Temporal Answer Set Programming *

Martín Diéguez

University of Corunna
Corunna, Spain
martin.dieguez@udc.es

Abstract

Answer Set Programming (ASP) has become a popular way for representing different kinds of scenarios from knowledge representation in Artificial Intelligence. Frequently, these scenarios involve a temporal component which must be considered. In ASP, time is usually represented as a variable whose values are defined by an extensional predicate with a finite domain. Dealing with a finite temporal interval has some disadvantages. First, checking the existence of a plan is not possible and second, it also makes difficult to decide whether two programs are *strongly equivalent*.

If we extend the syntax of Answer Set Programming by using temporal operators from temporal modal logics, then infinite time can be considered, so the aforementioned disadvantages can be overcome. This extension constitutes, in fact, a formalism called *Temporal Equilibrium Logic*, which is based on *Equilibrium Logic* (a logical characterisation of ASP).

Although recent contributions have shown promising results, Temporal Equilibrium Logic is still a novel paradigm and there are many gaps to fill. Our goal is to keep developing this paradigm, filling those gaps and turning it into a suitable framework for temporal reasoning.

1998 ACM Subject Classification F.4.1 Mathematical Logic/Temporal logic, I.2.3 Deduction and Theorem Proving/Logic programming

Keywords and phrases Answer Set Programming, Temporal Equilibrium Logic, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.445

1 Introduction and motivation

Modal Temporal Logics have become a popular paradigm used in protocol specification and representation of temporal scenarios. There are many techniques and tools that allow checking some desirable properties over temporal systems defined by this kind of logics, but most of them usually involve representational issues such as the limitation to propositional definition of the scenarios and several inherent problems like the frame problem [24].

Another paradigm used for temporal reasoning is *Answer Set Programming* [25, 23], whose roots are based on Logic Programming and Non-monotonic Reasoning. Thanks to the use of variables in the representation, reasoning with incomplete information and the existence of high performance solvers to compute the models (solutions) of the problem, ASP has become a suitable paradigm for representing and solving search problems in Artificial Intelligence. ASP has been commonly used to represent temporal scenarios [19] due to both the use of variables in the specifications and the easy definition of inertia rules which avoid the frame problem inherent to the formalisation of temporal problems. In fact, there are several action languages that use an ASP tool as a backend [14]. Despite all these

* This research was partially supported by Spanish MEC project TIN2009-14562-C05-04.


© Martín Diéguez;
licensed under Creative Commons License ND
Technical Communications of the 28th International Conference on Logic Programming (ICLP'12).
Editors: A. Dovier and V. Santos Costa; pp. 445–450

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

features, its main disadvantage is that time is represented by an extensional predicate with a finite domain, and consequently, a desired property like checking the existence of a plan for a particular representation is not possible. Furthermore, it makes checking whether two temporal theories are strongly equivalent more difficult (or impossible if arbitrary temporal formulas are allowed). In order to motivate our research, let us consider the following example:

▶ **Example 1.** A wolf, a goat and a cabbage are on one bank of a river and the boatman has to take them, safe and sound to the other side. The boatman can take at most one of them at a time. Finally, the following condition is applied: the wolf will eat the goat, and the goat will eat the cabbage, if they are left alone at the same side of the river.

```
1. time(0..max). opp(l,r). opp(r,l).
2. item(w). item(g). item(c).
3. eats(w,g). eats(g,c). object(b).
4. object(Z) :- item(Z).
5. next(I,I1) :- I1 = I+1, time(I1), time(I).

6. % Effect axiom for moving
7. at(X,A,T2):- at(X,B,T1), m(X,T1),
8.            opp(A,B), next(T1,T2).

9. % The boat is always moving
10. at(b,A,T2):- at(b,B,T1), opp(A,B),
11.                        next(T1,T2).

12. % Inertia
13. at(Y,A,T2) :- at(Y,A,T1), not at(Y,B,T2),
14.                     opp(A,B), next(T1,T2).

15. % State constraint
16. :- eats(X,Y), at(X,A,T), at(Y,A,T),
17.                at(b,B,T), opp(A,B).

18. % Unique value constraint
19. :- at(Y,A,T), at(Y,B,T), opp(A,B).

20. % Choice rules for action execution
21. m(X,T) :- not a(X,T), time(T),
22.                  item(X), T < max.
23. a(X,T) :- not m(X,T), time(T),
24.                  item(X), T < max.

25. % Action executability
26. :- m(X,T), at(b,A,T), at(X,B,T),
27.                         opp(A,B).

28. % Non-concurrent actions
29. :- m(X,T), m(Z,T), X != Z.

30. % Initial state: everything at
31. % left bank
32. at(Y,l,0):- object(Y).

33. % Goal: all items at right bank
34. g :- at(w,r,T), at(g,r,T), at(c,r,T).
35. :- not g.
```

As shown above, this representation encodes time in a variable `I` which is appended to every temporal predicate. This variable takes its values from {0,1,2,...,`max`}, being `max` a constant value which fixes the extension of the predicate `time/1`. Furthermore an example of an inertia rule is shown in lines 13 and 14. In an LTL formalisation, the information of this rule should be encoded in every fluent, considering every possible indirect effect.

Our proposal extends the syntax of ASP with operators to talk about time, like those defined in *Linear Temporal Logic* (LTL) [22], whose semantics is shown in Definition 2. To achieve this, we will make use of *Temporal Equilibrium Logic* (TEL) [10], which combines *Equilibrium Logic* [26] (a logical characterization of Answer Set Programming) and the aforementioned Linear Temporal Logic. This formalism introduces the definition of *Temporal Stable Models* (analogous to stable models [15] for any temporal theory). In TEL, every reference to time is encoded through modal operators. Therefore it is possible to consider an infinite-length narrative, overcoming the disadvantages of the current ASP approaches.

▶ **Definition 2. LTL semantics**
Let $\mathcal{I} = \{s_0, s_1, ..., s_n\}$ a set of set of atoms, i an integer ($i \geq 0$) and α an LTL formula. We say that $\mathcal{I}, i \models \alpha$ if:

- $\mathcal{I}, i \models p$ if $p \in s_i$.
- $\mathcal{I}, i \models \neg\alpha$ if $\mathcal{I}, i \not\models \alpha$.
- $\mathcal{I}, i \models \alpha \wedge \beta$ if $\mathcal{I}, i \models \alpha$ and $\mathcal{I}, i \models \beta$.
- $\mathcal{I}, i \models \alpha \vee \beta$ if $\mathcal{I}, i \models \alpha$ or $\mathcal{I}, i \models \beta$.
- $\mathcal{I}, i \models \Box\alpha$ if $\forall j \geq i, \mathcal{I}, j \models \alpha$.
- $\mathcal{I}, i \models \Diamond\alpha$ if $\exists j \geq i, \mathcal{I}, j \models \alpha$.
- $\mathcal{I}, i \models \bigcirc\alpha$ if $\mathcal{I}, i+1 \models \alpha$.
- $\mathcal{I}, i \models \alpha\mathcal{U}\beta$ if $\exists n \geq i, \mathcal{I}, n \models \beta$ and $\forall j, i \leq j < n, \mathcal{I}, j \models \alpha$.
- $\mathcal{I}, i \models \alpha\mathcal{R}\beta \Longleftrightarrow \mathcal{I}, i \models \neg(\neg\alpha\mathcal{U}\neg\beta)$.

2 Goal of research and preliminary results accomplished

As we have mentioned before, Temporal Equilibrium Logic is a novel research topic and of course, maturity of the results obtained so far is not comparable to the state of the art in Answer Set Programming. The foundations of Temporal Equilibrium Logic are defined in [10]. The property of strong equivalence among temporal logic programs has been studied in [3] where the authors define a translation that allows us to check this property in Linear Temporal Logic using an LTL model checker. The normal form of TEL programs is defined in [7]. Every TEL theory can be translated, like in the non-temporal case, into a set of implications, but in the case of TEL normal form, some of those can be under the scope of the LTL operator "$\Box$" (read "forever").

Recent results have focused on computing temporal equilibrium models. A first contribution [4] studies the computation of temporal equilibrium models for a subset of the already mentioned TEL normal form, which receives the name of *Splitable Temporal Logic Programs* (STLP's). These programs are characterised by the informal condition saying that the future does not depend on the past, that is, if the LTL operator "$\bigcirc$" (read "next state") appears in the body of a rule, it also applies to every atom in its head. By the splitting theorem [20], the authors have proven that the technique of *Loop Formulas* [13] can be applied to this kind of programs, so, its temporal equilibrium models can be computed by an LTL model checker. Such models are represented in terms of a Büchi automaton [6], which captures the whole behaviour of the system.

`STeLP` [1] [9] is the first tool designed to compute Temporal Equilibrium Models. It has its roots in the aforementioned work but it also adds some features to the input language like the use of variables in the specifications (like most ASP tools) and the use of arbitrary LTL expressions in the constraints. Example 1 would be formalised in `STeLP` as follows:

```
1. domain item(X).                                    % Domain declaration
2. static eats/2,object/1, opp/2.                     % Static predicates
3. fluent at/2.                                       % Fluent declaration
4. action m/1.                                        % Action declaration
5. opp(l,r). opp(r,l). eats(w,g). eats(g,c).
6. item(w). item(g). item(c). object(b).
7. object(Z) :- item(Z).
8. o at(X,A)::- at(X,B), m(X), opp(A,B).              % Effect axiom
9. o at(b,A)::- at(b,B), opp(A,B).                    % Effect axiom
10. ::- m(X), at(b,A), at(X,B), opp(A,B).             % Action executability
11. ::- at(X,A), at(X,B), opp(A,B).                   % Unique value
12. o at(X,A) ::- at(X,A), not o at(X,B), opp(A,B).   % Inertia
13. ::- eats(X,Y), at(X,A), at(Y,A), at(b,B), opp(A,B). % State constraint
14. m(X) ::- not a(X).                                % Choice action
15. a(X) ::- not m(X).                                % execution
```

[1] `http://kr.irlab.org/stelp_online`

```
16. ::- m(X), item(Z), m(Z), X != Z.          % Non-concurrent actions
17. at(Y,l):- object(Y).                      % Initial state
18. g ::- at(w,r), at(g,r), at(c,r).          % Goal state
19. :- always not g.                          % Goal must be satisfied
```

The input language of this tool is very similar to common ASP language but introduces two new operators in the definition of the rules: "o" which corresponds to "◯" in LTL and "::-" which replaces the traditional ":-" of ASP but under the implicit scope of the LTL operator "□" , that is, the rule must be satisfied in every instant of time. Furthermore, `STeLP` deals with arbitrary LTL formulas in the body of constraints (as shown in line 30, in the example above), which is useful, for instance, to check whether a temporal representation has a plan.

Like most ASP solvers, `STeLP` grounds the input program before computing the models. In this case, traditional grounding algorithms are not directly applicable because ground atoms may change their truth value along time. As a first approach to temporal grounding, the user has to identify a family of predicates, called `static`, whose truth value remains constant along time. Furthermore, to guarantee the domain independence property of the program[2], the input program must satisfy the following *safety* condition:

1. Any variable X occurring in a rule $B \rightarrow H$ or $\Box(B \rightarrow H)$, also occurs in some positive static predicate in B.
2. Rules of the form $B \rightarrow H$, where at least one static predicate occurs in H, only contain static predicates.

It is easy to see that, under this condition, static predicates cannot depend on the non-static ones. `STeLP` first computes a stable model of the "static program" and then uses it to ground the rest of the rules by instantiating any positive static predicate in their bodies.

Since static predicates must occur in any rule, this tool allows defining global variable names with a fixed domain (by the keyword *domain*), in a similar way to lparse[3]. For instance, in the example above `item(X)` means that any rule referring to variable X is implicitly extended by including the implicitly declared static predicate `item(X)`.

Apart from splitable programs, arbitrary temporal theories have been studied in [8]. Here, the authors present an algorithm for computing temporal equilibrium models based on several operations over Büchi automata. Furthermore, that paper achieves first results about the complexity of TEL, whose satisfiability is decidable in ExpSpace and, at least, PSpace-hard.

3 Current status of research and expected achievements

As we have said before, several results have been accomplished but there is still much work in progress. We are currently trying to improve the grounding algorithm implemented in `STeLP`. Until now, only static predicates had been considered during the instantiation of a rule. This causes a generation of irrelevant ground rules that increase the size of the resulting ground LTL theory while they could be easily detected and removed by a simple analysis of the temporal program. A new algorithm for temporal grounding has been considered

[2] that is, its set of stable models does not change with respect to the addition of constants
[3] `http://www.tcs.hut.fi/Software/smodels/src/lparse-1.1.2.tar.gz`

in [2]. In this paper, the authors have proven that the safety condition of DLV [17] (every variable which occurs in a rule, must occur in a positive predicate in its body) is enough to guarantee the property of domain independence for STLP's. Considering an STLP under this safety condition, authors also define a translation to compute its set of *derivable facts* (a set of ground atoms that are potentially members of at least one of its temporal equilibrium models) which is used to determine, in grounding time, if a ground rule can be removed from the final ground program. If a ground rule contains an atom in its positive body which does not belong to the set of derivable facts of the program, this rule can be omitted is not generated.

As a future work, on the one hand, we expect to keep developing the foundations of Temporal Equilibrium Logic. In [16], it has been shown that LTL has the same expressive power as a First Order theory of linear ordering. We conjecture that there is a corresponding relation between Temporal Equilibrium Logic and *Quantified Equilibrium Logic*.

On the other hand, we are also concerned about practical results. For instance, one of our constant goals is improving the performance of `STeLP` and its scalability for larger programs. The main bottlenecks are the grounding algorithm (it has already been improved) and the computation of the temporal equilibrium models. The latter is made translating a final LTL formula into a Büchi automaton (which is the costliest process). In some cases, if we just want to check the satisfiability of a formula, there are tools like TSPASS [21] that are based on resolution techniques and can check satisfiability without computing the whole automaton. Depending on users' requirements, these tools are more suitable than Spot [11], the current backend used by `STeLP`, due to their shorter response time.

In order to find practical applications for our work, one possibility is using STeLP as a backend for ASP-based action languages such as $\texttt{DLV}^k$ [12] or `ALM` [14]. In the current implementation of ASP-based action languages, checking whether a temporal or planning problem has a solution would mean exhausting the set of possible transitions (assuming we deal with a finite set of fluents and actions) and checking the appearance of repeated states outside the ASP program. STeLP offers the possibility of a fully automated method that builds the automaton from the generated temporal formulas. It could be plugged as an alternative backend for action languages when we are interested in verifying complex temporal properties or checking the existence of a plan. In fact our plan is to adapt the implementation of the action language `ALM` to use `STeLP` instead of its current backend.

Finally, like in every Ph.D. thesis, it is very important to make a comparison of our proposal with other frameworks for nonmonotonic temporal reasoning present in the literature, for instance [1], [18] and [5]. Finding a connection between Temporal Equilibrium Logic and those approaches would be useful to perceive the advantages and disadvantages of TEL with respect to other strategies.

References

1 M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8(3):277–295, 1989.

2 F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, and C. Vidal. Paving the Way for Temporal Grounding. In *ICLP '12*, 2012.

3 F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Strongly Equivalent Temporal Logic Programs. In *JELIA'08*, volume 5293 of *LNCS*, pages 8–20. Springer, 2008.

4 F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Loop Formulas for Splitable Temporal Logic Programs. In *LPNMR'11*, volume 6645 of *LNCS*, pages 80–92. Springer, 2011.

5 C. Baral and J. Zhao. Non-monotonic Temporal Logics for Goal Specification. *IJCAI 2007*.

6 R. Büchi. On a decision method in restricted second-order arithmetic. In *International Congress on Logic, Method and Philosophical Science'60*, pages 1–11, 1962.

7 P. Cabalar. A Normal Form for Linear Temporal Equilibrium Logic. In *JELIA'10*, volume 6341 of *LNCS*, pages 64–76. Springer, 2010.

8 P. Cabalar and S. Demri. Automata-based computation of temporal equilibrium models. In *21st International Workshop on Logic Program Synthesis and Transformation (LOPSTR'11)*, LNCS. Springer, 2011.

9 P. Cabalar and M. Diéguez. STeLP - a Tool for Temporal Answer Set Programming. In *LPNMR'11*, volume 6645 of *LNCS*, pages 370–375. Springer, 2011.

10 P. Cabalar and G. P. Vega. Temporal Equilibrium Logic: a first approach. In *11th International Conference on Computer Aided Systems Theory (EUROCAST'07)*, volume 4739 of *LNCS*, pages 241–248. Springer, 2007.

11 A. Duret-Lutz and D. Poitrenaud. SPOT: an Extensible Model Checking Library Using Transition-based Generalized Büchi Automata. In *Proc. of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*. IEEE Computer Society, 2004.

12 T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. In *1st International Conference on Computational Logic (CL '00)*. Springer, 2000.

13 P. Ferraris, J. Lee, and V. Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):79–101, 2006.

14 M. Gelfond and D. Inclezan. Yet Another Modular Action Language. In *2nd International Workshop on Software Engineering for Answer Set Programming (SEA'09)*, pages 64–78, 2009.

15 M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *ICLP'88*, pages 1070–1080. MIT Press, Cambridge, MA, 1988.

16 J. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California at Los Angeles, 1968.

17 N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7:499–562, 2006.

18 H. J. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2:159–178, 1998.

19 V. Lifschitz. Answer Set Programming and Plan Generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

20 V. Lifschitz and H. Turner. Splitting a Logic Program. In *ICLP'94*, pages 23–37. MIT press, 1994.

21 M. Ludwig and U. Hustadt. Implementing a fair monodic temporal logic prover. *AI Communications*, 23(2-3):69–96, 2010.

22 Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

23 V. Marek and M. Truszczyński. *Stable models and an alternative logic programming paradigm*, pages 169–181. Springer-Verlag, 1999.

24 J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence Journal*, 4:463–512, 1969.

25 I. Niemelä. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

26 D. Pearce. A new logical characterisation of stable models and answer sets. In *Selected papers from the Non-Monotonic Extensions of Logic Programming (NMELP'96)*, volume 1216 of *LNAI*, pages 57–70. Springer, 1996.