Past-present temporal programs over finite traces

Pedro Cabalar[1][0000−0001−7440−0953], Martín Diéguez[2][0000−0003−3440−4348], François Laferrière[3][0009−0006−8147−572X], and Torsten Schaub[3][0000−0002−7456−041X]

[1] University of Corunna, Spain
[2] University of Angers, France
[3] University of Potsdam, Germany

Abstract. Extensions of Answer Set Programming with language constructs from temporal logics, such as temporal equilibrium logic over finite traces (TEL_f), provide an expressive computational framework for modeling dynamic applications. In this paper, we study the so-called past-present syntactic subclass, which consists of a set of logic programming rules whose body references to the past and head to the present. Such restriction ensures that the past remains independent of the future, which is the case in most dynamic domains. We extend the definitions of completion and loop formulas to the case of past-present formulas, which allows for capturing the temporal stable models of past-present temporal programs by means of an LTL_f expression.

1 Introduction

Reasoning about dynamic scenarios is a central problem in the areas of Knowledge Representation [6] (KR) and Artificial Intelligence (AI). Several formal approaches and systems have emerged to introduce non-monotonic reasoning features in scenarios where the formalisation of time is fundamental [3, 4, 13, 20, 25]. In *Answer Set Programming* [7] (ASP), former approaches to temporal reasoning use first-order encodings [21] where the time is represented by means of a variable whose value comes from a finite domain. The main advantage of those approaches is that the computation of answer sets can be achieved via incremental solving [18]. Their downside is that they require an explicit representation of time points.

Temporal Equilibrium Logic [2] (TEL) was proposed as a temporal extension of *Equilibrium Logic* [23] with connectives from *Linear Time Temporal Logic* [24] (LTL). Due to the computational complexity of its satisfiability problem (ExpSpace), finding tractable fragments of TEL with good computational properties have also been a topic in the literature. Within this context, *splittable temporal logic programs* [1] have been proved to be a syntactic fragment of TEL that allows for a reduction to LTL via the use of Loop Formulas [16].

When considering incremental solving, logics on finite traces such as LTL_f [12] have been shown to be more suitable. Accordingly, *Temporal Equilibrium Logic* *on Finite traces* (TEL$_f$) [9] was created and became the foundations of the temporal ASP solver *telingo* [8].

We present a new syntactic fragment of TEL$_f$, named *past-present* temporal logic programs. Inspired by Gabbay's seminal paper [17], where the declarative character of past temporal operators is emphasized, this language consists of a set of logic programming rules whose formulas in the head are disjunctions of atoms that reference the present, while in its body we allow for any arbitrary temporal formula without the use of future operators. Such restriction ensures that the past remains independent of the future, which is the case in most dynamic domains, and makes this fragment advantageous for incremental solving.

As a contribution, we study the Lin-Zhao theorem [22] within the context of past-present temporal logic programs. More precisely, we show that when the program is *tight* [14], extending Clark's completion [11, 15] to the temporal case suffices to capture the answer sets of a finite past-present program as the LTL$_f$-models of a corresponding temporal formula. We also show that, when the program is not tight, the use of loop formulas is necessary. To this purpose, we extend the definition of loop formulas to the case of past-present programs and we prove the Lin-Zhao theorem in our setting.

The paper is organised as follows: in Section 2, we review the formal background and we introduce the concept of past-present temporal programs. In Section 3, we extend the completion property to the temporal case. Section 4 is devoted to the introduction of our temporal extension of loop formulas. Finally, in Section 5, we present the conclusions as well as some future research lines. The full version of this paper can be found in [10].

2 Past-present temporal programs over finite traces

In this section, we introduce the so-called *past-present* temporal logic programs and its semantics based on *Temporal Equilibrium Logic over Finite traces* (TEL$_f$ for short) as in [2]. The syntax of our language is inspired from the pure-past fragment of Linear Time Temporal Logic (LTL) [19], since the only future operators used are *always* and *weak next.*

We start from a given set $\mathcal{A}$ of atoms which we call the *alphabet.* Then, *past temporal formulas* φ are defined by the grammar:

$$\varphi ::= a \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bullet\varphi \mid \varphi_1 \,\mathsf{S}\, \varphi_2 \mid \varphi_1 \,\mathsf{T}\, \varphi_2$$

where $a \in \mathcal{A}$ is an atom. The intended meaning of the (modal) temporal operators is as in LTL. $\bullet\varphi$ means that φ is true at the previous time point; $\varphi \,\mathsf{S}\, \psi$ can be read as φ is true since ψ was true and $\varphi \,\mathsf{T}\, \psi$ means that ψ is true since both φ and ψ became true simultaneously or ψ has been true from the beginning.

Given $a \in \mathbb{N}$ and $b \in \mathbb{N}$, we let $[a..b] \stackrel{def}{=} \{i \in \mathbb{N} \mid a \leq i \leq b\}$ and $[a..b) \stackrel{def}{=} \{i \in \mathbb{N} \mid a \leq i < b\}$. A *finite trace* $\mathbf{T}$ of length λ over alphabet $\mathcal{A}$ is a sequence $\mathbf{T} = (T_i)_{i\in[0..\lambda)}$ of sets $T_i \subseteq \mathcal{A}$. To represent a given trace, we write a sequence of sets of atoms concatenated with '$\cdot$'. For instance, the finite trace $\{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset$ has length 4 and makes a true at even time points and false at odd ones.

A *Here-and-There trace* (for short *HT-trace*) of length λ over alphabet $\mathcal{A}$ is a sequence of pairs $(\langle H_i, T_i\rangle)_{i\in[0..\lambda)}$ with $H_i \subseteq T_i$ for any $i \in [0..\lambda)$. For convenience, we usually represent the HT-trace as the pair $\langle \mathbf{H}, \mathbf{T}\rangle$ of traces $\mathbf{H} = (H_i)_{i\in[0..\lambda)}$ and $\mathbf{T} = (T_i)_{i\in[0..\lambda)}$. Given $\mathbf{M} = \langle \mathbf{H}, \mathbf{T}\rangle$, we also denote its length as $|\mathbf{M}| \overset{def}{=} |\mathbf{H}| = |\mathbf{T}| = \lambda$. Note that the two traces $\mathbf{H}$, $\mathbf{T}$ must satisfy a kind of order relation, since $H_i \subseteq T_i$ for each time point i. Formally, we define the ordering $\mathbf{H} \leq \mathbf{T}$ between two traces of the same length λ as $H_i \subseteq T_i$ for each $i \in [0..\lambda)$. Furthermore, we define $\mathbf{H} < \mathbf{T}$ as both $\mathbf{H} \leq \mathbf{T}$ and $\mathbf{H} \neq \mathbf{T}$. Thus, an HT-trace can also be defined as any pair $\langle \mathbf{H}, \mathbf{T}\rangle$ of traces such that $\mathbf{H} \leq \mathbf{T}$. The particular type of HT-traces satisfying $\mathbf{H} = \mathbf{T}$ are called *total*.

An HT-trace $\mathbf{M} = \langle \mathbf{H}, \mathbf{T}\rangle$ of length λ over alphabet $\mathcal{A}$ *satisfies* a past temporal formula φ at time point $k \in [0..\lambda)$, written $\mathbf{M}, k \models \varphi$, if the following conditions hold:

1. $\mathbf{M}, k \models \top$ and $\mathbf{M}, k \not\models \bot$
2. $\mathbf{M}, k \models p$ if $p \in H_k$ for any atom $p \in \mathcal{A}$
3. $\mathbf{M}, k \models \varphi \wedge \psi$ iff $\mathbf{M}, k \models \varphi$ and $\mathbf{M}, k \models \psi$
4. $\mathbf{M}, k \models \varphi \vee \psi$ iff $\mathbf{M}, k \models \varphi$ or $\mathbf{M}, k \models \psi$
5. $\mathbf{M}, k \models \neg\varphi$ iff $\langle \mathbf{T}, \mathbf{T}\rangle, k \not\models \varphi$
6. $\mathbf{M}, k \models \bullet\varphi$ iff $k > 0$ and $\mathbf{M}, k{-}1 \models \varphi$
7. $\mathbf{M}, k \models \varphi \ \mathbf{S}\ \psi$ iff for some $j \in [0..k]$, we have $\mathbf{M}, j \models \psi$ and $\mathbf{M}, i \models \varphi$ for all $i \in (j..k]$
8. $\mathbf{M}, k \models \varphi \ \mathbf{T}\ \psi$ iff for all $j \in [0..k]$, we have $\mathbf{M}, j \models \psi$ or $\mathbf{M}, i \models \varphi$ for some $i \in (j..k]$

A formula φ is a *tautology* (or is *valid*), written $\models \varphi$, iff $\mathbf{M}, k \models \varphi$ for any HT-trace $\mathbf{M}$ and any $k \in [0..\lambda)$. We call the logic induced by the set of all tautologies *Temporal logic of Here-and-There over finite traces* (THT$_f$ for short).

The following equivalences hold in THT$_f$: 1. $\top \equiv \neg\bot$, 2. $\mathbf{I} \equiv \neg\bullet\top$, 3. $\blacksquare\varphi \equiv \bot \mathbf{T} \varphi$, 4. $\blacklozenge\varphi \equiv \top\ \mathbf{S}\ \varphi$, 5. $\widehat{\bullet}\varphi \equiv \bullet\varphi \vee \mathbf{I}$.

Definition 1 (Past-present Program). *Given alphabet $\mathcal{A}$, the set of* regular literals *is defined as $\{a, \neg a, \mid a \in \mathcal{A}\}$.*

A past-present rule *is either:*

- *an* initial rule *of form* $H \leftarrow B$
- *a* dynamic rule *of form* $\widehat{\circ}\,\square(H \leftarrow B)$
- *a* final rule *of form* $\square(\,\mathbf{F} \rightarrow (\bot \leftarrow B)\,)$

where B is an pure past formula for dynamic rules and $B = b_1 \wedge \cdots \wedge b_n$ with $n \geq 0$ for initial and final rules, the b_i are regular literals, $H = a_1 \vee \cdots \vee a_m$ with $m \geq 0$ and $a_j \in \mathcal{A}$. A past-present program *is a set of past-present rules.* □

We let $I(P)$, $D(P)$, and $F(P)$ stand for the set of all initial, dynamic, and final rules in a past-present program P, respectively. Additionally we refer to H as the *head* of a rule r and to B as the *body* of r. We let $B(r) = B$ and $H(r) = H$ for all types of rules above. For example, let consider the following past-present program P_1:

$$load \leftarrow \tag{1}$$
$$\widehat{\circ}\Box(shoot \lor load \lor unload \leftarrow\ ) \tag{2}$$
$$\widehat{\circ}\Box(dead \leftarrow shoot \land \neg unload\,\mathbf{S}\,load) \tag{3}$$
$$\widehat{\circ}\Box(shoot \leftarrow dead) \tag{4}$$
$$\Box(\mathbf{F} \rightarrow (\bot \leftarrow \neg dead)) \tag{5}$$

We get $I(P_1) = \{(1)\}$, $D(P_1) = \{(2),(3),(4)\}$, and $F(P_1) = \{(3)\}$. Rule (1) states that the gun is initially loaded. Rule (2) gives the choice to shoot, load, or unload the gun. Rule (3) states that if the gun is shot while it has been loaded, and not unloaded since, the target is dead. Rule (4) states that if the target is dead, we shoot it again. Rule (5) ensures that the target is dead at the end of the trace.

The satisfaction relation of a past-present rule on an HT-trace $\mathbf{M}$ of length λ and at time point $k \in [0..\lambda)$ is defined below:

- $\mathbf{M}, k \models H \leftarrow B$ iff $\mathbf{M}', k \not\models B$ or $\mathbf{M}', k \models H$, for all $\mathbf{M}' \in \{\mathbf{M}, \langle \mathbf{T}, \mathbf{T}\rangle\}$
- $\mathbf{M}, k \models \widehat{\circ}\,\Box(H \leftarrow B)$ iff $\mathbf{M}', i \not\models B$ or $\mathbf{M}', i \models H$ for all $\mathbf{M}' \in \{\mathbf{M}, \langle \mathbf{T}, \mathbf{T}\rangle\}$ and all $i \in [k+1..\lambda)$
- $\mathbf{M}, k \models \Box(\,\mathbf{F} \rightarrow (\bot \leftarrow B)\,)$ iff $\langle \mathbf{T}, \mathbf{T}\rangle, \lambda - 1 \not\models B$

An HT-trace $\mathbf{M}$ is a *model* of a past-present program P if $\mathbf{M}, 0 \models r$ for all rule $r \in P$. Let P be past-present program. A total HT-trace $\langle \mathbf{T}, \mathbf{T}\rangle$ is a *temporal equilibrium model* of P iff $\langle \mathbf{T}, \mathbf{T}\rangle$ is a *model* of P, and there is no other $\mathbf{H} < \mathbf{T}$ such that $\langle \mathbf{H}, \mathbf{T}\rangle$ is a *model* of P. The trace $\mathbf{T}$ is called a *temporal stable model* (TS-model) of P.

For length $\lambda = 2$, P_1 has a unique TS-model $\{load\} \cdot \{shoot, dead\}$.

3 Temporal completion

In this section, we extend the completion property to the temporal case of past-present programs.

An occurrence of an atom in a formula is *positive* if it is in the antecedent of an even number of implications, negative otherwise. An occurrence of an atom in a formula is *present* if it is not in the scope of $\bullet$ (previous). Given a past-present program P over $\mathcal{A}$, we define its *(positive) dependency graph* $G(P)$ as $(\mathcal{A}, E)$ such that $(a, b) \in E$ if there is a rule $r \in P$ such that $a \in H(r) \cap \mathcal{A}$ and b has positive and present occurence in $B(r)$ that is not in the scope of negation. A nonempty set $L \subseteq \mathcal{A}$ of atoms is called *loop* of P if, for every pair a, b of atoms in L, there exists a path of length > 0 from a to b in $G(P)$ such that all vertices in the path belong to L. We let $L(P)$ denote the set of loops of P.

Due to the structure of past-present programs, dependencies from the future to the past cannot happen, and therefore there can only be loops within a same time point. To reflect this, the definitions above only consider atoms with present occurences. For example, rule $a \leftarrow b \wedge \bullet c$ generates the edge (a, b) but not (a, c).

For P_1, we get for the initial rules $G(I(P_1)) = (\{\mathit{load}, \mathit{unload}, \mathit{shoot}, \mathit{dead}\}, \emptyset)$ whose loops are $L(I(P_1)) = \emptyset$. For the dynamic rules, we get $G(D(P_1)) = (\{\mathit{load}, \mathit{unload}, \mathit{shoot}, \mathit{dead}\}, \{(\mathit{dead}, \mathit{shoot}), (\mathit{dead}, \mathit{load}), (\mathit{shoot}, \mathit{dead})\})$ and $L(D(P_1)) = \{\{\mathit{shoot}, \mathit{dead}\}\}$.

In the following, $\varphi \rightarrow \psi \overset{def}{=} \psi \leftarrow \varphi$ and $\varphi \leftrightarrow \psi \overset{def}{=} \varphi \rightarrow \psi \wedge \varphi \leftarrow \psi$.

Definition 2 (Temporal completion). *We define the temporal* completion *formula of an atom a in a past-present program P over $\mathcal{A}$, denoted $CF_P(a)$ as:*

$$\square\big(a \leftrightarrow \bigvee_{r \in I(P), a \in H(r)} (\mathbf{I} \wedge S(r, a)) \vee \bigvee_{r \in D(P), a \in H(r)} (\neg \mathbf{I} \wedge S(r, a))\big)$$

where $S(r, a) = B(r) \wedge \bigwedge_{p \in H(r) \setminus \{a\}} \neg p$.

The temporal completion *formula of P, denoted $CF(P)$, is*

$$\{CF_P(a) \mid a \in \mathcal{A}\} \cup \{r \mid r \in I(P) \cup D(P), H(r) = \bot\} \cup F(P).$$

A past-present program P is said to be *tight* if $I(P)$ and $D(P)$ do not contain any loop.

Theorem 1. *Let P be a tight past-present program and $\mathbf{T}$ a trace of length λ. Then, $\mathbf{T}$ is a* TS*-model of P iff $\mathbf{T}$ is a LTL$_f$-model of $CF(P)$.* $\square$

The completion of P_1 is

$$CF(P_1) = \left\{ \begin{array}{c} \square(\mathit{load} \leftrightarrow \mathbf{I} \vee (\neg \mathbf{I} \wedge \neg \mathit{shoot} \wedge \neg \mathit{unload})), \\ \square(\mathit{shoot} \leftrightarrow (\neg \mathbf{I} \wedge \neg \mathit{load} \wedge \neg \mathit{unload})) \vee (\neg \mathbf{I} \wedge \mathit{dead})), \\ \square(\mathit{unload} \leftrightarrow (\neg \mathbf{I} \wedge \neg \mathit{shoot} \wedge \neg \mathit{load})), \\ \square(\mathit{dead} \leftrightarrow (\neg \mathbf{I} \wedge \mathit{shoot} \wedge \neg \mathit{unload} \, \mathbf{S} \, \mathit{load})), \\ \square(\mathbf{F} \rightarrow (\bot \leftarrow \neg \mathit{dead})) \end{array} \right\}.$$

For $\lambda = 2$, $CF(P_1)$ has a unique LTL$_f$-model $\{\mathit{load}\} \cdot \{\mathit{shoot}, \mathit{dead}\}$, which is identical to the TS-model of P_1. Notice that for this example, the TS-models of the program match the LTL$_f$-models of its completion despite the program not being tight. This is generally not the case. Let P_2 be the program made of the rules (1), (3), (4) and (5). The completion of P_2 is

$$CF(P_2) = \left\{ \begin{array}{l} \square(\mathit{load} \leftrightarrow \mathbf{I}),\ \square(\mathit{shoot} \leftrightarrow (\neg \mathbf{I} \wedge \mathit{dead})),\ \square(\mathit{unload} \leftrightarrow \bot), \\ \square(\mathit{dead} \leftrightarrow (\neg \mathbf{I} \wedge \mathit{shoot} \wedge \neg \mathit{unload} \, \mathbf{S} \, \mathit{load})),\ \square(\mathbf{F} \rightarrow (\bot \leftarrow \neg \mathit{dead})) \end{array} \right\}.$$

P_2 does not have any TS-model, but $\{\mathit{load}\} \cdot \{\mathit{shoot}, \mathit{dead}\}$ is a LTL$_f$-model of $CF(P_2)$. Under ASP semantics, it is impossible to derive any element of the loop $\{\mathit{shoot}, \mathit{dead}\}$, as deriving *dead* requires *shoot* to be true, and deriving *shoot* requires *dead* to be true. The completion does not restrict this kind of circular derivation and therefore is insufficient to fully capture ASP semantics.

4 Temporal loop formulas

To restrict circular derivations, Lin and Zhao introduced the concept of loop formulas in [22]. In this section, we extend their work to past-present programs.

Definition 3. *Let φ be a implication-free past-present formula and L a loop. We define the supporting transformation of φ with respect to L as*

$$\begin{aligned}
S_{\bot}(L) &\overset{def}{=} \bot\\
S_{p}(L) &\overset{def}{=} \bot \textit{ if } p \in L \textit{ ; } p \textit{ otherwise, for any } p \in \mathcal{A}\\
S_{\neg\varphi}(L) &\overset{def}{=} \neg\varphi\\
S_{\varphi\wedge\psi}(L) &\overset{def}{=} S_{\varphi}(L) \wedge S_{\psi}(L)\\
S_{\varphi\vee\psi}(L) &\overset{def}{=} S_{\varphi}(L) \vee S_{\psi}(L)\\
S_{\bullet\varphi}(L) &\overset{def}{=} \bullet\varphi\\
S_{\varphi \mathbf{T} \psi}(L) &\overset{def}{=} S_{\psi}(L) \wedge (S_{\varphi}(L) \vee \bullet(\varphi \mathbf{T} \psi))\\
S_{\varphi \mathbf{S} \psi}(L) &\overset{def}{=} S_{\psi}(L) \vee (S_{\varphi}(L) \wedge \bullet(\varphi \mathbf{S} \psi))
\end{aligned}$$

□

Definition 4 (External support). *Given a past-present program P, the external support formula of a set of atoms $L \subseteq \mathcal{A}$ wrt P, is defined as*

$$ES_P(L) = \bigvee_{r\in P, H(r)\cap L\neq\emptyset} \big(S_{B(r)}(L) \wedge \bigwedge_{a\in H(r)\setminus L} \neg a\big)$$

□

For instance, for $L = \{shoot, dead\}$, $ES_{P_2}(L)$ and $ES_{P_1}(L)$ are

$$\begin{aligned}
ES_{P_2}(L) &= S_{dead}(L) \vee S_{shoot\wedge\neg unload\,\mathbf{S}\,load}(L)\\
&= S_{dead}(L) \vee (S_{shoot}(L) \wedge S_{\neg unload\,\mathbf{S}\,load}(L))\\
&= S_{dead}(L) \vee (S_{shoot}(L) \wedge S_{\neg unload}(L) \vee \bullet(\neg unload\,\mathbf{S}\,load))\\
&= \bot \vee (\bot \wedge \neg unload \vee \bullet(\neg unload\,\mathbf{S}\,load)) = \bot.\\
ES_{P_1}(L) &= S_{dead}(L) \vee S_{shoot\wedge\neg unload\,\mathbf{S}\,load}(L) \vee (\neg load \wedge \neg unload)\\
&= \neg load \wedge \neg unload.
\end{aligned}$$

Rule (2) provides an external support for L. The body *dead* of rule (4) is also a support for L, but not external as *dead* belongs to L. The supporting transformation only keeps external supports by removing from the body any positive and present occurence of element of L.

Definition 5 (Loop formulas). *We define the set of loop formulas of a past-present program P over $\mathcal{A}$, denoted $LF(P)$, as:*

$$\bigvee_{a\in L} a \rightarrow ES_{I(P)}(L) \textit{ for any loop } L \textit{ in } I(P)$$

$$\widehat{\circ}\Box\Big(\bigvee_{a\in L} a \rightarrow ES_{D(P)}(L)\Big) \textit{ for any loop } L \textit{ in } D(P)$$

Theorem 2. *Let P be a past-present program and $\mathbf{T}$ a trace of length λ. Then, $\mathbf{T}$ is a* TS*-model of P iff $\mathbf{T}$ is a LTL$_f$-model of $CF(P) \cup LF(P)$.* $\square$

For our examples, we have that $LF(P_1) = \widehat{\circ}\square(shoot \vee dead \rightarrow \neg load \wedge \neg unload)$ and $LF(P_2) = \widehat{\circ}\square(shoot \vee dead \rightarrow \bot)$. It can be also checked that $\{load\}\cdot\{shoot, dead\}$ satisfies $LF(P_1)$, but not $LF(P_2)$. So, we have that $CF(P_1) \cup LF(P_1)$ has a unique LTL$_f$-model $\{load\} \cdot \{shoot, dead\}$, while $CF(P_2) \cup LF(P_2)$ has no LTL$_f$-model, matching the TS-models of the respective programs.

Ferraris et al. [16] proposed an approach where the computation of the completion can be avoided by considering unitary cycles. We extended such results for past-present programs in the extended version [10].

5 Conclusion

We have focused on temporal logic programming within the context of Temporal Equilibrium Logic over finite traces. More precisely, we have studied a fragment close to logic programming rules in the spirit of [17]: a past-present temporal logic program consists of a set of rules whose body refers to the past and present while their head refers to the present. This fragment is very interesting for implementation purposes since it can be solved by means of incremental solving techniques as implemented in *telingo*.

Contrary to the propositional case [16], where answer sets of an arbitrary propositional formula can be captured by means of the classical models of another formula ψ, in the temporal case, this is impossible to do the same mapping among the temporal equilibrium models of a formula φ and the LTL models of another formula ψ [5].

In this paper, we show that past-present temporal logic programs can be effectively reduced to LTL formulas by means of completion and loop formulas. More precisely, we extend the definition of completion and temporal loop formulas in the spirit of Lin and Zhao [22] to the temporal case, and we show that for tight past-present programs, the use of completion is sufficient to achieve a reduction to an LTL$_f$ formula. Moreover, when the program is not tight, we also show that the computation of the temporal completion and a finite number of loop formulas suffices to reduce TEL$_f$ to LTL$_f$.

Acknowledgments This work was supported by MICINN, Spain, grant PID2020-116201GB-I00, Xunta de Galicia, Spain (GPC ED431B 2019/03), Région Pays de la Loire, France, (project etoiles montantes CTASP) and DFG grant SCHA 550/15, Germany.

References

1. Aguado, F., Cabalar, P., Pérez, G., Vidal, C.: Loop formulas for splitable temporal logic programs. In: Delgrande, J., Faber, W. (eds.) Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LP-

NMR'11). Lecture Notes in Artificial Intelligence, vol. 6645, pp. 80–92. Springer-Verlag (2011)

2. Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., Schaub, T., Schuhmann, A., Vidal, C.: Linear-time temporal answer set programming. Theory Pract. Log. Program. **23**(1), 2–56 (2023)
3. Baral, C., Zhao, J.: Non-monotonic temporal logics for goal specification. In: Veloso, M.M. (ed.) IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 236–242 (2007)
4. Baral, C., Zhao, J.: Non-monotonic temporal logics that facilitate elaboration tolerant revision of goals. In: Fox, D., Gomes, C.P. (eds.) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008. pp. 406–411. AAAI Press (2008)
5. Bozzelli, L., Pearce, D.: On the expressiveness of temporal equilibrium logic. In: Michael, L., Kakas, A. (eds.) Proceedings of the Fifteenth European Conference on Logics in Artificial Intelligence (JELIA'16). Lecture Notes in Artificial Intelligence, vol. 10021, pp. 159–173. Springer-Verlag (2016)
6. Brachman, R.J., Levesque, H.J.: Knowledge Representation and Reasoning. Elsevier (2004), `http://www.elsevier.com/wps/find/bookdescription.cws\_home/702602/description`
7. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
8. Cabalar, P., Kaminski, R., Morkisch, P., Schaub, T.: telingo = ASP + time. In: Balduccini, M., Lierler, Y., Woltran, S. (eds.) Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19). Lecture Notes in Artificial Intelligence, vol. 11481, pp. 256–269. Springer-Verlag (2019)
9. Cabalar, P., Kaminski, R., Schaub, T., Schuhmann, A.: Temporal answer set programming on finite traces. Theory and Practice of Logic Programming **18**(3-4), 406–420 (2018)
10. Cabalar, P., Diéguez, M., Laferrière, F., Schaub, T.: Past-present temporal programs over finite traces (2023), `https://arxiv.org/pdf/2307.12620.pdf`
11. Clark, K.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
12. De Giacomo, G., Vardi, M.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13). pp. 854–860. IJCAI/AAAI Press (2013)
13. Emerson, E.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, pp. 995–1072. MIT Press (1990)
14. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3**(4-5), 499–518 (2003)
15. Fages, F.: Consistency of Clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1**, 51–60 (1994)
16. Ferraris, P., Lee, J., Lifschitz, V.: A generalization of the Lin-Zhao theorem. Annals of Mathematics and Artificial Intelligence **47**(1-2), 79–101 (2006)
17. Gabbay, D.: The declarative past and imperative future: Executable temporal logic for interactive systems. In: Banieqbal, B., Barringer, H., Pnueli, A. (eds.) Proceedings of the Conference on Temporal Logic in Specification. Lecture Notes in Computer Science, vol. 398, pp. 409–448. Springer-Verlag (1987)

18. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: Garcia de la Banda, M., Pontelli, E. (eds.) Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08). Lecture Notes in Computer Science, vol. 5366, pp. 190–205. Springer-Verlag (2008)
19. Giacomo, G.D., Stasio, A.D., Fuggitti, F., Rubin, S.: Pure-past linear temporal and dynamic logic on finite traces. In: Bessiere, C. (ed.) Proceedings of the Twenty-ninth International Joint Conference on Artificial Intelligence, (IJCAI'20). pp. 4959–4965. ijcai.org (2020)
20. González, G., Baral, C., Cooper, P.A.: Modeling Multimedia Displays Using Action Based Temporal Logic, pp. 141–155. Springer US, Boston, MA (2002)
21. Lifschitz, V.: Answer set planning. In: de Schreye, D. (ed.) Proceedings of the International Conference on Logic Programming (ICLP'99). pp. 23–37. MIT Press (1999)
22. Lin, F., Zhao, J.: On tight logic programs and yet another translation from normal logic programs to propositional logic. In: Gottlob, G., Walsh, T. (eds.) Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03). pp. 853–858. Morgan Kaufmann Publishers (2003)
23. Pearce, D.: A new logical characterisation of stable models and answer sets. In: Dix, J., Pereira, L., Przymusinski, T. (eds.) Proceedings of the Sixth International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96). Lecture Notes in Computer Science, vol. 1216, pp. 57–70. Springer-Verlag (1997)
24. Pnueli, A.: The temporal logic of programs. In: Proceedings of the Eight-teenth Symposium on Foundations of Computer Science (FOCS'77). pp. 46–57. IEEE Computer Society Press (1977)
25. Sandewall, E.: Features and fluents: the representation of knowledge about dynamical systems, vol. 1. Oxford University Press, New York, NY, USA (1994)