Telingo = ASP + Time

— An abridged report —

Pedro Cabalar[1], Roland Kaminski[2], and Torsten Schaub[2]

[1] University of Corunna, Spain
[2] University of Potsdam, Germany

For representing and reasoning about dynamic systems in Answer Set Programming (ASP; [8]), two major research avenues have been explored in ASP: temporal extensions of Equilibrium Logic [1], the host logic of ASP, and (several) action languages [7]. Although both constitute the main directions of non-monotonic temporal systems, their prevalence lags way behind the usage of plain ASP for modeling dynamic domains.

To address this, we recently proposed in [4] an alternative combination of the logics of HT and LTL whose semantics rests upon *finite* traces. On the one hand, this amounts to a restriction of THT and TEL to finite traces. On the other hand, this is similar to the restriction of LTL to LTL_f advocated by [5]; see also [2]. Our new approach, dubbed TEL_f, has the following advantages. First, it is readily implementable via ASP technology. Second, it can be reduced to a normal form which is close to logic programs and much simpler than the one obtained for TEL. Finally, its temporal models are finite and offer a one-to-one correspondence to plans. Interestingly, TEL_f also sheds light on concepts and methodology used in incremental ASP solving when understanding incremental parameters as time points.

We have implemented our approach in the system `telingo`[3] that deals with so-called present-centered TEL_f programs (cf. [4]) that are expressible in the full (non-ground) input language of `clingo` extended with temporal operators. In addition, `telingo` offers several syntactic extensions to facilitate temporal modeling: First, next operators can be used in singular heads and, second, arbitrary temporal formulas can be used in integrity constraints. All syntactic extensions beyond the normal form of TEL_f formulas are compiled away by means of the translation (used in the normal form proof). The resulting present-centered TEL_f programs are then processed according the point-wise translation, proposed in [4], that relies on the composition of logic program modules [9]..

To facilitate the use of temporal operators "previous" $\bullet$ and "next" $\circ$, `telingo` allows us to express them by adding leading or trailing quotes to the predicate names of atoms, respectively. For instance, the temporal literals $\bullet p(a)$ and $\circ q(b)$ can be expressed by `'p(a)` and `q'(b)`, respectively. For example, consider the representation of the sentence *"A robot cannot lift a box unless its capacity exceeds the box's weight plus that of all held objects"*:

```
:- lift(R,B), robot(R), box(B,W),
   #sum { C : capacity(R,C); -V,O : 'holding(R,O,V) } < W.
```

[3] `https://github.com/potassco/telingo`

Atom `'holding(R,O,V)` expresses what the robot was holding at the *previous* time point.

The distinction between different types of temporal rules is done in `telingo` via `clingo`'s `#program` directives [6], which allow us to partition programs into subprograms. More precisely, each rule in `telingo`'s input language is associated with a temporal rule r of form $(b_1 \wedge \cdots \wedge b_n \rightarrow a_1 \vee \cdots \vee a_m)$ over $\{a, \neg a, \bullet a, \neg \bullet a \mid a \in \mathcal{A}\}$, as detailed in [4], and interpreted as r, $\widehat{\bigcirc}\Box r$, or $\Box(\mathbb{F} \rightarrow r)$ depending on whether it occurs in the scope of a program declaration headed by `initial`, `dynamic`, or `final`, respectively. Additionally, `telingo` offers `always` for gathering rules preceded by $\Box$ (thus dropping $\widehat{\bigcirc}$ from dynamic rules). A rule outside any such declaration is regarded to be in the scope of `initial`. This allows us to represent the TEL$_f$ program

$$\{ \rightarrow a, \quad \widehat{\bigcirc}\Box(\bullet a \rightarrow b), \quad \Box(\mathbb{F} \rightarrow (\neg b \rightarrow \bot)) \} \tag{1}$$

in the two alternative ways shown in Table 1.

```
#program initial.
a.

#program dynamic.
b :- 'a.

#program final.
  :- not b.
```

```
#program always.
a :- &initial.


b :- 'a.


  :- not b, &final.
```

Table 1. Two alternative `telingo` encodings for the TEL$_f$ program in (1)

As mentioned, `telingo` allows us to use nested temporal formulas in integrity constraints as well as in negated form in place of temporal literals within rules. This is accomplished by encapsulating temporal formulas like φ in expressions of the form '`&tel { `φ` }`'. To this end, the full spectrum of temporal operators is at our disposal. They are expressed by operators built from `<` and `>` depending on whether they refer to the past or the future, respectively. So, `</1`, `<?/2`, and `<*/2` stand for past operators $\bullet$, S ("since"), and T ("trigger"), and `>/1`, `>?/2`, `>*/2` for future operators $\circ$, $\mathbb{U}$ ("until"), $\mathbb{R}$ ("release"). Accordingly, `<*/1`, `<?/1`, `<:/1` represent $\blacksquare$ ("always"), $\blacklozenge$ ("eventually"), $\widehat{\bullet}$ ("weak previous"), and analogously their future counterparts. I and $\mathbb{F}$ are are represented by `&initial` and `&final`. This is complemented by Boolean connectives `&`, `|`, `~`, etc. For example, the integrity constraint '$\mathit{shoot} \wedge \blacksquare \mathit{unloaded} \wedge \bullet\blacklozenge \mathit{shoot} \rightarrow \bot$' is expressed as follows.

```
:- shoot, &tel { <* unloaded & < <? shoot }.
```

Once `telingo` has translated an (extended) TEL$_f$ program into a regular one, it is incrementally solved by `clingo`'s multi-shot solving engine [6].

References

1. F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, and C. Vidal. Temporal equilibrium logic: a survey. *Journal of Applied Non-Classical Logics*, 23(1-2):2–24, 2013.
2. J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In Y. Gil and R. Mooney, editors, *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06)*, pages 788–795. AAAI Press, 2006.
3. A. Bosser, P. Cabalar, M. Diéguez, and T. Schaub. Introducing temporal stable models for linear dynamic logic. In M. Thielscher, F. Toni, and F. Wolter, editors, *Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'18)*, pages 12–21. AAAI Press, 2018.
4. P. Cabalar, R. Kaminski, T. Schaub, and A. Schuhmann. Temporal answer set programming on finite traces. *Theory and Practice of Logic Programming*, 18(3-4):406–420, 2018.
5. G. De Giacomo and M. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 854–860. IJCAI/AAAI Press, 2013.
6. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 2018. To appear.
7. M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 3(6):193–210, 1998.
8. V. Lifschitz. Answer set planning. In D. de Schreye, editor, *Proceedings of the International Conference on Logic Programming (ICLP'99)*, pages 23–37. MIT Press, 1999.
9. E. Oikarinen and T. Janhunen. Modular equivalence for normal logic programs. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, pages 412–416. IOS Press, 2006.