Progression for Monitoring in Temporal ASP*

Davide Soldá, Ignacio D. Lopez-Miguel, Ezio Bartocci and Thomas Eiter

TU Wien, Vienna, Austria
{davide.solda, ignacio.lopez, ezio.bartocci, thomas.eiter}@tuwien.ac.at
ORCiD ID:

Abstract. In recent years, there has been growing interest in the application of temporal reasoning approaches and non-monotonic logics from artificial intelligence in dynamic systems that generate data. A well-known approach to temporal reasoning is the use of the progression technique, which allows for the online computation of logical consequences of a logical knowledge base over time. We consider the progression technique for Temporal Here and There and Temporal Equilibrium Logic, which is the logic underlying answer programming over linear-temporal logic (LTL). Compared to usual LTL online computation, where the goal is to check whether a trace is compliant with a temporal specification, our approach provides also the means to compute non-monotonic temporal reasoning over a trace of observations. Besides formal notions and results, we also present an algorithm for performing progression to monitor a dynamic system, which has been implemented as a proof of concept and allows for handling expressive application scenarios.

1 Introduction

With the increase of data-driven applications, reasoning about their behavior as they evolve in time has become crucial. Temporal logics provide a formal framework to specify the desired temporal behavior in an unambiguous way. Over the last decade, there has been a great effort in artificial intelligence to develop temporal reasoning approaches based on non-monotonic logics and answer set programming [11, 24] (ASP). A popular example is *Temporal Equilibrium Logic* [1, 2] (TEL), which combines Linear Temporal Logic [28] (LTL) and *Equilibrium Logic* [27] in an orthogonal way. TEL is a nonmonotonic version of *Temporal Here-and-There Logic* [2] (THT), which is an intuitionistic version of LTL based on Heytings's Logic of Here-and-There [21] amounting to 3-valued Gödel Logic.

Tools for computing stable traces of temporal logic programs have been presented already in the literature, among them TELINGO[15], STeLP[14], and the algorithm in [13]. The latter two are automata-based; a Büchi-based approach as in [14] may have to handle an exponential number of loop formulas to ensure the stability of traces. TELINGO instead uses *incremental ASP* and is limited to finite traces.

How to monitor TEL specifications over infinite traces is an open problem. In this paper, we take inspiration from rewriting-based techniques proposed in runtime verification [30], in situation calculus [23] to reason about actions and in planning with goals specified using Metric Interval Temporal Logic [6]. However, all these works are based on classical semantics. The key idea of progression is to rewrite, as new observations of the system become available, an answer-set program expressing the obligations that need to be fulfilled in the future in order to have a true or a false verdict.

The main difference between TEL/THT and LTL is the semantics for the implication. TEL/THT is more expressive than LTL [2]: LTL formulas can be encoded into THT, but the converse is not always possible. Furthermore, LTL is monotonic while TEL is nonmonotonic and thus more suited for handling exceptions and dealing with incomplete information. In contrast with rule-based languages such as temporal Datalog [32, 29], TEL/THT uses explicit temporal modal operators to reason about time instead of rules. In [31] there has been proposed a stream reasoning framework based on Metric Temporal Logic, under stable semantics.

Computing stable models (aka answer sets or equilibrium models) of logic programs where time is involved can be a challenging task, especially when dealing with infinite traces, which may be necessary when observing a reactive system. The behavior of such a system may depend on the environment; hence the possibility to monitor complex properties that require reasoning is desirable. In nonmonotonic reasoning and logic programming, *skeptical* (*cautious*) reasoning is a common approach to inference, in which in a Tarskian manner the intersection of all answer sets of a program is considered. Our approach thus aims to compute the intersection of all prefixes of all stable traces of a temporal program, by taking also possible future observations on the system under scrutiny into account.

In order to compute prefixes of stable traces of the program, we introduce a novel online computation based on the idea of progression already used in runtime verification [8] and in planning where the goal is specified using Metric Temporal Logic [6]. To the best of our knowledge, this is the first time that this approach has been used in temporal answer set programming. In comparison to incremental ASP, this approach incorporates the observations into the program, which does not lead to an increase in the program size. Furthermore, it maintains a program at the symbolic level, making also visible which information is missing. It provides a basis for reasoning about future evolution in a flexible way, e.g., to find possible sequences of observations that will ensure a specification will be satisfied.

To validate and present an application of our algorithm, we use a temporal version of the well-known Σ_2^p strategic companies problem [16, 22], where we assume to have an incoming trace of observations about the evolution of ownership of companies and production of goods. An example of a temporal property that we may want to verify at runtime is whether $\mathbf{G}(contr\_by(c, c1, c2) \rightarrow \mathbf{F}(contr\_by(c, c1, c1)))$ is satisfied, which informally means that whenever at some point in the future, company c is controlled by

* ECAI 2023, to appear (uncorrected preprint)

$c1$ and $c2$, it will eventually be controlled only by $c1$.

Our contribution We define a novel approach for monitoring by progression TEL/THT formulas over an infinite sequence of observations. In particular, we develop an algorithm to compute online the intersection of all the stable traces with a stream of incoming observations. We provide an experimental implementation and validate our approach on a case study of the strategic companies problem.

Paper organization In Section 2 we provide the necessary background on Temporal Here and There Logic (THT) and Temporal Equilibrium Logic (TEL). Section 3 and 4 introduce our approach of progression for monitoring TEL and THT, respectively. In Section 5 we present an algorithm for computing online the intersection of all the temporal traces of a given temporal program with an incoming trace of observations. In Section 6, we demonstrate an application of our approach to a case study on monitoring strategic companies and provide some experimental results. We discuss related work in Section 7 and conclude in Section 8 with an outlook on future work.

2 Preliminaries

As Equilibrium Logic (EL) [27] can be seen as the logic underlying Answer Set Programming (ASP) [27], Temporal Equilibrium Logic (TEL) over infinite traces [2] can be interpreted as a pure logical temporal extension of ASP. Note that the definition of EL is based on Here-and-There Logic (HT), as TEL is defined over Temporal Here-and-There Logic (THT). The logics have the same formulas; we are interested in a fragment with the following syntax:

$$\begin{aligned} F &::= \bot \mid p \mid F \circ F \mid F \to F \mid \mathbf{X}F \mid \mathbf{G}\, F \mid \mathbf{F}\, F \mid \mathbf{Y}\, F' \\ F' &::= \bot \mid p \mid \mathbf{Y}\, F' \end{aligned} \tag{1}$$

Where $p \in \mathcal{P}$ for a finite set $\mathcal{P}$ of propositional atoms and $\circ \in \{\wedge, \vee\}$. Negation is defined as $\neg\phi \equiv \phi \to \bot$, and $\top \equiv \neg\bot$. The temporal operators $\mathbf{G}$ (globally) and $\mathbf{F}$ (finally) are defined as usual, viz. $\mathbf{G}\phi \equiv \bot\mathbf{R}\phi$ and $\mathbf{F}\phi \equiv \top\mathbf{U}\phi$ respectively. We denote by $\mathcal{L}$ the set of all the THT resp. TEL formulas.

The semantics of THT is defined over sequences of pairs of sets of atoms. A THT interpretation $\langle H, T\rangle$ is an infinite sequence of pairs $\langle H_i, T_i\rangle$ for $i \geq 0$, where $H_i \subseteq T_i$ for each i. In contrast, a TEL trace T can be viewed as a THT trace $\langle T, T\rangle$, and we may identify $\langle T, T\rangle$ by T if there is no confusion.

Definition 1 (THT-Satisfaction)**.** *Satisfaction of a THT formula by a THT-trace $I = \langle H, T\rangle$, at time k, where $k \geq 0$ is integer, is inductively defined as follows:*

- $I, k \models p$ *iff* $p \in H_k$*, for any atom* $p \in \mathcal{P}$
- $I, k \models \mathbf{Y}\,\phi$ *iff* $I, k-1 \models \phi$ *and* $k > 0$
- $I, k \models \phi \vee \psi$ *iff* $I, k \models \phi$ *or* $I, k \models \psi$
- $I, k \models \phi \wedge \psi$ *iff* $I, k \models \phi$ *and* $I, k \models \psi$
- $I, k \models \phi \to \psi$ *iff* $\begin{cases} \langle T, T\rangle, k \not\models \phi \text{ or } \langle T, T\rangle, k \models \psi, \text{ and} \\ I, k \not\models \phi \text{ or } I, k \models \psi \end{cases}$
- $I, k \models \mathbf{X}\,\phi$ *iff* $I, k+1 \models \phi$
- $I, k \models \phi\,\mathbf{U}\,\psi$ *iff there is* $j \geq k$ *s.t.* $I, j \models \psi$*, and for all* $j' \in [k, j-1]$*,* $I, j \models \phi$
- $I, k \models \phi\,\mathbf{R}\,\psi$ *iff for all* $j \geq k$ *s.t.* $I, j \not\models \psi$*, there exists* $j' \in [k, j-1]$*,* $I, j \models \phi$
- $I, k \not\models \bot$

A trace I is a model for a formula ϕ if $I, 0 \models \phi$.

We recall that $\langle T, T\rangle \models \phi$ if and only if $T \models_{LTL} \phi$ with $\phi \in \mathcal{L}$ [4]. An interpretation I is *total* if $H = T$. Furthermore, given two interpretations I and $\langle H', T\rangle$, and a trace of observations O such that $O_i \subseteq H_i$ and $O_i \subseteq H'_i$, for each $0 \leq i$, we say that $\langle H', T\rangle \leq_O \langle H, T\rangle$ if $H'_i \subseteq H_i$ for each $i \geq 0$. Intuitively, $\leq_O$ allows for H-minimality modulo observations; in our approach, observations are added online as facts and thus they do not need to be proven. To simplify the notation in the following sections if we do not have to make explicit the observation trace, we write $\langle H', T\rangle \leq \langle H, T\rangle$, instead of $\langle H', T\rangle \leq_O \langle H, T\rangle$. We are now ready to introduce the semantics of TEL.

Definition 2 (TEL-Satisfaction Modulo Observations)**.** *Given a trace of observations O, a trace T is an temporal equilibrium model of a formula $\phi \in \mathcal{L}$ modulo O if the following two conditions hold: (i) $\langle T, T\rangle \models \phi$, i.e., T is a total THT model of ϕ, and (ii) no $\langle H, T\rangle \leq_O \langle T, T\rangle$ s.t. $\langle H, T\rangle \models \phi$ exists, i.e., $\langle T, T\rangle$ has to be minimal modulo observations O.*

We note that if the trace of observations is the empty trace, Definition 2 collapses to classical TEL satisfaction [2]. Given two traces T and O, and a formula $\phi \in \mathcal{L}$, we denote by $T \models^O_{TEL} \phi$ that T is a equilibrium trace of ϕ modulo observations O. In case O is clear from the context, we may just write $T \models_{TEL} \phi$.

In the next sections, we will use interchangeably temporal equilibrium model and equilibrium/stable traces, furthermore, we will use a normal form for a generic THT resp. TEL formula $\phi \in \mathcal{L}$, called *temporal program*. The translation into normal form uses a Tseitin-style reduction and preserves equivalence under THT semantics and thus strong equivalence [3].

Definition 3 (Temporal program)**.** *Given a set $\mathcal{P}$ of propositional atoms, we define the set of* temporal literals *as* $\{p, \neg p, \mathbf{X}p, \neg\mathbf{X}p, \mathbf{G}p, \mathbf{F}p\}$*, where $p \in \mathcal{P}$. Atoms with the negation as failure in front of the atom are called* negative*, otherwise, they are called* positive*. A temporal rule is either:*

- *an* initial *rule of the form*

$$r : b_1 \wedge \ldots b_k \wedge \neg b_{k+1} \wedge \ldots \neg b_n \to c_1 \vee \cdots \vee c_l \tag{2}$$

  where all b_i, $c_j \in \{p, \mathbf{X}p\}$ and $\neg$ is negation as failure;
- *a* dynamic *rule of the form* $\mathbf{G}r$*, where r is an initial rule;*
- *a* fulfillment *rule of form either* $\mathbf{G}(\mathbf{G}p \to q)$ *or* $\mathbf{G}(p \to \mathbf{F}q)$*, where p, q are atoms.*

An initial or dynamic rule r is a constraint*, if its head is $\bot$, and is a* fact *if its body is empty (n=0) and its head is a single positive literal. A* temporal program *is any set of temporal rules.*

In the original definition of temporal program [12], negated literals were admitted in the head of a rule, while in Definition 3 we do not. We restrict the syntax to simplify our exposition in the upcoming sections. Note that by using a fresh auxiliary atom it is always possible to rewrite a rule with negation in the head into one without.

Temporal programs may be seen as a temporal extension of logic programs, which consist only of rules like (2), where $b_i, c_j \in \mathcal{P}$. We will use interchangeably the terms answer sets and stable models.

We introduce also some notations. If r is a temporal rule, we denote by $lits(r)$ the set of temporal literals appearing in r. Furthermore, let $B(r)$ and $H(r)$ be the set of temporal literals occurring, respectively, in the body and in the head of the rule r. Moreover, let $B^+(r)$ be the set of positive literals in $B(r)$, and $B^-(r)$ be the set of negative literals in $B(r)$. We also use the shortcuts $lits^+(r) = B^+(r) \cup H(r)$, and $lits^-(r) = B^-(r)$.

3 Progression for THT

In online computation, we usually do not have the full trace, but only a prefix of it. We thus propose the following THT_3 semantics. A prefix of a THT-trace I is any sequence $I^f = \langle H^f, T^f \rangle = \langle H_0, T_0 \rangle, \dots, \langle H_k, T_k \rangle$ (the prefix of length $k+1$) while a suffix of I is any sequence $I^{k,\cdots} = \langle H_k, T_k \rangle, \langle H_{k+1}, T_{k+1} \rangle, \dots$ (the suffix at k or k-suffix), where $k \geq 0$. A THT-prefix is a prefix of any THT-trace I; by Pre_{THT} we denote the set of all THT-prefixes. For any prefix I^f and trace O, a THT-trace I is an extension if I^f is a prefix of I and $O \leq H$; by $ext(I^f, O)$ we denote the set of all such extensions.

Definition 4 (THT_3 semantics)**.** *The truth value of $\phi \in \mathcal{L}$ with respect to a THT-prefix I^f and a trace O of observations is as follows:*

$$I^f \models^O_{THT_3} \phi = \begin{cases} \top & \textit{if } I \models \phi \textit{ for every } I \in ext(I^f, O), \\ \bot & \textit{if } I \not\models \phi \textit{ for every } I \in ext(I^f, O), \\ ? & \textit{otherwise}. \end{cases}$$

Note that in case O is the empty trace, Definition 4 is the Temporal Here and There version of the LTL_3 logic proposed in [8]. We add the observation trace O as a parameter since we use Definition 4 for a 3-valued logic for TEL, where minimality on the trace matters.

In order to process one state at a time, we resort to the concept of progression and introduce it for THT. In that, we omit the temporal operators $\mathbf{U}$ and $\mathbf{R}$, which do not appear in the normal form, and focus on $\mathbf{F}$ and $\mathbf{G}$.

In the progressive evaluation of a THT formula, we may be able to evaluate an implication $p \to \mathbf{X}q$ only partially in the current state, and we must delegate the remaining part of the evaluation to the future. To this end, we introduce $\to_c$ as a new type of implication for evaluation in the *There* part of the trace, in order to ensure that the remaining evaluation is compliant with the THT semantics.

We denote by $\mathcal{L}^P$ the set of formulas generated by the grammar in (1), where in place of $\to$ also $\to_c$ may occur. Note that we exclude nesting of $\mathbf{G}, \mathbf{F}, \mathbf{X}$ into $\mathbf{Y}$ operators. We can now introduce the definition of THT progression.

Definition 5 (THT progression on a state of a prefix)**.** *Progression $P_{THT} : \mathcal{L}^P \times Pre_{THT} \times \mathbb{N} \to \mathcal{L}^P$ is the partial function that maps a formula ψ, a THT-prefix $I^f = \langle H^f, T^f \rangle$ of length k, and an integer i such that $0 \leq i < k$ to an $\mathcal{L}^P$ formula as follows:*

- $P_{THT}(\bot, I^f, i) = \bot$
- $P_{THT}(p, I^f, i) = \top$ *if* $p \in H^f_i$, *and* $p \in \mathcal{P}$
- $P_{THT}(p, I^f, i) = \bot$ *if* $p \notin H^f_i$, *and* $p \in \mathcal{P}$
- $P_{THT}(\mathbf{Y}\phi, I^f, i) = P_{THT}(\phi, I^f_{i-1})$ *if* $i > 0$, *otherwise* $\bot$
- $P_{THT}(\phi_1 \lor \phi_2, I^f, i) = P_{THT}(\phi_1, I^f, i) \lor P_{THT}(\phi_2, I^f, i)$
- $P_{THT}(\phi_1 \land \phi_2, I^f, i) = P_{THT}(\phi_1, I^f, i) \land P_{THT}(\phi_2, I^f, i)$
- $P_{THT}(\mathbf{X}\,\phi, I^f, i) = \phi$
- $P_{THT}(\phi_1 \to \phi_2, I^f, i) =$ $\begin{cases} P_{THT}(\phi_1, I^f, i) \to P_{THT}(\phi_2, I^f, i) \land \\ P_{THT}(\phi_1, \langle T^f_i, T^f_i \rangle) \to_c P_{THT}(\phi_2, \langle T^f_i, T^f_i \rangle) \end{cases}$
- $P_{THT}(\phi_1 \to_c \phi_2, I^f, i) =$ $P_{THT}(\phi_1, \langle T^f_i, T^f_i \rangle) \to_c P_{THT}(\phi_2, \langle T^f_i, T^f_i \rangle)$
- $P_{THT}(\mathbf{G}\phi, I^f, i) = P_{THT}(\phi, I^f, i) \land \mathbf{G}\phi$
- $P_{THT}(\mathbf{F}\phi, I^f, i) = P_{THT}(\phi, I^f, i) \lor \mathbf{F}\phi$

In addition, $\top \to^* \bot$ is replaced by $\bot$; $\bot \to^* \phi$ by $\top$; and $\phi \to^* \top$ by $\top$, for each formula ϕ and $\to^* \in \{\to, \to_c\}$. Furthermore, $\top \lor \phi$ is replaced by $\top$; $\bot \lor \bot$ by $\bot$; $\bot \land \phi$ by $\bot$; and $\top \land \top$ by $\top$.

Note that we do not apply the progression recursively on the future states, but we indeed apply it recursively on the sub-prefix of the trace, as we assume to have access to the current and past states.

Let us denote the recursive application of the progression over a finite trace of length $k+1$ in the following way

Definition 6 (THT progression over prefixes)**.** *For any THT-prefix of length $k+1$ and formula $\phi \in \mathcal{L}$, the application of the progression to ϕ over I^f is defined as*

$$P_{THT}(\phi, I^f) \equiv P_{THT}(\dots P_{THT}(\phi, I^f, 0) \dots, I^f, k) \quad (3)$$

We have now all the definitions needed to state the main result of this section.

Theorem 1 (THT verdict on prefixes)**.** *For every THT-prefix I^f, trace O of observations, and formula $\phi \in \mathcal{L}$, progression leads to the same verdict of the $\models^O_{THT_3}$ semantics, i.e.,*

$$P_{THT}(\phi, I^f) = v \implies I^f \models^O_{THT_3} \phi = v, \textit{ for } v \in \{\top, \bot\}.$$

4 Progression for TEL

For progression of TEL formulas, we start with a 3-valued semantics.

Definition 7 (TEL_3 semantics)**.** *Let T^f be a TEL-prefix of length k, $\phi \in \mathcal{L}$, and O be a trace of observations. Then*

$$T^f \models^O_{TEL_3} \phi = \begin{cases} \top & \textit{if } T^f O^{k,\cdots} \models^O_{TEL} \phi, \\ \bot & \textit{if } T \not\models^O_{TEL} \phi \,\forall\, \langle H, T \rangle \in ext(\langle T^f, T^f \rangle, O), \\ ? & \textit{otherwise}. \end{cases}$$

The following example explains why we require minimal LTL models in Definition 7.

Example 1. *Let us consider first $\phi = \mathbf{G}(\neg\neg p \to p)$, i.e., a TEL tautology. In this case, we have that each trace T is a temporal equilibrium trace, and we conclude that $T^f \models^O_{TEL_3} \phi = \top$ for any possible O.*

Let us focus on an LTL tautology that is not a TEL tautology. Let $\phi = \top$, and $T^f = \emptyset$ is the prefix of lengh 1. Then, because of minimality, the only possible extension that is a TEL trace is the $T^f O^{1,\cdots}$, which is indeed the minimal LTL trace modulo observation O. Therefore, $T^f \models^O_{TEL_3} \phi = \top$.

Given $T = \emptyset$, let us analyze now a more interesting LTL tautology, $\phi = \mathbf{GF}p \lor \mathbf{GF}\neg p$. Thanks to minimality in the $Here$, the only extension of T that is an equilibrium trace is the one $p \in T_i$ if and only if $p \in O_i$. We conclude also in this case that $T^f \models^O_{TEL_3} \phi = \top$.

Definition 8 (TEL progression on a prefix of a trace)**.** *Let T^f be a prefix of a TEL trace and $\phi \in \mathcal{L}$ a TEL formula. Then the TEL progression of ϕ on T^f is defined as*

$$P_{TEL}(\phi, T^f) = \begin{cases} \top & \textit{if } \phi' = \top, \textit{ and } \psi(H^f) = \bot \textit{ for all } H^f \subset T^f \\ \bot & \textit{if } \phi' = \bot, \textit{ or } \psi(H^f) = \top \textit{ for some } H^f \subset T^f \\ ? & \textit{otherwise,} \end{cases}$$

where $\phi' = P_{THT}(\phi, \langle T^f, T^f \rangle)$, $\psi(H^f) = P_{THT}(\psi, \langle H^f, T^f \rangle)$.

Theorem 2 (TEL verdict on prefixes)**.** *Let T^f be a TEL-prefix of length k, O be a trace of observations, and ϕ be a formula. Then progression leads to the same verdict of the $\models_{THT_3}$ semantics, i.e.,*

$$P_{TEL}(\phi, T^f) = v \implies T^f \models^O_{TEL_3} \phi = v, \textit{ for } v \in \{\top, \bot\}.$$

Example 2 (**GF** $\neg p$)**.** *Let $\phi =$ **GF** $\neg p$, $O = \emptyset^\omega$ and $T^f = \emptyset$, then $P_{TEL}(\phi, T^f) =$?, and $T^f \models_{TEL_3} \phi = \top$. Therefore,*

$$T^f \models^O_{TEL_3} \phi = \top \;\not\Rightarrow\; P_{TEL}(\phi, T^f) = \top$$
$$P_{TEL}(\phi, T^f) = ? \;\not\Rightarrow\; T^f \models^O_{TEL_3} \phi \;= ?$$

Example 3 (**GF** p)**.** *Let $\phi =$ **GF** p, $O = \emptyset^\omega$ and $T^f = \emptyset$, then $P_{TEL}(\phi, T^f) =$?, and $T^f \models_{TEL_3} \phi = \bot$. Therefore,*

$$T^f \models^O_{TEL_3} \phi = \bot \;\not\Rightarrow\; P_{TEL}(\phi, T^f) = \bot,$$
$$P_{TEL}(\phi, T^f) = ? \;\not\Rightarrow\; T^f \models^O_{TEL_3} \phi \;= ?$$

We also notice that both the 3-value THT and TEL logic introduced in Definitions 4 and 7 take the observation trace into account, but the respective definitions of progression 5, 8 do not. The reason is that we are progressing a prefix of a trace and we assume to have the observations already encoded in the trace. Given a finite trace $I^f = \langle H^f, T^f \rangle$, we extend the definition of $<_O$ over finite prefixes by $H^f <_O T^f$ if for each $i = 0, \ldots, k-1$, $O_i \subseteq H_i$ and exists $i \in \{0, \ldots, k-1\}$ such that $H_i \subset T_i$. If we replace $H^f \subset T^f$ with $H^f <_O T^f$ in Definition 8, Theorem 2 still holds.

5 Computing the Intersection of TEL Traces

In this section, we describe an algorithm to compute online the intersection of all the equilibrium traces of a given temporal program π, and an incoming trace of observations O. During the computation, it may happen that the truth value of one atom is determined not only by other non-future atoms, but also by future atoms, e.g., `p:- X(q).`. In this case, if we are not able to determine the truth value of the atom, we may delay the computation, by pastifying an atom, i.e. adding a previous operator, obtaining, for instance, `Y(p) :- q.`. Furthermore, we may instantiate some dynamic or fulfillment rules, adding them to the initial part of the temporal program. Therefore, during the computation, we may deviate from the Definition 3.

In order to clearly define the set of atoms whose truth value we can compute, we proceed by introducing the dependency graph for programs with possibly disjunctive rules.

Definition 9 (Dependency graph)**.** *The* dependency graph *of a logic program π is the directed graph $DG_\pi = \langle N, E \times \{+, -\}\rangle$ where (i) each atom of π is a node in N, (ii) there is a positive (resp. negative) arc in E from a node a to a node b if $a \in H(r)$ and $b \in B^+(r)$ (resp. $b \in B^-(r)$) for some rule r in π, and (iii) for every rule r in π and $a \neq b \in H(r)$, there is a positive arc from a to b in E.*

Given the definition of the dependency graph, we can now introduce the following concept.

Definition 10 (Negative dependency)**.** *Given a logic program π, and its corresponding dependency graph DG_π, we say that an atom p* depends negatively *on q if there exists a path from p to q passing through a negative arc.*

Furthermore, we say that a set U of atoms *is closed under negative dependencies* if $q \in U$ holds for every atom q such that some atom $p \in U$ negatively depends on q.

Example 4. *If we apply Definition 9 to the following logic program, we obtain the dependency graph in Figure 1.*

$$\neg\, next(b) \to a, \quad a \vee b, \quad next(b)$$

Furthermore, by applying Definition 10, we see that $U = \emptyset$ is the only set closed under negative dependencies not containing $next(b)$, as both a and b depend negatively on $next(b)$.

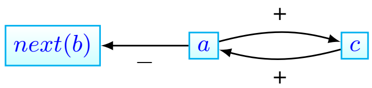

Figure 1: Dependency graph of the program of Example 4

In order to compute the intersection of all the equilibrium traces with a stream of incoming observations, we resort to some splitting techniques to decompose the program into a *lower* and an *upper* part. Intuitively, the lower part refers to the part of the program related to the current and the past states, while the upper part is related to future states. As we are interested in deriving as many facts as soon as possible, we extend the Splitting Theorem. We define two new functions that resemble the e_U and b_U functions of the Splitting Theorem, respectively $filter$ and $progress$. Instead of requiring U to be a splitting set, we have the more relaxed requirement of being a set closed under negative dependency. We first define $filter$.

Definition 11 (Filter)**.** *Given a logic program π and $U \subseteq \mathcal{P}$, we let $filter(\pi, U)$ consist of all rules $r \in \pi$ that contain only atoms from U and where each negative literal $\neg p$ occurring in r depends only on atoms in U.*

We are now ready to introduce the reduction $skep\_prog(\pi, C, B, U)$. Besides the program π, it takes as input a set C of atoms that are considered to be proved, a set B of atoms that may possibly be proved, and a set U of atoms closed under negative dependencies.

Definition 12 (Skeptical Progress)**.** *Let us consider a logic program π, and $C, B, U, U^s \subseteq \mathcal{P}$, where U^s is the maximal splitting set of π contained in U. Let us define $skep\_prog(\pi, C, B, U) = \pi'$ to be obtained by removing all the rules $r \in \pi$, such that either*

$i)$ there is a $p \in H(r)$ such that $p \in C$,
$ii)$ there is a $p \in B^+(r)$ such that $p \in U^s \setminus B$, or
$iii)$ there is a $p \in B^-(r)$ such that $p \in C$.

From each remaining rule r, all positive literals $p \in$ are deleted, and all negated literals $\neg p$ such that $p \in U^s \setminus B$ are deleted.

The following theorem says that by a proper application of the filter function on the progress function, we are able to compute an answer set of the program π. The intuition behind Theorem 3 is that given U, you can filter rules that are defeasible with respect to atoms in U only, where a rule is defeasible if its body depends on some negated by default literal.

Theorem 3 (Non-Defeasible Splitting Theorem)**.** *Let U be a set of atoms closed under negative dependencies w.r.t. a logic program π. Then π has an answer set Z only if $Z = X \cup Y$ such that*

- *X is an answer set of $filter(\pi, U)$ and*
- *Y is an answer set of $skep\_prog(\pi, X, X, U)$.*

The converse holds if each rule $r \in filter(\pi, U)$ with disjunctive head satisfies $H(r) \cap (U \setminus U^s) = \emptyset$, where U^s is the maximal splitting set of π contained in U.

If we compare Theorem 3 with the well-known Splitting Theorem [24], it is interesting to stress that Theorem 3 uses a notion of being closed *under defeasiblity*, while the Splitting Theorem requires the splitting set to be closed *under definition*, i.e., for each atom p in splitting set S, all atoms occurring in a rule r that defines p should be contained in S as well. Notably, for normal (disjunction-free) logic programs Theorem 3 extends the Splitting Theorem, providing an if and only if characterizations of answer sets.

As we are interested in the computation of the intersection of equilibrium traces, we exploit Theorem 3 to prove the following.

Theorem 4 (Skeptical Non-Defeasible Splitting Theorem)**.** *Suppose $U \subseteq \mathcal{P}$ is closed under negative dependencies w.r.t. a logic program π. Let C and C_X (B and B_X) be the skeptical (brave) consequences of π and $filter(\pi, U)$, respectively, and let C_Y (B_Y) be the skeptical (brave) consequences of $skep\_prog(\pi, C_X, B_X, U)$. Then $C_X \cup C_Y \subseteq C$ and $B \subseteq B_X \cup B_Y$.*

We remark that for normal logic programs π, we in fact can show that in Theorem 4 $C_X \cup C_Y = C$ and $B = B_X \cup B_Y$ holds, i.e. the cautious and brave conclusions remain invariant under progression.

Even if the results just stated are related to logic programs in general, in our framework we are interested in selecting U as the largest set of atoms closed under negative dependency containing only past atoms or present atoms, currently appearing in the temporal program. I.e., atoms preceded only by a non-negative number of previous operators, that is $Y^i p$ with $i \geq 0$. For convenience, let us introduce the notation $history\_atoms(\pi_{init}) = \{\mathbf{Y}^i p \mid \exists r \in \pi_{init}$ and $\mathbf{Y}^i p \in lits^+(r) \cup lits^-(r), i \geq 0\}$ for the past time literals in π_{init}.

Now we have all terminology to present our reasoning algorithm (Algorithm 1). At the very beginning, it copies the initial segment of the temporal program in π_i (line 2), then it instantiates rules from the dynamic part and from the fulfillment part via the function $inst$, adding them to the initial part of the program (line 4).

Next, it computes U_i, the set of current and past atoms such that they are closed under negation (line 5). It filters out the program π_i using U as the filtering parameter, obtaining a new filtered program π_f (line 6). If there is no *local answer set*, instability is detected and it is signaled to the user (line 8). Otherwise, if π_f admits a stable model, its skeptical and brave consequences (respectively C_i, and B_i) are computed, and the skeptical consequences C_i are fed into the output trace (lines 10-12). Once C_i and B_i are available, they can be used to simplify the program via the application of *skep_prog* (line 13). The intuition of (line 14) is to add a previous operator before atoms appear in the initial segment (see Defn. 13 for details).

Algorithm 1 Main algorithm. Input: π, O. Output: Skeptical trace

```
1:  i := 0
2:  π_c^i := π_init
3:  while ⊤ do
4:      π_c^i := π_c^i ∪ inst(π_dyn) ∪ inst(π_ful) ∪ O_i
5:      U_c^i := history_atoms(π_c^i) \ get_neg_deps(π_c^i)
6:      π_f := filter(π_c^i, U_c^i)
7:      if π_f does not admit any answer set then
8:          noStableTraceError
9:      end if
10:     C_i := skeptical_conseq(π_f)
11:     B_i := brave_conseq(π_f)
12:     feed_skeptical_trace(C_i)
13:     π_c^i := skep_prog(π_c^i, C_i, B_i, U_i)
14:     π_c^{i+1} := pastify(π_c^i)
15:     i := i + 1
16: end while
```

Definition 13 (Pastify)**.** *Given a temporal program of form $\pi = \pi_{init}$, $pastify(\pi)$ results by rewriting its rules as follows: rewrite*

- $\bigvee_{k=1}^{m} \mathbf{Y}^{i_k} b_k \bigvee_{k=m+1}^{n} \neg \mathbf{Y}^{i_k} b_k \to \bigvee_{k=1}^{l} \mathbf{Y}^{j_k} c_k$ *to* $\bigvee_{k=1}^{m} \mathbf{Y}^{i_k+1} b_k \bigvee_{k=m+1}^{n} \neg \mathbf{Y}^{i_k+1} b_k \to \bigvee_{k=1}^{l} \mathbf{Y}^{j_k+1} c_k$;
- $\mathbf{Y}^i q \to \mathbf{Y}^i p \vee \ldots p \vee \mathbf{F} p$ *to* $\mathbf{Y}^{i+1} q \to \mathbf{Y}^i p \vee \ldots p \vee \mathbf{F} p$; *and*
- $\mathbf{G} q \wedge q \wedge \ldots \mathbf{Y}^i q \to \mathbf{Y}^i p$ *to* $\mathbf{G} q \wedge q \wedge \ldots \mathbf{Y}^{i+1} q \to \mathbf{Y}^{i+1} p$.

In what follows, we present different results with the aim of showing that the algorithm computes an approximation of the intersection of all equilibrium traces of the input program π modulo observations O. For simplicity, we assume that the observations added in Algorithm 1 as facts at run-time are already encoded in the input program. Set operations such as intersection, union, and set minus over traces must be considered state-wise.

Definition 14 (Unfolding)**.** *Given a temporal program π and $k \geq 0$, the temporal program $unfold(\pi, k)$ contains (i) all rules in π_{init} and (ii) for each rule $r = \bigwedge_{j=1}^{k} b_j \wedge \bigwedge_{j'=k+1}^{n} \neg b_{j'} \to \bigvee_{h=1}^{l} c_h$, where all b_js, $b_{j'}$ and c_h are positive temporal literals, in $\pi_{dyn} \cup \pi_{ful}$ the rules $r_i = r[\mathbf{X}^i]$, for each $0 \leq i \leq k$, where*

$$r[\mathbf{X}^i] = \bigwedge_{j=1}^{k} \mathbf{X}^i b_j \wedge \bigwedge_{j'=k+1}^{n} \neg \mathbf{X}^i b_{j'} \to \bigvee_{h=1}^{l} \mathbf{X}^i c_h$$

Notice that $unfold(\pi, k)$ can be viewed as a non-temporal ASP program that contains a set of atoms from $\mathcal{P}^k = \{\mathbf{X}^i p : p \in \mathcal{P}$ for $i = 0, \ldots, k\}$. Given a set $T^k \subseteq \mathcal{P}^k$, we define $trace(T^k)$, as the prefix of a trace starting from T^k, as follows. If $p \in T^k$, then $p \in trace(T^k)_0$. And, if $\mathbf{X}^i p \in T^k$, then $p \in trace(T^k)_i$ for each $0 < i \leq k$. Using this notation, we can introduce the limit version for $k \to \infty$, obtaining $\mathcal{P}^\omega$, $\pi^\omega = unfold(\pi)^\omega$, where $unfold(\pi) = unfold(\pi, 1)$. Furthermore, if T^ω is a set of atoms in π^ω, then $trace(T^\omega)$ is the corresponding infinite trace. In order to simplify notation, we will use $T \equiv trace(T^\omega)$.

Theorem 5 (Trace Equivalence)**.** *Let π be a temporal program without fulfillment rules. Then T^ω is a stable model for π^ω iff $trace(T^\omega)$ is an equilibrium trace for π. Furthermore, if in the latter case $trace(T^\omega) \models \pi_{ful}$ for a set π_{ful} of fulfillment rules, then $trace(T^\omega)$ is an equilibrium trace for $\pi \cup \pi_{ful}$.*

Let $\pi^\omega = unfold(\pi)^\omega$ be an unfolded temporal program π, $\pi_p^0 = \pi^0$ and U^i maximal subset of $\cup_{j=0,\ldots,i} \mathcal{P}^j$ closed under negated dependencies in π_p^i. Let us denote by T^i a generic answer set of $filter(\pi_p^i, U^i)$. We define $\pi_p^{i+1} = skep\_prog(\pi^i, T^i, T^i, U^i)$ for some T^i non-deterministically chosen. Then,

Theorem 6 (Sequence Non-Defeasible Splitting)**.** *Let π be a temporal program without fulfillment rules. If π admits a temporal equilibrium model T, then $T = trace(\cup_{i \geq 0} T^i)$, for some sequence $T^0, T^1, \ldots$ is an equilibrium trace of π. Furthermore, if $T \models \pi_{ful}$ for some set π_{ful} of fulfillment rules, then $T \models \pi \cup \pi_{ful}$.*

Theorem 5 and 6 just presented pertain to properties that establish a relationship between models of the unfolded program π^ω and those of the original one π, as well as a property about splitting a temporal program using the newly introduced concept of Non-Defeasible Splitting. However, in certain cases, it may be possible to observe a finite number of equilibrium traces and a large number of local answer sets at each step of the computation. Consequently, determining a single trace would require an accurate guess of the T^i stable model at time step i to be utilized during the program's evolution. Since a non-deterministic choice is involved, some backtracking procedures would also be required. To address this challenge, our proposed algorithm aims to compute an approximation of the unique intersection of all the equilibrium traces, instead. Theorem 5 and 6 can be used to prove their skeptical counterpart.

Denote by $AS(\pi)$ the set of all stable models of a logic program π.

Theorem 7 (Skeptical Equivalence)**.** *Let π be a temporal program without fulfillment rules. If π admits an equilibrium trace, then* $trace(\bigcap AS(\pi^\omega)) = \bigcap TEL(\pi)$.

Let π^ω be the unfolded temporal program π. $\pi^0_\cap = \pi^0$ and U^i maximal subset of $\cup_{j=0,\dots,i}\mathcal{P}^j$ closed under negative dependencies in $\pi^i_\cap$. Let us denote by C^i, and B^i, respectively, the skeptical and the brave consequences of $filter(\pi^i_p, U^i)$. We define $\pi^{i+1}_\cap = skep\_prog(\pi^i_\cap, C^i, B^i, U^i)$. Then,

Theorem 8 (Sequence Skeptical Non-Defeasible Splitting)**.** *Let π be a temporal program where $\pi_{ful} = \emptyset$. If π has an equilibrium trace,*

$$trace(\textstyle\bigcup_{i\geq 0}\bigcap AS(\pi^i_\cap)) \subseteq \bigcap TEL(\pi) \textit{ and}$$
$$trace(\textstyle\bigcup_{i\geq 0}\bigcup AS(\pi^i_\cap)) \supseteq \bigcup TEL(\pi).$$

We can extend Theorems 7 and 8, which are already applicable to a temporal program $\pi = \pi_{init} \cup \pi_{dyn}$, to include also a set of fulfillment rules π_{ful} under some assumption. Syntactically, (i) for each rule $r \in \pi_{ful}$ such that $p \in H(r)$ or $\mathbf{F}p \in H(r)$, p can occur only in heads of π, or with r added, would not feed back to the component in which q occurs in a modular program decomposition such as program splitting. Otherwise, (ii) for $\mathbf{F}p \in H(r)$, we may instead require the observation trace O to be *fair* with respect to p, i.e., infinitely many observations of p must occur. Under such constraints, if $trace(T) \models_{TEL} \pi_{ful}$ holds for each $T \in AS(\pi^\omega)$, then we can replace $TEL(\pi)$ with $TEL(\pi \cup \pi_{ful})$ in both Theorems 7 and 8. Intuitively, case (i) holds because p can be proved at the very last step and its truth value does not affect the other atoms in the answer set. Case (ii) holds because thanks to *fairness* on the observation, if we prove $\mathbf{F}p$, we do not have to do any guess where to add p in the trace, but just wait for the next occurrence in O thanks to minimality.

In Theorems 5–8 we deal with the π_{ful} part of a program differently from the π_{init} and π_{dyn} parts. Let us consider the following simple program $\pi = \pi_{init} \cup \pi_{dyn} \cup \pi_{ful}$, where $\pi_{init} = \{p\}$, $\pi_{dyn} = \{\neg p, q \to \bot\}$ and $\pi_{ful} = \{p \to \mathbf{F}q\}$. There is only one equilibrium trace for π, viz. the trace $\{p, q\}, \emptyset^\omega$, which trivially coincides with the intersection of all equilibrium traces. However, if we proceed as in Algorithm 1, we only compute the local intersection and union of all the stable models, but we never make a guess where to fulfill $\mathbf{F}q$. Therefore, we can fulfill in our approach a promise $\mathbf{F}q$ only if we derive q by some other rule in the initial or the dynamic part. On the other hand, by admitting some fulfillment rules more expressive possibilities are offered.

We point out that we cannot apply Theorem 8 directly to Algorithm 1 because it refers to the unfolded version of the input program π^ω, while in Algorithm 1 rules are added at runtime and the state-counter i is incremented at each step. However, we can still exploit Theorem 8 as it is not hard to see that for each rule r, we have $I, i \models r \iff I, i+1 \models pastify(r) \iff I, 0 \models \mathbf{X}^i r$.

6 Case study: Temporal Strategic Companies

Strategic Companies [5] is a well-known Σ^2_p-complete problem that has been used for systems comparisons, also in ASP competitions [17, 20]. In the original Strategic Companies problem, a collection $C = c_1, ..., c_m$ of companies is given. Each company c_i produces some goods from a set G of goods and is possibly controlled by a set $W_i \subseteq C$ of owner companies, for each $i = 1, \dots, m$. A set $C' \subseteq C$ is a "strategic set" of companies if it is $\subseteq$-minimal among all sets such that (1) the companies in C' produce all goods in G, and (2) if $W_i \subseteq C'$, all companies c_i owned by W_i must belong to C'.

For the presentation of this problem, we will assume that (i) each product is produced by at most two companies, and (ii) each company is controlled by at most two companies.

That product $g \in G$ is produced by c_0 and c_1 is represented by $prBy(p, c_0, c_1)$, that c_0 and c_1 control c by $ctrBy(c, c_0, c_1)$., and that c_i belongs to the strategic set by $str(c_i)$.

With this notation, we can now encode the previous conditions:

$$prBy(P, C_0, C_1) \to str(C_0) \ \vee \ str(C_1),$$
$$ctrBy(C, C_0, C_1) \wedge str(C_0) \wedge str(C_1) \to str(C).$$

A company c_i is *unnecessary*, $unn(c_i)$, if it does not belong to the strategic set; this is expressed by

$$\neg str(C) \to unn(C).$$

Since the set of ownerships and the set of companies producing each good can change over time, in this example, we want to reason whether two given companies can be unnecessary in two consecutive time steps. For this purpose, we will include the corresponding property for each company consisting of the following two rules:

$$unn(c_i) \wedge \mathbf{X}(unn(c_i)) \to prop(c_i),$$
$$\neg prop(c_i) \to negProp(c_i).$$

By introducing $prop(c_i)$, we are able to identify which property has been violated. This is a difference from usual LTL monitoring, in which one single automaton combining all the properties is built, which does not allow distinguishing between violations.

Unless new information is given, ownership and productions remain unchanged. This *inertia* principle can be encoded as follows:

$$prBy(P, C_0, C_1) \wedge \neg\mathbf{X}(chg(pr(P))) \to \mathbf{X}(prBy(P, C_0, C_1)),$$
$$prBy(P, C_0, C_1), \mathbf{X}(prBy(P, C_2, C_3)) \to \mathbf{X}(chg(pr(P))),$$

where $C_0 \neq C_2$ or $C_1 \neq C_3$; similarly for ownerships.

We may monitor temporal properties on the observations, such as:

$$prBy(p, c_0, c_1) \to \mathbf{F}(prBy(p, c_0, c_0) \vee prBy(p, c_1, c_1)), \quad \text{(P1)}$$
$$prBy(p, c_0, c_1) \to$$
$$prBy(p, c_0, c_1) \ \mathbf{U} \ (prBy(p, c_0, c_0) \vee prBy(p, c_1, c_1)) \quad \text{(P2)}$$

Property P1 requires that if the product p is produced by c_0 and c_1 at a certain point, that product will eventually only be produced by one of the two companies. Property P2 indicates a convergence in the companies producing a good. That is, if product p is produced by companies c_0 and c_1, they must keep producing it until it is only produced by one of them. While we omitted the until operator $\mathbf{U}$ here, it can be expressed in the normal form using the $\mathbf{X}$ and $\mathbf{F}$ operators and auxiliary predicates. As already mentioned after Theorem 8, fairness in the observations is assumed to ensure the approach remains valid.

We encoded this problem with the presented properties in an open-source prototype that uses the Clingo [19, 25] API for Python. A user can input a temporal logic program and enter new observations at each time step, as well as translate a TEL formula into normal form. The number of companies (3) and products (2) was not changed for the different tests as it did not affect the patterns we observed. The property related to a company being unnecessary in two consecutive steps was also successfully included but it did not affect the number of rules. The results of the tests are shown in Figure 2.

In the the case of property P1, the initial state fires the rule of the property, making the program wait for the atom inside of the $\mathbf{F}$. As an optimization, fulfillment rules of type $\mathbf{G}(p \rightarrow \mathbf{F}q)$ are not instantiated when the rule has been fired in the past and q does not depend on the future. Without optimization, as the rule by inertia fired in every step the number of rules keeps growing until an observation arrives that makes the head of the rule true (step 40). If no observation would have arrived, the number of rules would have kept growing. In step 42, a new observation arrives making the body of the rule true, which makes the number of rules start to grow again.

The traces of observations to test property P2 consisted in (T1) no observations, (T2) one observation at step 2 that makes the head of the rule true, and (T3) like (T2) but for every step. Since some of the rules in the program are simplified when the property is satisfied, we can see in Figure 2 that with the trace (T3), the number of rules is lower than with the other two traces. Since (T1) and (T2) are very similar, their respective program evolutions are almost overlapped.

The number of rules keeps growing at each step due to the instantiation of the dynamic and fulfillment rules. This is because the normal form from [2] results in future dependencies, which does not allow us to simplify the program at each step as it happened with property P1.

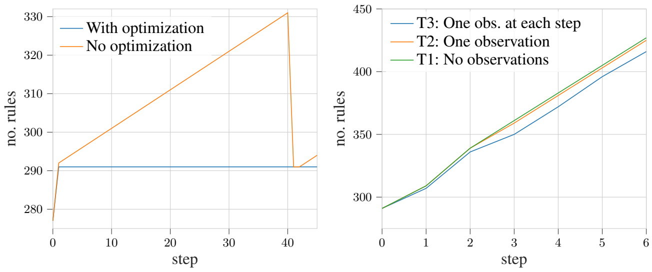

Figure 2: Evolution of the case study with property P1 containing the $\mathbf{F}$ operator (left) and property P2 containing the $\mathbf{U}$ operator (right).

7 Related Work

In the field of AI, there are other related approaches to reason about temporal properties, particularly in the context of stream reasoning. Streamlog [32] is a temporal Datalog language where the specification is expressed as a set of rules with time-stamped predicates, similar to the work in [29]. It is also interesting to point out some syntactic restrictions introduced in [32]. Requiring time stamps of the head to be greater than the time stamps of the body gives the possibility to easily compute a local stratification and obtain then a unique trace. We could investigate further these restrictions to use our algorithm for trace generation purposes, too. In particular, given a locally stratified program π_{init}, our algorithm eventually returns a prefix T^f such that $P_{TEL}(\pi_{init}, T^f) = \top$. Therefore, by application of Theorem 2, $T^f \vDash_{TEL_3} \pi_{init} = \top$.

Other works such as LARS [10, 9] provide streaming reasoning capabilities with rules using temporal operators such as `always` and `eventually` (but not `until` or `release`) that are evaluated within a finite window of time points. In contrast, our approach is geared to all temporal operators of LTL [28].

The solver TELINGO [15] also deals with TEL/THT, but is different from our approach in several respects. First, only it handles finite equilibrium traces, while our algorithm computes the intersection of infinite TEL traces online. TELINGO is more geared to tasks like planning with a finite horizon, while our algorithm addresses the monitoring problem. Furthermore, TELINGO has some syntactic constraints such as disallowing future operators in rule bodies and past operators in rule heads. In our approach, instead, future operators can occur both in heads and bodies. Another difference is that TELINGO uses incremental reasoning in ASP, while we use progression-based monitoring by rewriting the formula to be monitored at each step.

STELP [14] is another ASP solver for TEL that addresses a restricted class of temporal logic programs called "*Splittable temporal logic programs*" [4]. It handles temporal operators like `always` and `until` as constraints (the head of their rule is empty): this is helpful when one wants to discard particular TEL models. Instead, under some restrictions we can use `always` and `until` also for generating models (the head of the rule can also be non-empty).

In [13], the authors provide an approach to construct a Büchi automaton accepting TEL models. However, it is particularly computationally expensive (due to EXPSPACE-completeness of TEL satisfiability) and to the best of our knowledge has not yet been implemented. Another important difference with the automata-based approach is that in our approach it is easier to monitor multiple formulas in parallel and we can provide an explanation of the violation, by identifying the subformula responsible for it.

Our approach takes inspiration from the rewriting-based approach for runtime verification proposed first in [30], where the authors employed the Maude system [18] as a rewriting logic engine to implement LTL rewriting rules. Later, the authors of [7] leveraged a progression-based monitoring approach for LTL formulas using rewriting in the context of decentralized monitoring. A problem in this setting is that to satisfy the LTL specification, each node may need to know at a certain moment whether an event/proposition has occurred in another node before the next synchronization step. This problem is solved by rewriting the formula using a past operator in front of the formula when it contains propositions controlled by other nodes: in this case, the verdict is delayed until the synchronization with the other nodes occurs. We also use this trick but in a completely different context. In the ASP setting implication is interpreted differently than in LTL. For example, if we have a formula $\mathbf{X}p \rightarrow q$ in the classical interpretation corresponds to $\neg\mathbf{X}p \vee q$ and will result in the verdict true if q occurs at the current moment or will be rewritten in the obligation $\neg p$ to be held in the next step. However, in the ASP setting, it is not enough to have q true in the current moment, as the truth of q must be justified, which fails if $\mathbf{X}p$ is false in the future; this is why the body is rewritten as $\mathbf{Y}q$ using the past operator.

8 Conclusion and Future Work

In this paper, we presented a novel approach to temporal reasoning using the progression technique for Temporal Here and There and Temporal Equilibrium Logic. Our approach allows for non-monotonic reasoning over a trace of observations, providing the means to compute logical consequences of a temporal knowledge base over time. We presented the theoretical foundations to apply progression to these logics and proposed an algorithm to monitor dynamic systems that has been implemented as a proof-of-concept.

Our work contributes to the growing interest in the application of temporal reasoning approaches and non-monotonic logics. By using the progression technique, we have shown that it is possible to go beyond the usual LTL online computation and provide a more expressive approach to temporal reasoning. This can be useful in various domains, such as robotics, control systems, and autonomous vehicles, where real-time monitoring and decision-making are crucial.

In addition, we identified some future lines of work to improve and extend our approach. These include the inclusion of explicit negation and assumptions, the use of variables in the implementation, consistency analysis, optimization of the algorithm, and the encoding of norms and exceptions. Furthermore, the usage of paraconsistent programs could be an approach to include assumptions. We will also apply this technique in a real-world case study to further validate our approach. Overall, our findings show the potential for our approach to be used in a variety of dynamic systems, paving the way for future research in this area.

References

[1] Felicidad Aguado, Pedro Cabalar, Martín Diéguez, Gilberto Pérez, Torsten Schaub, Anna Schuhmann, and Concepción Vidal, 'Linear-time temporal answer set programming', *Theory Pract. Log. Program.*, **23**(1), 2–56, (2023).

[2] Felicidad Aguado, Pedro Cabalar, Martín Diéguez, Gilberto Pérez, and Concepción Vidal, 'Temporal equilibrium logic: a survey', *Journal of Applied Non-Classical Logics*, **23**(1-2), 2–24, (2013).

[3] Felicidad Aguado, Pedro Cabalar, Gilberto Pérez, and Concepción Vidal, 'Strongly equivalent temporal logic programs', in *Proc. of JELIA 2008: the 11th European Conference on Logics in Artificial Intelligence*, pp. 8–20. Springer, (2008).

[4] Felicidad Aguado, Pedro Cabalar, Gilberto Pérez, and Concepción Vidal, 'Loop formulas for splitable temporal logic programs', in *Logic Programming and Nonmonotonic Reasoning*, eds., James P. Delgrande and Wolfgang Faber, pp. 80–92, Berlin, Heidelberg, (2011). Springer Berlin Heidelberg.

[5] Mario Alviano, Marco Maratea, and Francesco Ricca, 'Strategic Companies', in *ASP Competition 2013*, eds., UniversitÃă della Calabria and TU Wien, Knowledge-Based Systems Group - TU Wien, (2013).

[6] Fahiem Bacchus and Froduald Kabanza, 'Planning for temporally extended goals', *Annals of Mathematics and Artificial Intelligence*, **22**, 5–27, (1998).

[7] Andreas Bauer and Ylies Falcone, 'Decentralised LTL monitoring', in *Proc. of FM 2012: the 18th International Symposium on Formal Methods*, pp. 85–100. Springer, (2012).

[8] Andreas Bauer, Martin Leucker, and Christian Schallhart, 'The good, the bad, and the ugly, but how ugly is ugly?', in *Proc. of RV 2007: the 7th International Workshop on Runtime Verification*, pp. 126–138. Springer, (2007).

[9] Harald Beck, Minh Dao-Tran, and Thomas Eiter, 'Equivalent stream reasoning programs', in *Proc. IJCAI 2016: the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, p. 929âĂŞ935. AAAI Press, (2016).

[10] Harald Beck, Minh Dao-Tran, Thomas Eiter, and Michael Fink, 'LARS: A logic-based framework for analyzing reasoning over streams', in *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, eds., Blai Bonet and Sven Koenig, pp. 1431–1438. AAAI Press, (2015).

[11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski, 'Answer set programming at a glance', *Commun. ACM*, **54**(12), 92–103, (2011).

[12] Pedro Cabalar, 'A normal form for linear temporal equilibrium logic', in *Proc. of JELIA 2010: the 12th European Conference on Logics in Artificial Intelligence*, pp. 64–76. Springer, (2010).

[13] Pedro Cabalar and Stéphane Demri, 'Automata-based computation of temporal equilibrium models', in *Proc. of LOPSTR 2011: the 21st International Symposium on Logic-Based Program Synthesis and Transformation*, pp. 57–72. Springer, (2012).

[14] Pedro Cabalar and Martín Diéguez, 'STeLP–a tool for temporal answer set programming', in *Proc. of LPNMR 2011: the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 370–375. Springer, (2011).

[15] Pedro Cabalar, Roland Kaminski, Philip Morkisch, and Torsten Schaub, 'telingo= ASP + time', in *Proc. of LPNMR 2019: the 15th International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 256–269. Springer, (2019).

[16] Marco Cadoli, Thomas Eiter, and Georg Gottlob, 'Default logic as a query language', *IEEE Trans. Knowl. Data Eng.*, **9**(3), 448–463, (1997).

[17] Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca, 'Design and results of the fifth answer set programming competition', *Artificial Intelligence*, **231**, 151–181, (2016).

[18] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Jose F. Quesada, 'Maude as a metalanguage', in *Proc. of WRLA 1998: the 1998 International Workshop on Rewriting Logic and its Applications*, eds., Claude Kirchner and Hélène Kirchner, volume 15 of *Electronic Notes in Theoretical Computer Science*, pp. 147–160. Elsevier, (1998).

[19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub, 'Multi-shot asp solving with clingo', *Theory and Practice of Logic Programming*, **19**(1), 27âĂŞ82, (2019).

[20] Martin Gebser, Marco Maratea, and Francesco Ricca, 'The sixth answer set programming competition', *Journal of Artificial Intelligence Research*, **60**, 41–95, (09 2017).

[21] Arend Heyting, 'Die formalen Regeln der intuitionistischen Logik', *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-Mathematische Klasse*, 42–56, (1930).

[22] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello, 'The DLV system for knowledge representation and reasoning', *ACM Trans. Comput. Log.*, **7**(3), 499–562, (2006).

[23] Hector J Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B Scherl, 'Golog: A logic programming language for dynamic domains', *The Journal of Logic Programming*, **31**(1-3), 59–83, (1997).

[24] Vladimir Lifschitz, 'Answer set planning', in *Proc. of the 1999 International Conference on Logic Programming*, ed., Danny De Schreye, pp. 23–37. MIT Press, (1999).

[25] Vladimir Lifschitz, *Answer Set Programming*, Springer, 2019.

[26] Vladimir Lifschitz and Hudson Turner, 'Splitting a logic program', in *Proc. of the Eleventh International Conference on Logic Programming*, ed., Pascal Van Hentenryck, pp. 23–37. MIT Press, (1994).

[27] David Pearce, 'Equilibrium logic', *Annals of Mathematics and Artificial Intelligence*, **47**(1-2), 3, (2006).

[28] Amir Pnueli, 'The temporal logic of programs', in *Proc. of FOCS 1977: the 18th Annual Symposium on Foundations of Computer Science*, pp. 46–57. IEEE Computer Society, (1977).

[29] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks, 'Stream reasoning in temporal datalog', in *Proc. of the 32nd AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, (2018).

[30] Grigore Rosu and Klaus Havelund, 'Rewriting-based techniques for runtime verification', *Autom. Softw. Eng.*, **12**(2), 151–197, (2005).

[31] Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev, 'DatalogMTL over the integer timeline', in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, eds., Diego Calvanese, Esra Erdem, and Michael Thielscher, pp. 768–777, (2020).

[32] Carlo Zaniolo, 'Logical foundations of continuous query languages for data streams', in *Datalog in Academia and Industry: Second International Workshop, Datalog 2.0, Vienna, Austria, September 11-13, 2012. Proceedings*, pp. 177–189. Springer, (2012).

Appendix. Proofs

In order to simplify the proofs, we introduce the following Lemma that is a direct consequence of the Definition 1 of satisfaction for the THT logic.

Lemma 1 (Derived THT-Satisfaction)**.** *We can derive the following cases for the THT-Satisfaction:*

- $\langle H, T\rangle, k \vDash \mathbf{G}\,\phi$ *iff* $\langle H, T\rangle, k \vDash \phi$ *and* $\langle H, T\rangle, k+1 \vDash \mathbf{G}\,\phi$
- $\langle H, T\rangle, k \vDash \mathbf{F}\,\phi$ *iff* $\langle H, T\rangle, k' \vDash \phi$ *for some* $k' \geq k$

Furthermore, if we admit the implication symbol $\rightarrow_c$ *in the grammar (1) and extend the semantics of THT with the following entry:*

- $\langle H, T\rangle, k \vDash \phi \rightarrow_c \psi$ *iff* $\langle T, T\rangle, k \nvDash \phi$*, or* $\langle T, T\rangle, k \vDash \psi$
- *then,*
$$\langle H, T\rangle, k \vDash \phi \rightarrow \psi \text{ iff } \begin{cases} \langle H, T\rangle, k \vDash \phi \rightarrow \psi, \text{ and} \\ \langle H, T\rangle, k \vDash \phi \rightarrow_c \psi \end{cases}$$

Hereon, if not stated otherwise, we are going to use the extended semantics introduced in Lemma 1. Furthermore, since temporal THT formulas can be rewritten into a strongly equivalent formula with only $\mathbf{X}$, $\mathbf{G}$, and $\mathbf{F}$ as temporal operators, we will not consider operators like $\mathbf{U}$ or $\mathbf{R}$ in the following proofs, in order to keep the explanation closer to the temporal normal form used in our algorithm.

Let us introduce the following notation. Given a finite trace $I^f_{[i,\ldots,j]} = \langle H^f, T^f\rangle_{[i,\ldots,j]} = \langle H_i, T_i\rangle, \langle H_{i+1}, T_{i+1}\rangle, \ldots, \langle H_j, T_j\rangle$, where I^f is of length k and $0 \leq i \leq j < k$. The proof of the following Lemma is omitted since it follows directly from Definition 5. We just observe that the requirement of not having future operators $\{\mathbf{X}, \mathbf{F}, \mathbf{G}\}$ as an argument of the previous operator $\mathbf{Y}$ is needed.

Lemma 2. *Let us consider the prefix* I^f *of length* $k+1$*, the following properties hold for* $0 \leq i \leq k$ *and for all* $\phi \in \mathcal{L}$

$$P_{THT}(\bot, I^f_{[i\ldots k]}) = \bot \tag{4}$$

$$P_{THT}(p, I^f_{[i\ldots k]}) = \top \text{ if } p \in H^f_i \tag{5}$$

$$P_{THT}(p, I^f_{[i\ldots k]}) = \bot \text{ if } p \notin H^f_i \tag{6}$$

$$P_{THT}(\phi \vee \psi, I^f_{[i\ldots k]}) = P_{THT}(\phi, I^f_{[i\ldots k]}) \vee P_{THT}(\psi, I^f_{[i\ldots k]}) \tag{7}$$

$$P_{THT}(\phi \wedge \psi, I^f_{[i\ldots k]})) = P_{THT}(\phi, I^f_{[i\ldots k]}) \wedge P_{THT}(\psi, I^f_{[i\ldots k]}) \tag{8}$$

$$P_{THT}(\phi \rightarrow \psi, I^f_{[i\ldots k]}) = \begin{cases} P_{THT}(\phi, I^f_{[i\ldots k]}) \rightarrow P_{THT}(\psi, I^f_{[i\ldots k]}) \wedge \\ P_{THT}(\phi, \langle T^f, T^f\rangle_{[i\ldots k]}) \rightarrow_c P_{THT}(\psi, \langle T^f, T^f\rangle_{[i\ldots k]}) \end{cases} \tag{9}$$

$$P_{THT}(\mathbf{G}\,\phi, I^f_{[i\ldots k]}) = \bigwedge_{j=i,\ldots,k} P_{THT}(\phi, I^f_{[j\ldots k]}) \wedge\ \mathbf{G}\,\phi \tag{10}$$

$$P_{THT}(\mathbf{F}\,\phi, I^f_{[i\ldots k]}) = \bigvee_{j=i,\ldots,k} P_{THT}(\phi, I^f_{[j\ldots k]}) \vee\ \mathbf{F}\,\phi \tag{11}$$

$$P_{THT}(\mathbf{X}\,\phi, I^f_{[i\ldots k]}) = \begin{cases} \phi & \text{if } k = i, \text{ otherwise} \\ P_{THT}(\phi, I^f_{[i+1\ldots k]}) \end{cases} \tag{12}$$

$$P_{THT}(\mathbf{Y}\,\phi, I^f_{[i\ldots k]}) = \begin{cases} \bot & \text{if } i = 0, \text{ otherwise} \\ P_{THT}(\phi, I^f_{[i-1\ldots k]}) \end{cases} \tag{13}$$

Lemma 3. *Let* $I_{[i,\ldots,k]}$ *be a segment of a THT trace* I*,* $0 \leq i \leq k$*, and* $\phi \in \mathcal{L}$ *a THT formula. Then,*

$$I, i \vDash \phi \text{ iff } I, k \vDash P_{THT}(\phi, I_{[i,\ldots,k]})$$

Proof. We proceed by structural induction on the formula ϕ.

Consider the case $\phi = \bot$. Then, $I, i \nvDash \bot$ by Definition 1. Furthermore $P_{THT}(\bot, I_{[i,\ldots,k]}) = \bot$ by Lemma 2. Furthermore, we also have $I, k \nvDash \bot$ again by Definition 1.

We proceed in a similar way for $\phi = p$, where $p \in \mathcal{P}$. Then, by Definition 1

$$I, i \vDash p \text{ iff } p \in H_i$$

By Lemma 2 if $p \in H_0$, then

$$P_{THT}(p, I_{[i,\ldots,k]}) = \top$$

And by Definition 1

$$I, k \vDash P_{THT}(p, I_{[i,\ldots,k]})$$

The case $p \notin H_i$ is similar.

Let us first consider the case $\phi \vee \psi$ for the induction step. By Definition 1

$$\begin{aligned}
& I, i \vDash \phi \vee \psi \overset{\text{By Def. 1}}{\Longleftrightarrow} I, i \vDash \phi \text{ or } I, i \vDash \psi \\
& \underset{\text{By Ind. Hypothesis}}{\Longleftrightarrow} \begin{cases} I, k \vDash P_{THT}(\phi, I_{[i,\ldots,k]}), \text{ or} \\ I, k \vDash P_{THT}(\psi, I_{[i,\ldots,k]}) \end{cases} \\
& \underset{\text{By Definition 1}}{\Longleftrightarrow} I, k \vDash P_{THT}(\phi, I_{[i,\ldots,k]}) \vee P_{THT}(\psi, I_{[i,\ldots,k]}) \\
& \underset{\text{By Lemma 2}}{\Longleftrightarrow} I, k \vDash P_{THT}(\phi \vee \psi, I_{[i,\ldots,k]})
\end{aligned}$$

A similar argument can be used for $\phi \wedge \psi$.

Let us analyze the implication case.

$$\begin{aligned}
& I, i \vDash \phi \rightarrow \psi \overset{\text{By Def. 1}}{\Longleftrightarrow} \\
& \begin{cases} I, i \nvDash \phi \text{ or } I, i \vDash \psi \\ \langle T, T\rangle, i \nvDash \phi \text{ or } \langle T, T\rangle, i \vDash \psi \end{cases} \underset{\text{By Ind. Hypothesis}}{\Longleftrightarrow} \\
& \begin{cases} I, k \nvDash P_{THT}(\phi, I_{[i,\ldots,k]}) \text{ or } I, k \vDash P_{THT}(\psi, I_{[i,\ldots,k]}) \\ \begin{cases} \langle T, T\rangle, k \nvDash P_{THT}(\phi, \langle T_{[i,\ldots,k]}, T_{[i,\ldots,k]}\rangle) \text{ or} \\ \langle T, T\rangle, k \vDash P_{THT}(\psi, \langle T_{[i,\ldots,k]}, T_{[i,\ldots,k]}\rangle) \end{cases} \end{cases} \\
& \underset{\text{By Definition 1 and Lemma 1}}{\Longleftrightarrow} \\
& \begin{cases} I, k \vDash P_{THT}(\phi, I_{[i,\ldots,k]}) \rightarrow P_{THT}(\psi, I_{[i,\ldots,k]}) \\ I, k \vDash P_{THT}(\phi, \langle T_{[i,\ldots,k]}, T_{[i,\ldots,k]}\rangle) \rightarrow_c P_{THT}(\psi, \langle T_{[i,\ldots,k]}, T_{[i,\ldots,k]}\rangle) \end{cases} \\
& \underset{\text{By Lemma 2}}{\Longleftrightarrow} I, k \vDash P_{THT}(\phi \rightarrow \psi, I_{[i,\ldots,k]})
\end{aligned}$$

Let us consider the case $\mathbf{G}\,\phi$. We have that

$$\begin{aligned}
& I, i \vDash \mathbf{G}\,\phi \overset{\text{By Def. 1}}{\Longleftrightarrow} I, i \vDash \phi \text{ and } I, i+1 \vDash \phi \text{ and } \ldots I, k \vDash \phi \wedge \mathbf{G}\phi \\
& \underset{\text{By Ind. Hypothesis}}{\Longleftrightarrow} I, k \vDash P_{THT}(\phi, I_{[i,\ldots,k]}) \text{ and} \\
& I, k \vDash P_{THT}(\phi, I_{[i+1,\ldots,k]}) \text{ and } \ldots I, k \vDash P_{THT}(\mathbf{G}\phi, I_k) \\
& \underset{\text{By Def. 1}}{\Longleftrightarrow} I, k \vDash \bigwedge_{j=i,\ldots,k-1} P_{THT}(\phi, I_{[j,\ldots,k]}) \wedge P_{THT}(\mathbf{G}\phi, I_k) \\
& \underset{\text{By Def. 5}}{\Longleftrightarrow} I, k \vDash \bigwedge_{j=i,\ldots,k} P_{THT}(\phi, I_{[j,\ldots,k]}) \wedge \mathbf{G}\phi \\
& \underset{\text{By Lemma 2}}{\Longleftrightarrow} I, k \vDash P_{THT}(\mathbf{G}\,\phi, I_{[i,\ldots,k]})
\end{aligned}$$

The proof of the eventually case ($\mathbf{F}\ \phi$) follows the same schema of the always one ($\mathbf{G}\ \phi$). If the formula consists of $\mathbf{X}\ \phi$ and $i < k$, then

$I, i \models \mathbf{X}\ \phi \overset{\text{By Def. 1}}{\iff} I, i+1 \models \phi \overset{\text{By Ind. Hypothesis}}{\iff}$
$I, k \models P_{THT}(\phi, I_{[i+1,\dots,k]}) \overset{\text{By Lemma 2}}{\iff} I, k \models P_{THT}(\mathbf{X}\ \phi, I_{[i,\dots,k]})$
Otherwise if $i = k$,
$I, i \models \mathbf{X}\ \phi \overset{\text{By Def. 1}}{\iff} I, k \models P_{THT}(\phi, I_{[i+1,\dots,k]}) \overset{\text{By Lemma 2}}{\iff}$
$I, k \models P_{THT}(\mathbf{X}\ \phi, I_{[i,\dots,k]})$

The case with the previous operator follows a similar proof. □

We can now derive Theorem 1.

Proof of Theorem 1. Let I^f be a prefix of a THT trace of length k, O be a trace of observations, and $\phi \in \mathcal{L}$ (where in subformulas $\mathbf{Y}\ \psi$ of ϕ future operators do not occur). We have to show that

$$P_{THT}(\phi, I^f) = v \implies I^f \models^O_{THT_3} \phi = v, \text{ for } v \in \{\top, \bot\}.$$

According to Lemma 3, given a prefix $I^f = I_{[0,\dots,k]}$ of any THT-trace I and a formula ϕ in $\mathcal{L}$, we have that

$$I, 0 \models \phi \text{ iff } I, k \models P_{THT}(\phi, I_{[0,\dots,k]}).$$

Therefore, if $P_{THT}(\phi, I_{[0,\dots,k]}) = \top$, we have $I, 0 \models \phi$; as this holds for every extension I of I^f we obtain $I^f \models_{THT_3} \phi = \top$. Similarly, if $P_{THT}(\phi, I_{[0,\dots,k]}) = \bot$, then since $I, k \not\models \bot$, it follows that $I, 0 \not\models \phi$, and as this holds for every extension I of I^f, we obtain $I^f \models_{THT_3} \phi = \bot$. The result then follows by the fact that $I^f \models_{THT_3} \phi = v$ implies $I^f \models^O_{THT_3} \phi = v$, for $v \in \{\top, \bot\}$, since $ext(I^f, O)$ restricts the extensions I of I^f to consider. □

To prove Theorem 2, we first prove some lemmas.

Lemma 4. *For any finite trace T^f and formula ϕ, we have $P_{TEL}(\phi, T^f) = \top \implies T^f \models_{TEL}\ \phi = \top$*

Proof.

$$P_{TEL}(\phi, T^f) = \top \overset{\text{By Definition 8}}{\Longrightarrow}$$

$$\begin{cases} P_{THT}(\phi, \langle T^f, T^f\rangle) = \top, \text{ and} \\ P_{THT}(\psi, \langle H^f, T^f\rangle) = \bot \text{ for all } H^f \subset T^f \end{cases} \overset{\text{By Theorem 1}}{\Longrightarrow}$$

$$\begin{cases} \langle T^f, T^f\rangle \models_{THT_3} \phi = \top, \text{ and} \\ \langle H^f, T^f\rangle \models_{THT_3} \phi = \bot \text{ for all } H^f \subset T^f \end{cases} \overset{\text{By Definition 4}}{\Longrightarrow}$$

$$\begin{cases} \langle T, T\rangle \models \phi\ \forall \langle T, T\rangle \in ext(\langle T^f, T^f\rangle), \text{ and} \\ \langle H, T\rangle \not\models \phi\ \forall \langle H, T\rangle \in ext(\langle H^f, T^f\rangle) \text{ for all } H^f \subset T^f \end{cases} \Longrightarrow$$

$$\begin{cases} \langle T^f \emptyset^\omega, T^f \emptyset^\omega\rangle \models \phi, \text{ and} \\ \langle H^f \emptyset^\omega, T^f \emptyset^\omega\rangle \not\models \phi \text{ for all } H^f \subset T^f \end{cases} \overset{\text{By Definition 2}}{\Longrightarrow}$$

$$T^f \emptyset^\omega \models_{TEL} \phi \overset{\text{Since it is the only minimal extension, by Def. 7}}{\Longrightarrow}$$

$$T^f \models_{TEL_3} \phi = \top$$

We note that the result generalizes from $\models_{TEL_3}$ to $\models^O_{TEL_3}$ for any trace O of observations. To this end, the unique minimal traces H_0 and T_O in $ext(H^f, O)$ resp. $ext(T^f, O)$ are considered (if O is not compatible with T^f, the statement is vacuously true). □

Lemma 5. *Given a finite trace T^f and a formula ϕ, we want to prove: $P_{TEL}(\phi, T^f) = \bot \implies T^f \models_{TEL}\ \phi = \bot$*

Proof.

$$P_{TEL}(\phi, T^f) = \bot \overset{\text{By Definition 8}}{\Longrightarrow}$$

$$\begin{cases} P_{THT}(\phi, \langle T^f, T^f\rangle) = \bot, \text{ or} \\ P_{THT}(\psi, \langle H^f, T^f\rangle) = \top \text{ for some } H^f \subset T^f \end{cases} \overset{\text{By Theorem 1}}{\Longrightarrow}$$

$$\begin{cases} \langle T^f, T^f\rangle \models_{THT_3} \phi = \bot, \text{ or} \\ \langle H^f, T^f\rangle \models_{THT_3} \phi = \top \text{ for some } H^f \subset T^f \end{cases} \overset{\text{By Definition 4}}{\Longrightarrow}$$

$$\begin{cases} \langle T, T\rangle \not\models \phi\ \forall \langle T, T\rangle \in ext(\langle T^f, T^f\rangle), \text{ or} \\ \langle H, T\rangle \models \phi\ \forall \langle H, T\rangle \in ext(\langle H^f, T^f\rangle) \text{ for some } H^f \subset T^f \end{cases} \Longrightarrow$$

Let us choose one such H^f if some exists. For all possible extension T of T^f, and H of H^f.

$$\begin{cases} \langle T, T\rangle \not\models \phi\ , \text{ or} \\ \langle H, T\rangle \models \phi \end{cases} \overset{\text{by Definition 2}}{\Longrightarrow}$$

$$\text{For all } T \text{ extension of } T^f,\ T \not\models_{TEL} \phi \overset{\text{By Def. 7}}{\Longrightarrow}$$

$$T^f \models_{TEL_3} \phi = \bot$$

As in the proof of Lemma 4 above, we note that the result generalizes from $\models_{TEL_3}$ to $\models^O_{TEL_3}$ for any trace O of observations. □

We are now ready to show Theorem 2.

Proof of Theorem 2. In Lemmas 4, 5, we have already shown that

- $P_{TEL}(\phi, T^f) = \top \implies T^f \models_{TEL} \phi = \top$, and
- $P_{TEL}(\phi, T^f) = \bot \implies T^f \models_{TEL} \phi = \bot$.

Similar as above, we note that $T^f \models_{TEL} \phi = v$ implies $T^f \models^O_{TEL} \phi = v$, for every $v \in \{\top, \bot\}$ and trace O of observations, which proves the result. □

Let us introduce first the definition of satisfaction for Here and There (HT) logic and of Equilibrium Logic (EL).

Definition 15 (HT-Satisfaction). *Given an interpretation $\langle H, T\rangle$, the satisfaction of a HT formula by $\langle H, T\rangle$ is inductively defined as follows:*

- $\langle H, T\rangle \models p$ *iff* $p \in H$, *for any atom* $p \in \mathcal{P}$
- $\langle H, T\rangle \models \phi \vee \psi$ *iff* $\langle H, T\rangle \models \phi$ *or* $\langle H, T\rangle \models \psi$
- $\langle H, T\rangle \models \phi \wedge \psi$ *iff* $\langle H, T\rangle \models \phi$ *and* $\langle H, T\rangle \models \psi$
- $\langle H, T\rangle \models \phi \rightarrow \psi$ *iff* $\begin{cases} \langle T, T\rangle \not\models \phi \text{ or } \langle T, T\rangle \models \psi, \text{ and} \\ \langle H, T\rangle \not\models \phi \text{ or } \langle H, T\rangle \models \psi \end{cases}$
- $\langle H, T\rangle \not\models \bot$

An interpretation $\langle H, T\rangle$ is a model for a formula ϕ if $\langle H, T\rangle \models \phi$.

An interpretation $\langle H, T\rangle$ is *total* if $H = T$. Furthermore, given two interpretations $\langle H, T\rangle$ and $\langle H', T\rangle$, we say that $\langle H', T\rangle \leq \langle H, T\rangle$ if $H \subseteq T$. We are now ready to introduce the semantics of EL.

Definition 16 (EL-Satisfaction)**.** *A set T is an equilibrium model of a formula ϕ if the following two conditions hold. $i)$ $\langle T, T\rangle \models \phi$, i.e., if T is a total HT model of ϕ, and $ii)$ $H \subset T$ s.t. $\langle H, T\rangle \models \phi$ does not exist, i.e., $\langle T, T\rangle$ has to be minimal.*

Lemma 6. *If a set S of atoms does not occur in the heads of a logic program π, then each answer set S' of π is such that $S' \cap S = \emptyset$.*

We continue with establishing the results of Section 5. To this end, we shall derive a number of auxiliary results, some of which are of interest in their own right.

We start by noting that

Lemma 7. *Suppose $\langle H, T\rangle \models \phi$ and let $a \in H$ be an atom. Then, the substitution of a with $\top$ in ϕ leads to a new logic program ϕ' which has $\langle H \setminus \{a\}, T \setminus \{a\}\rangle$ as a HT model.*

Proof. Let us consider ϕ a HT formula, and by ϕ', $\phi' \equiv \phi[\top/a]$. We can show that $\langle H, T\rangle$ is a model of ϕ with $a \in H$ iff $\langle H', T'\rangle$ is a model of ϕ', where $H' = H \setminus \{a\}$ and $T' = T \setminus \{a\}$.

We can proceed by induction on the complexity of the formula. If $\phi \equiv p$ with $p \in \mathcal{P} \setminus \{a\}$, then $\langle H, T\rangle \vDash p$ iff $p \in H$ by Definition 1. Another base case is the following one: $\langle H, T\rangle \vDash a$ iff $\langle H', T'\rangle \vDash \top$. Finally, we can easily see that $\langle H, T\rangle \vDash \bot$ iff $\langle H', T'\rangle \vDash \bot$ by Definition 1. The induction part simply follows from the Definition 1. □

Lemma 8. *Let $\langle H, T\rangle$ be an H-minimal model of a HT-formula ϕ. Then, the substitution of $a \in H$ with $\top$ in ϕ leads to a new logic program ϕ' which has $\langle H \setminus \{a\}, T \setminus \{a\}\rangle$ as a H-minimal HT model.*

Proof. By Lemma 7, $\langle H', T'\rangle \vDash \phi'$ where $H' = H \setminus \{a\}$ and $T' = T \setminus \{a\}$. Towards a contradiction, assume that $\langle H', T'\rangle$ is not H-minimal for ϕ'. Then $\langle H'', T'\rangle \models \phi'$ for some $H'' \subset H'$. Since p does not occur in ϕ', it follows that $\langle H'' \cup \{p\}, T' \cup \{p\}\rangle \models p \wedge \phi'$, which in turn implies that $\langle H'' \cup \{p\}, T' \cup \{p\}\rangle \models \phi$, as we can resubstitute $\top$ in ϕ by p. Since $H'' \cup \{p\} \subset H$ and $T = T' \cup \{p\}$, it follows that $\langle H, T\rangle$ is not an H-minimal model of ϕ, which is a contraduction. □

Using Lemma 7 and 8, applying Definition 16, we can straightforwardly conclude that

Lemma 9. *Let π be logic program with S as an answer set. Then the substitution of $a \in S$ in π with $\top$ yields a new logic program π' that has $S \setminus \{a\}$ as an answer set.*

Note that splitting sets are closed under union; hence there exists always one maximal splitting set.

Lemma 10. *Let $U \subseteq \mathcal{P}$ be closed under negative dependencies w.r.t. a logic program π, and let U^s be the (unique) maximal splitting set of π contained in U. Then for every $Z \in AS(\pi)$ some $X \in AS(filter(\pi, U))$ exists such that (i) $X|_{U^s} = X^s$ and (ii) $X|_{U \setminus U^s} \subseteq Y^s$, where*

- *X^s is an answer set of $b_{U^s}(\pi)$, and*
- *Y^s is an answer set of $e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s)$.*

Proof. As U^s is a splitting set of π, by the Splitting Theorem [26] every $Z \in AS(\pi)$ is of the form $Z = X^s \cup Y^s$ as in the statement.

As $filter(\pi, U) \subseteq \pi$, (1) the set U^s is also as splitting set of $filter(\pi, U)$. Furthermore, (2) $b_{U^s}(filter(\pi, U)) = b_{U^s}(\pi)$ must hold. Indeed, every rule $r \in b_{U^s}(filter(\pi, U))$ is over U^s and thus $r \in b_{U^s}(\pi)$ as well. Suppose that some $r \in b_{U^s}(\pi) \setminus b_{U^s}(filter(\pi, U))$ exists. By definition of $filter(\pi, U)$, this means that some negative literal $\neg a$ occurs in the body of r where a depends on some atom $b \in \mathcal{P} \setminus U$. However, U is closed under negative dependencies, and thus $b \in U$ must hold, which is a contradiction.

In view of (1) and (2), by the Splitting Theorem each set $X = X^s \cup Y'$ such that Y' is an answer set of $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(filter(\pi, U)), X^s) = e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$, is an answer set of $filter(\pi, U)$.

We claim that (3) $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$ is a positive program. Indeed, denote for any atom a by U_a the smallest splitting set of π that contains a (which exists since splitting sets are closed under intersection). If $\neg a$ occurs in the body of a rule r in $filter(\pi, U)$, we obtain $U_a \subseteq U$ as U is closed under negative dependencies. Furthermore, $U_a \subseteq U^s$ by maximality of U^s. It follows $a \in U^s$, and thus $\neg a$ does not occur in rule bodies of $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$.

Now since $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s) \subseteq e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s)$, Y^s is a model of $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$; by (3), every minimal model Y' of $e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$ is an answer set of that program, and some $Y' \subseteq Y^s$ always exists. This proves the result. □

We next consider progression at the level of single models, which we shall then use to analyze progresson of the skeptical consequences. To this end, we shall consider the general progression operator for inputs $\mathit{skep\_prog}(\pi, X, X, U)$, which we simply refer to as $\mathit{prog}(\pi, X, U)$.

Lemma 11. *Let U be a set of atoms closed under negative dependencies w.r.t. a logic program π. Then every answer set Z of π is of the form $Z = X \cup Y$ for some $X \in AS(filter(\pi, U))$ and some $Y \in AS(\mathit{prog}(\pi, X, U))$.*

Proof. By Lemma 10, we have that some $X \in AS(filter(\pi, U))$ exists such that $X|_{U^s} = X^s$ and $X|_{U \setminus U^s} \subseteq Y^s$ (where U^s is the maximal splitting set contained in U), $X^s \in AS(b_{U^s}(\pi))$, and $Y^s \in AS(e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s))$.

Consider $\mathit{prog}(\pi, X^s, U)$. This program coincides with $e_{U^s}(\pi \setminus b_{U^a}(\pi), X^s)$; this follows directly from Definition 12 and $X^s \subseteq U^s$. Therefore, $Y^s \in AS(\mathit{prog}(\pi, X^s, U))$. Since $X|_{U \setminus U^s} \subseteq Y^s$, we can by Lemma 9 substitute atoms in $X|_{U \setminus U^s}$ by $\top$, obtaining as a result $Y \in AS(\mathit{prog}(\pi, X, U))$. □

Lemma 12. *Let $U \subseteq \mathcal{P}$ be closed under negative dependencies w.r.t. a logic program π, and let U^s be the (unique) maximal splitting set of π contained in U. If every disjunctive rule $r \in \pi$ satisfies $H(r) \cap (U \setminus U^s) = \emptyset$, then for every $X \in AS(filter(\pi, U))$ and $Y \in AS(\mathit{prog}(\pi, U, X))$ the set $Z = X \cup Y$ is an answer set of π.*

Proof. Let $X \in AS(filter(\pi, U))$ and $Y \in AS(\mathit{prog}(\pi, X, U))$. We show that $X^s = X \cap U^s$ and $Y^s = Y \cup (X \setminus U^s)$ are answer sets of $b_{U^s}(\pi)$, and $e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s)$, respectively; then by the Splitting Theorem [26], $Z = X^s \cup Y^s = (X \cap U^s) \cup Y \cup (X \setminus U^s) = X \cup Y$ is an answer set of π.

As in the proof of Lemma 10, we conclude that (1) $X^s = X|_{U^s}$ is an answer set of $b_{U^s}(filter(\pi, U))$, which coincides with $b_{U^s}(\pi)$, (2) $X \setminus X^s$ is an answer set of $\pi' = e_{U^s}(filter(\pi, U) \setminus b_{U^s}(\pi), X^s)$, and (3) π' is a positive program such that $\pi' \subseteq e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s)$. Moreover, by the assertion of the lemma, (4) each rule $r \in \pi'$ is non-disjunctive.

Suppose towards a contradiction that Z is not an answer set of π. Let $Z' \subset Z$ be a witness of this. As X^s is an answer set of $b_{U^s}(\pi)$, we must have $Z' \cap U^s = X^s$. Thus $Z' \cap (\mathcal{P} \setminus U^s) \subset Y^s$ must hold.

By (3) and (4), we conclude that $X \setminus U^s \subseteq Z'$ must hold, as π' has a unique minimal model (which is $X \setminus U^s$). Hence, $Z' \cap U = X$ must hold. Consider now $\mathit{prog}(\pi, X, U)$. This program is equivalent to the program π'' that results from $e_{U^s}(\pi \setminus b_{U^s}(\pi), X^s)$ by substituting each atom $a \in X \setminus U^s$ with $\top$. Since each such atom is in Z', we obtain that Z' is a model of π'', and hence of $\mathit{prog}(\pi, X, U)$. As $Z' \subset Z$, then also $Y' = Z' \cap (\mathcal{P} \setminus U)$ is model of $\mathit{prog}(\pi, X, U)$, As $Y' \subset Y$ must hold, this contradicts that Y is an answer set of $prog(\pi, X, U)$. □

Now we have all auxiliary results to show Theorem 3.

Proof of Theorem 3. Let U be a set of atoms closed under negative dependencies on atoms in $\mathcal{P} \setminus U$ w.r.t. π. If Z is an answer set of π, then by Lemma 11 there exist an answer set X of $filter(\pi, X)$ and an answer set Y of $prog(\pi, U, X)$ such that $Z = X \cup Y$.

As for the converse direction, under the syntactic constriction on disjunction in heads of rules r that $H(r) \cap U \setminus U^s = \emptyset$, we can apply Lemma 12, obtaining that if X is an answer set of $filter(\pi, X)$ and Y answer set of $prog(\pi, U, X)$, then Z is an answer set of π. □

Observation 1. *We just observe that if S is an answer set of a logic program π, then by GL definition, S is the minimal model of π^S. Hence, for any atom p $p \notin S$, we can remove rules where p appears positive in the body or negative in the head, and delete literals p in the head and negated literal $\neg p$ in the body of the remaining rules. If p is not a credulous consequence of the program π, modifying the program π in this way does not change the set of the answer sets.*

We are now in a position to show also Theorem 4.

Proof of Theorem 4. Consider a logic program π, and let $U \subseteq \mathcal{P}$ be a set of atoms closed under negative dependencies. Let again U^s denote the maximal splitting set of π such that $U^s \subseteq U$. Let us denote (i) by C and B the sets of the skeptical and the brave consequences of π, respectively; (ii) by C_X and B_X the sets of the skeptical and the brave consequences of $filter(\pi, U)$, respectively; and (iii) by C_Y and B_Y the sets of the skeptical and the brave consequences of $skep\_prog(\pi, C_X, B_X, U)$, respectively.

Then, we need to prove that $C_X \cup C_Y \subseteq C$, and $B \subseteq B_X \cup B_Y$.

Let us consider the set $\Gamma(\pi, U) = \{(X, Y) \mid X \in AS(\mathit{filter}(\pi, U)), Y \in AS(prog(\pi, X, U))\}$. By Lemma 11, we know that (1) every the answer set Z of π is expressed as $Z = X \cup Y$ for some $(X, Y) \in \Gamma(\pi, U)$. Hence C_X is contained in the set of all the skeptical consequences of π, i.e., $C_X \subseteq C$ holds. By application of Lemma 9, we can (i) replace in the program π the atoms in C_X with $\top$, which will preserve each answer set S of the original program π modulo C_X, but possibly will create new ones. Regarding the brave consequences, we note that $(B \cap U) \subseteq B_X$ must hold.

Furthermore, by Lemma 11 we see that if an p atom from U^s does not appear in any X from $\Gamma(\pi, U)$, then p will not appear in any answer set of the original program π. Thanks to Observation 1, we can (2) remove rules where $p \in U^s \setminus B_X$ appears positive in the body or negated in the head, and delete negated literals $\neg p$ in the body of the remaining rules. These modifications do not alter the answer sets of the program.

Steps (i) and (ii) corresponds exactly to the application of function $skept\_progress(\pi, C_X, B_X, U)$. As already pointed out, the resulting program may have new answer sets, but each answer set of the original program π (modulo atoms in C_X) is preserved. Therefore, we can conclude that $C_Y \subseteq C$ and that $B \cap (\mathcal{P} \setminus U) \subseteq B_Y$ must hold. Putting things together, we then obtain that $C_X \cup C_Y \subseteq C$ and $B \subseteq B_X \cup B_Y$, which proves the result. □

Lemma 13 (Lemma 2 from [4])**.** *Let π be a temporal program without fulfillment rules. Then $I \models \pi$ in THT iff $I^\omega \models \pi^\omega$ in HT.*

Lemma 14 (Theorem 1 from [4])**.** *Let π be a temporal program without fulfillment rules. Then $T = \langle T, T \rangle$ is a temporal equilibrium model of π iff T^ω is a stable model of π^ω.*

We are now prepared to derive Theorem 5. In the following proof, we will use the notation $T \equiv trace(T^\omega)$.

Proof of Theorem 5. We want to prove the following statement. Let π be a temporal program without fulfillment rules. Then (i) T^ω is a stable model for π^ω if and only if T is an equilibrium trace for π. Furthermore, (ii) if T is an equilibrium trace for π and $T \models \pi_{ful}$ for a set π_{ful} of fulfillment rules, then T is an equilibrium trace for $\pi \cup \pi_{ful}$.

Item (i) is a direct consequence of Lemma 14. For item (ii), let us assume that $T \models \pi_{ful}$. We will prove satisfiability and minimality:

- Satisfiability. If T satisfies both π_{ful} and π, then by Definition 1 T also satisfies $\pi \cup \pi_{ful}$.
- Minimality. By Lemma 13, we can conclude that if $T = \langle T, T \rangle$ is a stable model of π, then T^ω is a stable model of π^ω. That is, no $\langle H', T \rangle \models \pi$ exists such that $H' < T$. Let us assume there is $\langle H', T \rangle \models \pi \cup \pi_{ful}$, such that $H' < T$. From Definition 1, we get that $\langle H', T \rangle \models \pi_{ful}$ and $\langle H', T \rangle \models \pi$. The latter means that T is not an equilibrium trace for π, which is a contradiction.

□

Theorem 6 is then not difficult to show.

Proof of Theorem 6. (Sketch) The result is obtained by repeated application of Theorem 3, where the definition of the sets U^i takes care that they are closed under negatives dependencies. □

We next consider Theorem 7.

Proof of Theorem 7. Let π be a temporal program without fulfillment rules. We want to prove that if π admits an equilibrium trace, then $trace(\bigcap AS(\pi^\omega)) = \bigcap TEL(\pi)$.

By Theorem 5, T^ω is a stable model for π^ω iff $trace(T^\omega)$ is an equilibrium trace for π. Therefore, we also have that (i) $trace(\bigcap AS(\pi^\omega)) = \bigcap TEL(\pi)$.

□

Finally, we consider Theorem 8.

Proof of Theorem 8. Let π be a temporal program without fulfillment rules, If π admits an equilibrium trace, then we want to prove

$$trace(\textstyle\bigcup_{i \geq 0} \bigcap AS(\pi^i_\cap)) \subseteq \bigcap TEL(\pi), \text{ and}$$
$$trace(\textstyle\bigcup_{i \geq 0} \bigcup AS(\pi^i_\cap)) \supseteq \bigcup TEL(\pi)$$

The result is obtained by repeated application of Theorem 4, where the definition of the sets U^i takes care that they are closed under negatives dependencies, and C^i and B^i corresponds, respectively, to the skeptical and the brave consequences of $filter(\pi^i_\cap)$.

□