-intensive environments.

Let $\mathcal{B}$ be an ASP program, referred to as the *background knowledge*. In typical ASP modeling, $\mathcal{B}$ has a fixed *fact schema*, meaning that all atoms conforming to a specific signature are considered *input facts* representing the problem instance and do not appear in the head of any rule within $\mathcal{B}$. We denote the set of all possible input facts for $\mathcal{B}$ as $\mathcal{F}$. Let $\mathcal{A}$ be a finite set of propositional symbols, such that $\mathcal{A} \cap \mathcal{F} = \emptyset$.

Given a trace $\pi$ over the alphabet $\mathcal{A}$ and a fact projection $prj$, the *trace projection* $\pi^*_{prj}$, or simply $\pi^*$, is defined as a trace over $\mathcal{F}$ (interpreted as propositional symbols), specified as:

$$\pi^*(i) = \bigcup_{a \in \pi(i)} prj(a)$$

**Example 7.** *Consider the trace* $\pi = \{a, b\}, \{b, c\}, \{a, c\}$ *over* $\mathcal{A} = \{a, b, c\}$, *and* $prj$ *to be defined as* $prj(a) = \{f(x)\}$, $prj(b) = \{f(y)\}$ *and* $prj(c) = \{g(z), h(y, x)\}$. *Then* $\pi^* = \{f(x), f(y)\}, \{f(y), g(z), h(y, x)\}, \{f(x), g(z), h(y, x)\}$.

Whenever this does not cause ambiguities, with a slight abuse of notation we will describe traces $\pi$ directly as a sequence of sets of facts, rather than explicitly stating the projection function $prj$.

### 4.1. Syntax & Semantics

An $\text{LTL}_\text{f}^\text{ASP}$ formula over a program $\mathcal{B}$ and a propositional alphabet $\mathcal{A}$ is defined by extending the $\text{LTL}_\text{f}$ grammar with additional atomic formulae, as follows:

$$\psi := a \mid b?h \mid c?h$$
$$\varphi := \psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi$$

where $a \in \mathcal{A}$, $h \in \mathcal{B}(P)$, and $b?$ and $c?$ denote *brave entailment* and *cautious entailment*, respectively. Standard shorthands for propositional and temporal logic apply. The satisfaction relation for a formula $\varphi$ is defined inductively, with propositional and temporal operators following the standard $\text{LTL}_\text{f}$ semantics. The specific semantics for atomic formulae $a$, $b?h$, and $c?h$ are as follows:

- $\pi, i \models a$ if $a \in \pi(i)$;
- $\pi, i \models c?h$ if $\mathcal{B} \cup \pi^*_{prj}(i)$ cautiously entails $h$, meaning $h$ is present in all answer sets of the program $\mathcal{B} \cup \pi^*_{prj}(i)$;
- $\pi, i \models b?h$ if $\mathcal{B} \cup \pi^*_{prj}(i)$ bravely entails $h$, meaning $h$ appears in at least one answer set of $\mathcal{B} \cup \pi^*_{prj}(i)$.

It is important to note that $\text{LTL}_\text{f}^\text{ASP}$ extends $\text{LTL}_\text{f}$ only at the level of atomic formulae, leaving the temporal semantics of $\text{LTL}_\text{f}$ unchanged. Thus, any $\text{LTL}_\text{f}$ formula is also a valid $\text{LTL}_\text{f}^\text{ASP}$ formula (using a projection function which assigns each $a \in \mathcal{A}$ the singleton $\{a'\}$). Subformulae of the types $b?h$ and $c?h$ express additional constraints in terms of brave or cautious entailment over $\mathcal{B} \cup \pi^*(i)$, enriching the expressive power of the logic.

### 4.2. Trace Verification

We focus on the *trace verification* problem for $\text{LTL}_\text{f}^\text{ASP}$, defined as follows:

**Definition 1 (Trace Verification).** *Let* $\varphi$ *be an* $\text{LTL}_f^{ASP}$ *formula and* $\pi$ *a trace. The* trace verification *problem consists of determining whether* $\pi \models \varphi$.

To address the trace verification problem, we can apply any standard technique used for evaluating LTL$_f$ formulas over a trace, handling standard temporal operators and atomic formulae directly, while using an external oracle to evaluate the brave and cautious entailment of specific atomic formulae. Specifically, given a formula $\varphi$, we define its *boolean abstraction* $\varphi^*$ as the LTL$_f$ formula obtained by replacing each brave and cautious entailment atomic formula with a propositional variable, effectively treating $c?h$ and $b?h$ as propositional symbols.

We then use standard techniques to check whether $\pi \models \varphi^*$. However, when an entailment needs evaluation, we delegate the task to an external ASP oracle. In the worst case, this approach may require up to $|\pi| \times |Q|$ calls to the external oracle, where $|Q|$ represents the number of entailment queries.

From an implementation perspective, another viable approach could be a two-phase process: first, evaluate all entailment queries over the trace to generate a projection, and then verify the resulting LTL$_f$ formula against this projection.

## 5. Green-Aware Application Scenarios of LTL$_f^{ASP}$

The introduction of LTL$_f^{ASP}$ offers powerful tools for reasoning about dynamic, knowledge-intensive systems in green-aware applications. By combining the temporal reasoning capabilities of LTL$_f$ with the expressive power of Answer Set Programming (ASP), LTL$_f^{ASP}$ enables advanced modeling and verification of complex temporal behaviors that are crucial for sustainable resource management. In this section, we explore several application scenarios where LTL$_f^{ASP}$ can be effectively employed to enhance energy efficiency, optimize resource utilization, and support decision-making processes in environmentally conscious contexts.

**Green-Aware Energy Flow Control.** Consider an energy distribution network modeled as a directed graph $G(V, E)$, where nodes represent substations and consumers, and edges represent power lines connecting them. The goal is to verify that critical nodes, represented by the set $C \subset V$ (e.g., hospitals, data centers), remain powered while the network configuration is dynamically adjusted to minimize energy losses. Figure 1 depicts an example. Red nodes represent critical nodes, the yellow one is the power source node and the green ones the functioning nodes at each time instant.

We can model this scenario as follows. The background knowledge $\mathcal{B}$ captures the "static" properties of the network, its underlying graph as facts, as well as the notions of *node isolation* and *being powered* (e.g., connected to the source node).

```
% Network Topology
node(1). node(2). node(3). node(4).  node(5). node(6). node(7). node(s).
edge(1,2). edge(1,6). edge(2,3). edge(2,6). edge (2,7). edge(3,4).
edge(3,7). edge(4,7). edge(4,5). edge(5,7). edge(6,7).
edge(source,1). edge(source,6).
criticalConsumer(2). criticalConsumer(4).
% Domain definition: Connectedness & Isolation
reachable(X, Y) :- edge(X, Y), on(X), on(Y).
reachable(X, Z) :- edge(X, Y), on(X), reachable(Y, Z).
isolated(X) :- node(X), not reachable(source, X).
% A critical consumer is not connected to power
fail :- criticalConsumer(X), isolated(X).
```

The rule `fail :- criticalConsumer(X), isolated(X).` derives the atom `fail` whenever there's a critical, isolated node. Notice that since $\mathcal{B}$ is a Datalog program with a unique answer set, brave reasoning and cautious reasoning coincide. Input facts for this program will match the signature $on/1$, where an atom $on(x)$ denotes that node $x$ is powered on.

The LTL$_f^{ASP}$ formula below specifies that certain critical consumers (represented by *criticalConsumer*$(x)$) should not be isolated, i.e., they must remain connected to an energy

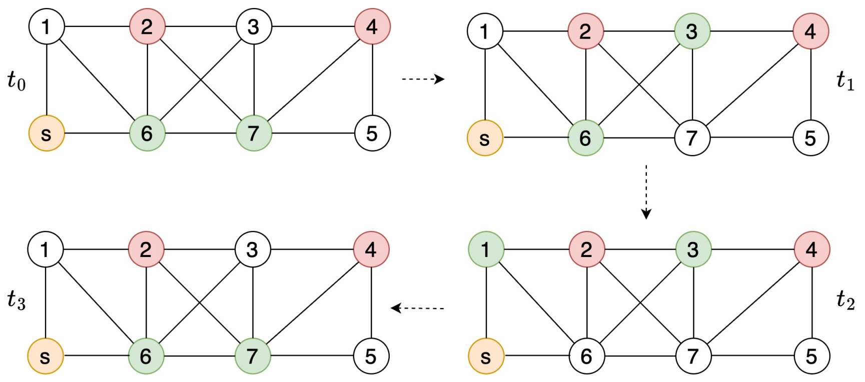

**Figure 1:** Temporal evolution of a green-aware energy management system showing critical consumers (in red), source node (in yellow), and active components (in green) at each time step.

source. This requirement is enforced over time using a cautious entailment query. Moreover, to avoid node overload it also specifies that each (not critical) node must be switched off at a certain point.

$$\varphi = \mathsf{G}\,(\neg c?\mathit{fail}) \wedge \bigwedge_{n \in V \setminus C} (\mathsf{F}\,(\neg \mathrm{on}(n)))$$

Traces capture the "dynamic" part of the system, that is which nodes are powered over time. The following trace encodes the network evolution in Figure 1:

$$\pi = \{\mathrm{on}(\mathit{source}), \mathrm{on}(6), \mathrm{on}(7), \mathrm{on}(2), \mathrm{on}(4)\}, \{\mathrm{on}(\mathit{source}), \mathrm{on}(6), \mathrm{on}(3), \mathrm{on}(2), \mathrm{on}(4)\},$$
$$\{\mathrm{on}(\mathit{source}), \mathrm{on}(1), \mathrm{on}(3), \mathrm{on}(2), \mathrm{on}(4)\}, \{\mathrm{on}(\mathit{source}), \mathrm{on}(6), \mathrm{on}(7), \mathrm{on}(2), \mathrm{on}(4)\}$$

The trace $\pi$ satisfies the formula $\varphi$. In fact, for every non-critical consumer node there is at least one time instant in which the node is switched off and, in all time instants, critical nodes are reachable from the source node by only passing through functioning nodes.

**Efficient Energy Distribution in Smart Grids.** Consider two graphs: $G(V, E)$, representing a full power grid, and $H(V', E')$, representing an energy-efficient subnetwork configuration. Nodes in the smart grid may be damaged, overloaded, or under maintenance, making them temporarily unavailable. The task is to determine whether, during the grid's operation, there always exists a subgraph of the evolving grid $G$ that is isomorphic to $H$, reflecting an optimal energy flow configuration. Similarly to the previous apprication scenario, the background knowledge $\mathcal{B}$ is used to capture the structure of the two networks, as well as the notion of *isomorphism.*

To model this scenario, we assume the input trace contains facts matching the signature $on/1$, to model that a certain node is active. The background knowledge $\mathcal{B}$ models the full power grid by means of facts $node/1$, $edge/2$ that encode $V$ and $E$, while the energy-efficient subnetwork is modeled by the predicates $hnode/1$, $hedge/2$.

```
success.
active(X,Y) :- on(X), on(Y), edge(X,Y).
{ match(HX,X): on(X) } = 1 :- hnode(HX).
:- match(HX,X), match(HY,Y), hedge(HX,HY), not active(X,Y).
```

The $active/2$ predicate filters grid links between active nodes, discarding ones where at least one of the endpoints is not active. The choice rule generates graph matching between the energy-efficient subnetwork and the active power grid, where the $match(HX, X)$ atom models that the node $HX$ in the energy-efficient subnetwork is matched onto the active node $x$ of the full power grid. Finally, a constraint discards candidate solutions that violate the definition of graph isomorphism.

As in the previous application case, input facts for this program have signature $on/1$, where an atom $on(x)$ denotes that node $x$ is functioning.

The $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ formula below specifies that always, during its working, the subnetwork of $G$ induced by only functioning nodes, contains the given optimal energy flow configuration, i.e., a subgraph isomorphic to the energy-efficient subnetwork. This requirement is enforced over time using a brave uentailment query.

$$\varphi = \mathrm{G}\ (b?\mathrm{success})$$

## 6. Conclusion

In this paper, we introduced $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$, a novel framework that extends Linear Temporal Logic over Finite Traces ($\mathrm{LTL}_\mathrm{f}$) by integrating it with Answer Set Programming (ASP), offering a powerful approach for reasoning about dynamic systems in green-aware applications. By combining the temporal reasoning capabilities of $\mathrm{LTL}_\mathrm{f}$ with the declarative and expressive power of ASP, $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ enables efficient modeling and verification of complex temporal behaviors, making it particularly well-suited for knowledge-intensive domains.

Through a series of examples, we demonstrated how $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ provides expressive and manageable solutions, particularly when handling properties like node reachability or graph connectivity. The integration of ASP allows for high-level, declarative reasoning about individual states derived from temporal specifications, significantly enhancing the framework's flexibility and expressiveness in modeling complex systems. Future work will explore additional reasoning tasks within the $\mathrm{LTL}_\mathrm{f}^\mathrm{ASP}$ framework, such as satisfiability checking, and extend its applicability beyond green-aware domains to other dynamic systems requiring advanced temporal reasoning.

**Acknowledgments.** This work was supported by the Italian Ministry of Research (MUR) under PRIN project PINPOINT - CUP H23C22000280006; PRIN project HypeKG - CUP H53D23003710006; PRIN 2022 PNRR project DISTORT funded by European Union Next Generation EU Mission 4 Component 1 - CUP: H53D23008170001; PNRR projects FAIR "Future AI Research" - Spoke 9 - WP9.1 and WP9.2, Spoke 5 - WP5.1 and Spoke 6, through the NRRP MUR Program funded by the NextGenerationEU and by the ERC-ADG WhiteMech (No. 834228).

## References

[1] V. Bolón-Canedo, L. Morán-Fernández, B. Cancela, A. Alonso-Betanzos, A review of green artificial intelligence: Towards a more sustainable future, Neurocomputing 599 (2024) 128096.
[2] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1977, pp. 46–57.
[3] D. Calvanese, G. De Giacomo, M. Y. Vardi, Reasoning about actions and planning in LTL action theories, in: KR, 2002, pp. 593–602.
[4] 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.
[5] 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.
[6] 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/AAAI, 2013, pp. 854–860.
[7] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623.
[8] C. Dodaro, V. Fionda, G. Greco, LTL on weighted finite traces: Formal foundations and algorithms, in: IJCAI, ijcai.org, 2022, pp. 2606–2612.
[9] V. Fionda, A. Guzzo, Control-flow modeling with declare: Behavioral properties, computational complexity, and tools, IEEE TKDE 32 (2020) 898–911.
[10] C. Di Ciccio, M. Montali, Declarative process specifications: Reasoning, discovery, monitoring, in: Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 108–152.

[11] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[12] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386.

[13] L. Geatti, A. Gianola, N. Gigante, S. Winkler, Decidable fragments of ltl$_f$ modulo theories, in: K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, R. Radulescu (Eds.), 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*, IOS Press, 2023, pp. 811–818.

[14] P. Baldan, P. Mancarella, A. Raffaetà, F. Turini, Mutaclp: A language for temporal reasoning with multiple theories, in: Computational Logic: Logic Programming and Beyond, volume 2408 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 1–40.

[15] L. Geatti, N. Gigante, A. Montanari, G. Venturato, SAT meets tableaux for linear temporal logic satisfiability, J. Autom. Reason. 68 (2024) 6.

[16] B. Finkbeiner, P. Heim, N. Passing, Temporal stream logic modulo theories, in: FoSSaCS, volume 13242 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 325–346.

[17] S. Demri, D. D'Souza, An automata-theoretic approach to constraint LTL, Inf. Comput. 205 (2007) 380–415.

[18] A. Artale, A. R. Gnatenko, V. Ryzhikov, M. Zakharyaschev, On deciding the data complexity of answering linear monadic datalog queries with LTL operators (extended abstract), in: Description Logics, volume 3739 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024.

[19] A. Artale, A. Mazzullo, A. Ozaki, First-order temporal logic on finite traces: Semantic properties, decidable fragments, and applications, ACM Trans. Comput. Log. 25 (2024) 13:1–13:43.

[20] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning: Semantics and complexity, ACM Trans. Comput. Log. 5 (2004) 206–263.

[21] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann, C. Vidal, Linear-time temporal answer set programming, Theory Pract. Log. Program. 23 (2023) 2–56.

[22] E. D. Valle, S. Ceri, F. van Harmelen, D. Fensel, It's a streaming world! reasoning upon rapidly changing information, IEEE Intell. Syst. 24 (2009) 83–89.

[23] H. Beck, T. Eiter, C. Folie, Ticker: A system for incremental asp-based stream reasoning, Theory Pract. Log. Program. 17 (2017) 744–763.

[24] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: ICLP (Technical Communications), volume 52 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:15.

[25] C. W. Barrett, C. Tinelli, Satisfiability modulo theories, in: Handbook of Model Checking, Springer, 2018, pp. 305–343.

[26] A. Armando, C. Castellini, E. Giunchiglia, M. Idini, M. Maratea, TSAT++: an open platform for satisfiability modulo theories, in: D/PDPAR@IJCAR, volume 125 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2004, pp. 25–36.

[27] M. Ehsanpour, L. Baresi, M. Rossi, E. Damiani, Analysis of energy-efficient buildings through simulation and formal methods, in: SIMPDA, volume 1757 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 113–119.

[28] M. Selvi, R. Logambigai, S. Ganapathy, L. S. Ramesh, H. K. Nehemiah, K. Arputharaj, Fuzzy temporal approach for energy efficient routing in WSN, in: ICIA, ACM, 2016, pp. 117:1–117:5.

[29] F. Bacchus, F. Kabanza, Planning for temporally extended goals, Ann. Math. Artif. Intell. 22 (1998) 5–27.

[30] S. Sohrabi, J. A. Baier, S. A. McIlraith, Preferred explanations: Theory and generation via planning, in: AAAI, AAAI Press, 2011, pp. 261–267.

[31] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, S. M. LaValle, Controlling wild bodies using linear temporal logic, in: Robotics: Science and Systems, 2011.

[32] X. C. Ding, S. L. Smith, C. Belta, D. Rus, Optimal control of markov decision processes with linear temporal logic constraints, IEEE Trans. Autom. Control. 59 (2014) 1244–1257.

[33] F. M. Maggi, M. Westergaard, M. Montali, W. M. P. van der Aalst, Runtime verification of ltl-based declarative process models, in: RV, volume 7186 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 131–146.

[34] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82.

[35] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.