Paving the Way for Temporal Grounding*

Felicidad Aguado, Pedro Cabalar, Martín Diéguez, Gilberto Pérez, and Concepción Vidal

University of Corunna
Corunna, Spain
{aguado,cabalar,martin.dieguez,gperez,eicovima}@udc.es

Abstract

In this paper we consider the problem of introducing variables in temporal logic programs under the formalism of *Temporal Equilibrium Logic* (TEL), an extension of Answer Set Programming (ASP) for dealing with linear-time modal operators. We provide several fundamental contributions that pave the way for the implementation of a grounding process, that is, a method that allows replacing variables by ground instances in all the possible (or better, relevant) ways.

1998 ACM Subject Classification D.1.6 Logic Programming, F.4.1 Mathematical Logic

Keywords and phrases ASP, linear temporal logic, grounding, temporal equilibrium logic

Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.290

1 Introduction

Many application domains and example scenarios from Answer Set Programming (ASP) [13, 11] contain a dynamic component, frequently representing transition systems over discrete time. However, temporal reasoning in ASP tends to be quite rudimentary, just treating time as an integer variable which is grounded for a finite interval[1]. To cope with more elaborated temporal reasoning, in [1] a formalism called *Temporal Equilibrium Logic* (TEL) was proposed. TEL is syntactically identical to propositional *Linear-time Temporal Logic* (LTL) [16], but semantically, it relies on a temporal extension of *Equilibrium Logic* [14], the most general and best studied logical characterisation of stable models (or answer sets) [7]. A recent work [2] introduced a reduction of TEL into regular LTL, for a syntactic subclass of temporal theories called *Splitable Temporal Logic Programs*. Although this syntactic fragment is a strict subset of the TEL normal form obtained in [4], it deals with temporal rules in which, informally speaking, "past does not depend on the future," something general enough to cover most (if not all) existing examples of ASP temporal scenarios. The reduction was implemented in a tool, `STeLP`[2] [5], that computes the temporal stable models of a given program, showing the result in the form of a Büchi automaton.

Although the theoretical results on which `STeLP` is based are restricted to the propositional case, the input language was extended with the introduction of variables. This was done imposing some strict limitations on the syntax, forcing that any variable instance is not only *safe* (that is, occurring in the positive body of the rule) but also "typed" by a *static* predicate, i.e., a predicate whose extent does not vary along time. In many cases, this restriction implied the generation of irrelevant ground rules that increase the size of the

* This research was partially supported by Spanish MEC project TIN2009-14562-C05-04.
[1] More elaborated approaches [12] deal with arbitrary temporal distances by using a constraint satisfaction tool as a backend.
[2] `http://kr.irlab.org/stelp_online`

Technical Communications of the 28th International Conference on Logic Programming (ICLP'12).
Editors: A. Dovier and V. Santos Costa; pp. 290–300

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

```
static city/1, car/1, road/2.
                    o at(X,A)  ::- driveto(X,A), car(X), city(A).   % (1)
driveto(X,B) v no_driveto(X,B) ::- at(X,A), car(X), road(A,B).      % (2)
o at(X,A)     ::- at(X,A), not o no_at(X,A), car(X), city(A).       % (3)
  no_at(X,A) ::- at(X,B), A!=B, car(X), city(A), city(B).           % (4)
::- at(X,A), at(X,B), A!=B, car(X), city(A), city(B).               % (5)
```

Figure 1 A simple car driving scenario.

resulting ground LTL theory while they could be easily detected and removed by a simple analysis of the temporal program. Furthermore, the treatment of variables had not been proved to be sound with respect to the important property of *domain independence* [3] – essentially, a program is domain independent when its stable models do not vary under the arbitrary addition of new constants. Although the DLV definition of safe variables guarantees domain independence, there was no formal proof for temporal logic programs under TEL.

In this paper we provide several fundamental results that pave the way for an improved grounder for temporal logic programs with variables. The rest of the paper is organised as follows. In the next section, we explain our motivations using an illustrative example. In Section 3 we introduce the first order extension of TEL and provide some basic definitions, explaining the syntactic form for our input language. Next, we study the relaxed definition of safe variables and prove that it guarantees domain independence. Section 5 defines the concept of *derivable facts*, explaining how they can be computed and used afterwards to generate smaller ground theories. Finally, Section 6 concludes the paper.

2 A motivating example

For a better understanding of our motivations, let us consider a simple illustrative example.

▶ **Example 1.** Suppose we have a set of cars placed at different cities and, at each transition, we can drive a car from one city to another in a single step, provided that there is a road connecting them. ◀

Figure 1 contains a possible representation of this scenario in the language of STeLP. In the rules, operator 'o' stands for "next" whereas '::-' corresponds to the standard ASP conditional ':-', but holding at all time points. Rule (1) is the effect axiom for driving car X to city A. The disjunctive rule (2) is used to generate possible occurrences of actions in a non-deterministic way. Rules (3) and (4) represent the inertia[3] of fluent at(X,A). Finally, rule (5) just forbids that a car is at two different cities simultaneously.

As we can see in the first line, predicates city/1, car/1 and road/2 are declared to be static. The scenario would be completed with rules for static predicates. These rules conform what we call the *static program* and can only refer to static predicates without containing temporal operators. An example of a static program for this scenario could be:

```
road(A,B) :- road(B,A).  % roads are bidirectional
city(A) :- road(A,B).
car(1). car(2).
road(lisbon,madrid). road(madrid,paris). road(boston,ny). road(ny,nj).
```

[3] Auxiliary predicates no_driveto(X,B) and no_at(X,A) play the role here of strong negation.

Additionally, our temporal program would contain rules describing the initial state like, for instance, the pair of facts:

```
at(1,madrid). at(2,ny).
```

Note that all variables in a rule are always in some atom for a static predicate in the positive body. This sometimes makes rule bodies quite long and slightly redundant. The current grounding process performed by `STeLP` just consists in feeding the static program to `DLV` and, once it provides an extension for all the static predicates, each temporal rule is instantiated for each possible substitution of variables according to static predicates. In our running example, for instance, `DLV` provides a unique model[4] for the static program containing the facts:

```
car(1), car(2), city(lisbon), city(madrid), city(paris), city(boston),
city(ny), city(nj), road(lisbon,madrid), road(madrid,lisbon),
road(madrid,paris), road(paris,madrid), road(boston,ny),
road(ny,boston), road(ny,nj), road(nj,ny)
```

With these data, rule `(1)` generates 12 ground instances, since we have two possible cars for `X` and six possible cities for `A`. Similarly, rule `(4)` would generate 60 instances as there are 30 pairs `A,B` of different cities and two cars for `X`. Many of these ground rules, however, are irrelevant. Consider, for instance, the following pair of generated rules:

```
o at(1,ny) ::- driveto(1,ny).
no_at(1,paris) ::- at(1,ny).
```

corresponding, respectively, to possible instantiations of `(1)` and `(4)`. In both cases, the body refers to a situation where car 1 is located or will drive to New York, while we can observe that it was initially at Madrid and that the European roadmap is disconnected from the American one. Of course, one could additionally encode a static reachability predicate to force that rule instances refer to reachable cities for a given car, but this would not be too transparent or elaboration tolerant. One would expect that the grounder was capable of detecting these "non-derivable" cases ignoring them in the final ground theory, if possible.

On the other hand, if we forget, for a moment, the temporal operators and we consider the definition of safe variables used in `DLV`, one may also wonder whether it is possible to simply require that each variable occurs in the positive body of rules, without needing to refer to static predicates mandatorily. Figure 2 contains a possible variation of the same scenario allowing this possibility. Our goal is allowing this new, more flexible definition of safe variables and exploiting, if possible, the information in the temporal program to reduce the set of generated ground rules.

3 Temporal Quantified Equilibrium Logic

Syntactically, we consider function-free first-order languages $\mathcal{L} = \langle C, P\rangle$ built over a set of *constant* symbols, C, and a set of *predicate* symbols, P. Using $\mathcal{L}$, connectors and variables, an $\mathcal{L} = \langle C, P\rangle$-*formula* F is defined following the grammar:

$$F ::= p \mid \bot \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \rightarrow F_2 \mid \bigcirc F \mid \Box F \mid \Diamond F \mid \forall x F(x) \mid \exists x F(x)$$

[4] If the static program yields several stable models, each one generates a different ground theory whose temporal stable models are computed independently.

```
static city/1, car/1, road/2.
                  o at(X,A)    ::- driveto(X,A).
driveto(X,B) v no_driveto(X,B) ::- at(X,A), road(A,B).
                  o at(X,A)    ::- at(X,A), not o no_at(X,A).
                    no_at(X,A) ::- at(X,B), A!=B, city(A).
                                ::- at(X,A), at(X,B), A!=B.
```

Figure 2 A possible variation of the cars scenario.

where $p \in P$ is an atom, x is a variable and $\bigcirc$, $\Box$ and $\Diamond$ respectively stand for "next", "always" and "eventually." A *theory* is a finite set of formulas. We use the following derived operators and notation: $\neg F \stackrel{\text{def}}{=} F \rightarrow \bot$, $\top \stackrel{\text{def}}{=} \neg\bot$ and $F \leftrightarrow G \stackrel{\text{def}}{=} (F \rightarrow G) \wedge (G \rightarrow F)$ for any formulas F, G. An *atom* is any $p(t_1, \ldots, t_n)$ where $p \in P$ is a predicate with n-arity and each t_i is a term (a constant or a variable) in its turn. We say that a term or a formula is *ground* if it does not contain variables. An $\mathcal{L}$-*sentence* or closed-formula is a formula without free-variables. The application of i consecutive $\bigcirc$'s is denoted as follows: $\bigcirc^i\varphi \stackrel{\text{def}}{=} \bigcirc(\bigcirc^{i-1}\varphi)$ for $i > 0$ and $\bigcirc^0\varphi \stackrel{\text{def}}{=} \varphi$. A *temporal fact* is a construction of the form $\bigcirc^i A$ where A is an atom. If D is a non-empty set, we denote by $At(D, P)$ the set of ground atomic sentences of the language $\langle D, P\rangle$. For the semantics, we will also define a mapping $\sigma\colon C \cup D \rightarrow D$ such that $\sigma(d) = d$ for all $d \in D$.

A first-order LTL-interpretation is a structure $\langle(D, \sigma), \mathbf{T}\rangle$ where D is a non-empty set (the *domain*), σ is a mapping as defined above (the interpretation of constants) and $\mathbf{T}$ is an infinite sequence of sets of ground atoms $\mathbf{T} = \{T_i\}_{i\geq 0}$. Intuitively, $T_i \subseteq At(D, P)$ contains those ground atoms that are true at situation i. Given two LTL-interpretations $\mathbf{H}$ and $\mathbf{T}$ we say that $\mathbf{H}$ is *smaller than* $\mathbf{T}$, written $\mathbf{H} \leq \mathbf{T}$, when $H_i \subseteq T_i$ for all $i \geq 0$. As usual, $\mathbf{H} < \mathbf{T}$ stands for: $\mathbf{H} \leq \mathbf{T}$ and $\mathbf{H} \neq \mathbf{T}$. We define the ground temporal facts associated to $\mathbf{T}$ as follows: $Facts(\mathbf{T}) \stackrel{\text{def}}{=} \{\bigcirc^i p \mid p \in T_i\}$. It is easy to see that $\mathbf{H} \leq \mathbf{T}$ iff $Facts(\mathbf{H}) \subseteq Facts(\mathbf{T})$.

▶ **Definition 2.** A *temporal-here-and-there* $\mathcal{L}$-structure with static domains, or a **TQHT**-*structure*, is a tuple $\mathcal{M} = \langle(D, \sigma), \mathbf{H}, \mathbf{T}\rangle$ where $\langle(D, \sigma), \mathbf{H}\rangle$ and $\langle(D, \sigma), \mathbf{T}\rangle$ are two LTL-interpretations satisfying $\mathbf{H} \leq \mathbf{T}$. ◀

A **TQHT**-structure of the form $\mathcal{M} = \langle(D, \sigma), \mathbf{T}, \mathbf{T}\rangle$ is said to be *total*. If $\mathcal{M} = \langle(D, \sigma), \mathbf{H}, \mathbf{T}\rangle$ is a **TQHT**-structure and k any positive integer, we denote by $(\mathcal{M}, k) = \langle(D, \sigma), (\mathbf{H}, k), (\mathbf{T}, k)\rangle$ the temporal-here-and-there $\mathcal{L}$-structure with $(\mathbf{H}, k) = \{H_i\}_{i\geq k}$ and $(\mathbf{T}, k) = \{T_i\}_{i\geq k}$. The satisfaction relation for $\mathcal{M} = \langle(D, \sigma), \mathbf{H}, \mathbf{T}\rangle$ is defined recursively forcing us to consider formulas from $\langle C \cup D, P\rangle$. Formally, if φ is an $\mathcal{L}$-sentence for the atoms in $At(C \cup D, P)$, then:

- If $\varphi = p(t_1, \ldots, t_n) \in At(C \cup D, P)$, then

$$\mathcal{M} \models p(t_1, \ldots, t_n) \quad \text{iff} \quad p(\sigma(t_1), \ldots, \sigma(t_n)) \in H_0.$$
$$\mathcal{M} \models t = s \quad \text{iff} \quad \sigma(t) = \sigma(s)$$

- For $\bot$, $\wedge$ and $\vee$, as usual.
- $\mathcal{M} \models \varphi \rightarrow \psi$ iff $\langle(D, \sigma), w, \mathbf{T}\rangle \not\models \varphi$ or $\langle(D, \sigma), w, \mathbf{T}\rangle \models \psi$ for all $w \in \{\mathbf{H}, \mathbf{T}\}$
- $\mathcal{M} \models \bigcirc\varphi$ if $(\mathcal{M}, 1) \models \varphi$.
- $\mathcal{M} \models \Box\varphi$ if $\forall j \geq 0$, $(\mathcal{M}, j) \models \varphi$
- $\mathcal{M} \models \Diamond\varphi$ if $\exists j \geq 0$, $(\mathcal{M}, j) \models \varphi$

- $\langle (D,\sigma), \mathbf{H}, \mathbf{T}\rangle \models \forall x \varphi(x)$ iff $\langle (D,\sigma), w, \mathbf{T}\rangle \models \varphi(d)$ for all $d \in D$ and for all $w \in \{\mathbf{H}, \mathbf{T}\}$.
- $\mathcal{M} \models \exists x \varphi(x)$ iff $\mathcal{M} \models \varphi(d)$ for some $d \in D$.

The resulting logic is called *Quantified Temporal Here-and-There Logic with static domains*, and denoted by **SQTHT** or simply by **QTHT**. It is not difficult to see that, if we restrict to total **TQHT**-structures, $\langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle \models \varphi$ iff $\langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle \models \varphi$ in first-order LTL. Furthermore, the following property can be easily checked by structural induction.

▶ **Proposition 3.** *For any formula φ, if $\langle (D,\sigma), \mathbf{H}, \mathbf{T}\rangle \models \varphi$, then $\langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle \models \varphi$*

A *theory* Γ is a set of $\mathcal{L}$-sentences. An interpretation $\mathcal{M}$ is a model of a theory Γ, written $\mathcal{M} \models \Gamma$, if it satisfies all the sentences in Γ.

▶ **Definition 4** (Temporal Equilibrium Model)**.** A *temporal equilibrium* model of a theory Γ is a total model $\mathcal{M} = \langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle$ of Γ such that there is no $\mathbf{H} < \mathbf{T}$ satisfying $\langle (D,\sigma), \mathbf{H}, \mathbf{T}\rangle \models \Gamma$. ◀

If $\mathcal{M} = \langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle$ is a temporal equilibrium model of a theory Γ, we say that the First-Order LTL interpretation $\langle (D,\sigma), \mathbf{T}\rangle$ is a *temporal stable model* of Γ. We write $TSM(\Gamma)$ to denote the set of temporal stable models of Γ. The set of *credulous consequences* of a theory Γ, written $CredFacts(\Gamma)$ contains all the temporal facts that occur at some temporal stable model of Γ, that is:

$$CredFacts(\Gamma) \stackrel{\text{def}}{=} \bigcup_{\langle (D,\sigma), \mathbf{T}\rangle \in TSM(\Pi)} Facts(\mathbf{T})$$

A property of TEL directly inherited from Equilibrium Logic (see Proposition 5 in [15]) is the following:

▶ **Proposition 5** (Cummulativity for negated formulas)**.** *Let Γ be some theory and let $\neg\varphi$ be some formula such that $\mathcal{M} \models \neg\varphi$ for all temporal equilibrium models of Γ. Then, the theories Γ and $\Gamma \cup \{\neg\varphi\}$ have the same set of temporal equilibrium models.* ◀

It is well-known that stable models (and so Equilibrium Logic) do not satisfy cummulativity in the general case: that is, if a formula is satisfied in all the stable models, adding it to the program may vary the consequences we obtain. However, when we deal with negated formulas, Proposition 5 tells us that cummulativity is guaranteed.

In this work, we will further restrict the study to a syntactic subset called *splitable*[5] temporal formulas (STF) which will be of one of the following types:

$$B \wedge N \rightarrow H \tag{1}$$
$$B \wedge \bigcirc B' \wedge N \wedge \bigcirc N' \rightarrow \bigcirc H' \tag{2}$$
$$\Box(B \wedge \bigcirc B' \wedge N \wedge \bigcirc N' \rightarrow \bigcirc H') \tag{3}$$

where B and B' are conjunctions of atomic formulas, N and N' are conjunctions of $\neg p$, being p an atomic formula and H and H' are disjunctions of atomic formulas.

[5] The name *splitable* refers to the fact that these programs can be splitted using [10] thanks to the property that rule heads never refer to a time point previous to those referred in the body.

▶ **Definition 6.** A *splitable temporal logic program* (*STL-program* for short) is a finite set of sentences like

$$\varphi = \forall x_1 \forall x_2 \ldots \forall x_n \psi,$$

where ψ is a splitable temporal formula with $x_1, x_2, \ldots, x_n$ free variables.

We will also accept in an STL-program an implication of the form $\Box(B \wedge N \rightarrow H)$ (that is, containing $\Box$ but not any $\bigcirc$) understood as an abbreviation of the pair of STL-formulas:

$$\begin{array}{rcl} B \wedge N & \rightarrow & H \\ \Box(\bigcirc B \wedge \bigcirc N & \rightarrow & \bigcirc H) \end{array}$$

▶ **Example 7.** The following theory Π_7 is an STL-program:

$$\neg p \rightarrow q \tag{4}$$
$$q \wedge \neg \bigcirc r \rightarrow \bigcirc p \tag{5}$$
$$\Box(q \wedge \neg \bigcirc p \rightarrow \bigcirc q) \tag{6}$$
$$\Box(r \wedge \neg \bigcirc p \rightarrow \bigcirc r \vee \bigcirc q) \tag{7}$$

For an example including variables, the encoding of Example 1 in Figure 2 is also an STL-program Π_1 whose logical representation corresponds to:

$$\Box(\ Driveto(x,a) \rightarrow \bigcirc At(x,a)\ ) \tag{8}$$
$$\Box(\ At(x,a) \wedge Road(a,b) \rightarrow Driveto(x,b) \vee NoDriveto(x,b)\ ) \tag{9}$$
$$\Box(\ At(x,a) \wedge \neg \bigcirc NoAt(x,a) \rightarrow \bigcirc At(x,a)\ ) \tag{10}$$
$$\Box(\ At(x,b) \wedge City(a) \wedge a \neq b \rightarrow NoAt(x,a)\ ) \tag{11}$$
$$\Box(\ At(x,a) \wedge At(x,b) \wedge a \neq b \rightarrow \bot\ ) \tag{12}$$

Remember that all rule variables are implicitly universally quantified. For simplicity, we assume that inequality is a predefined predicate.

An STL-program is said to be *positive* if for all rules (1)-(3), N and N' are empty (an empty conjunction is equivalent to $\top$). An STL-program is said to be *normal* if it contains no disjunctions, i.e., for all rules (1)-(3), H and H' are atoms. Given a propositional combination φ of temporal facts with $\wedge, \vee, \bot, \rightarrow$, we denote φ^i as the formula resulting from replacing each temporal fact A in φ by $\bigcirc^i A$. For a formula $r = \Box\varphi$ like (3), we denote by r^i the corresponding φ^i. For instance, $(6)^i = (\bigcirc^i q \wedge \neg \bigcirc^{i+1} p \rightarrow \bigcirc^{i+1} q)$. As $\bigcirc$ behaves as a linear operator in THT, in fact $F^i \leftrightarrow \bigcirc^i F$ is a THT tautology.

▶ **Definition 8** (expanded program)**.** Given an STL-program Π for signature Σ we define its *expanded program* Π^∞ as the infinitary logic program containing all rules of the form (1), (2) in Π plus a rule r^i per each rule r of the form (3) in Π and each integer value $i \geq 0$. ◀

The program Π_7^∞ would therefore correspond to (4), (5) plus the infinite set of rules:

$$\bigcirc^i q \wedge \neg \bigcirc^{i+1} p \rightarrow \bigcirc^{i+1} q \qquad\qquad \bigcirc^i r \wedge \neg \bigcirc^{i+1} p \rightarrow \bigcirc^{i+1} r \vee \bigcirc^{i+1} q$$

for $i \geq 0$. We can interpret the expanded program as an infinite non-temporal program where the signature is the infinite set of atoms of the form $\bigcirc^i p$ with $p \in At$ and $i \geq 0$.

▶ **Theorem 9** (Theorem 1 in [2])**.** *$\langle \mathbf{T}, \mathbf{T} \rangle$ is a temporal equilibrium model of Π iff $\{\bigcirc^i p \mid p \in T_i, i \geq 0\}$ is a stable model of Π^∞ under the (infinite) signature $\{\bigcirc^i p \mid p \in \Sigma\}$.* ◀

▶ **Proposition 10.** *Any normal positive STL-program Π has a unique temporal stable model $\langle (D, \sigma), \mathbf{T} \rangle$ which coincides with its $\leq$-least LTL-model. We denote $LM(\Pi) = Facts(\mathbf{T})$.* ◀

4 Safe Variables and Domain Independence

In this section we consider the new definition of safe variables which does not refer to static predicates any more. As a result, we obtain a direct extrapolation of DLV-safety by just ignoring the temporal operators.

▶ **Definition 11.** A splitable temporal formula φ of type (1), (2) or (3) is said to be *safe* if, for any variable x occurring in φ, there exists an atomic formula p in B or B' such that x occurs in p. A formula $\forall x_1 \forall x_2 \ldots \forall x_n \psi$ is safe if the splitable temporal formula ψ is safe.

For instance, rules (8)-(12) are safe. A simple example of unsafe rule is the splitable temporal formula $\top \rightarrow P(x)$ where x does not occur in the positive body (in fact, the rule body is empty). Although an unsafe rule not always leads to lack of domain independence (see examples in [6]) it is frequently the case. We prove next that domain independence is, in fact, guaranteed for safe STL-programs.

▶ **Theorem 12.** *If φ is a safe sentence and $\langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle$ is a temporal equilibrium model of φ, then $\mathbf{T}|_C = \mathbf{T}$ and $T_i \subseteq At(\sigma(C), P)$ for any $i \geq 0$.*

Let (D,σ) be a domain and $D' \subseteq D$ a finite subset; the grounding over D' of a sentence φ, denoted by $\mathrm{Gr}_{D'}(\varphi)$, is defined recursively

$$
\begin{aligned}
\mathrm{Gr}_{D'}(p) &\stackrel{\text{def}}{=} p, \text{ where } p \text{ denotes any atomic formula} \\
\mathrm{Gr}_{D'}(\varphi_1 \odot \varphi_2) &\stackrel{\text{def}}{=} \mathrm{Gr}_{D'}(\varphi_1) \odot \mathrm{Gr}_{D'}(\varphi_2), \text{ with } \odot \text{ any binary operator in } \{\wedge, \vee, \rightarrow\} \\
\mathrm{Gr}_{D'}(\forall x \varphi(x)) &\stackrel{\text{def}}{=} \bigwedge_{d \in D'} \mathrm{Gr}_{D'} \varphi(d) \\
\mathrm{Gr}_{D'}(\exists x \varphi(x)) &\stackrel{\text{def}}{=} \bigvee_{d \in D'} \mathrm{Gr}_{D'} \varphi(d) \\
\mathrm{Gr}_{D'}(\bigcirc \varphi) &\stackrel{\text{def}}{=} \bigcirc \mathrm{Gr}_{D'}(\varphi) \\
\mathrm{Gr}_{D'}(\Box \varphi) &\stackrel{\text{def}}{=} \Box \mathrm{Gr}_{D'}(\varphi) \\
\mathrm{Gr}_{D'}(\Diamond \varphi) &\stackrel{\text{def}}{=} \Diamond \mathrm{Gr}_{D'}(\varphi)
\end{aligned}
$$

▶ **Proposition 13.** *Given any non-empty finite set D:*
$\langle (D,\sigma), \mathbf{H}, \mathbf{T}\rangle \models \varphi$ *iff* $\langle (D,\sigma), \mathbf{H}, \mathbf{T}\rangle \models \mathrm{Gr}_D(\varphi)$. ◀

▶ **Theorem 14** (Domain independence)**.** *Let φ be safe splitable temporal sentence. Suppose we expand the language $\mathcal{L}$ by considering a set of constants $C' \supseteq C$. A total* **QTHT***-model $\langle (D,\sigma), \mathbf{T}, \mathbf{T}\rangle$ is a temporal equilibrium model of $\mathrm{Gr}_{C'}(\varphi)$ if and only if it is a temporal equilibrium model of $\mathrm{Gr}_C(\varphi)$.*

5 Derivable ground facts

In this section we present a technique for grounding safe temporal programs based on the construction a positive normal ASP program with variables. The least model of this program can be obtained by the ASP grounder[6] DLV and it can be used afterwards to provide the variable substitutions to be performed on the STL-program. Besides, in some cases, this technique means a reduction of the number of generated ground rules with respect to the previous strategy that relied on static predicates.

[6] Or any other ASP grounder, such as gringo, respecting DLV definition of safe variables.

The method is based on the idea of *derivable* ground temporal facts for an STL-program Π. This set, call it Δ, will be an upper estimation of the credulous consequences of the program, that is, $CredFacts(\Pi) \subseteq \Delta$. Of course, the ideal situation would be that $\Delta = CredFacts(\Pi)$, but the set $CredFacts(\Pi)$ requires the temporal stable models of Π and these (apart from being infinite sequences) will not be available at grounding time. In the worst case, we could choose Δ to contain the whole set of possible temporal facts, but this would not provide relevant information to improve grounding. So, we will try to obtain some superset of $CredFacts(\Pi)$ as small as possible, or if preferred, to obtain the largest set of *non-derivable* facts we can find. Note that a non-derivable fact $\bigcirc^i p \notin \Delta$ satisfies that $\bigcirc^i p \notin CredFacts(\Pi)$ and so, by Proposition 5, $\Pi \cup \{\neg \bigcirc^i p\}$ is equivalent to Π, that is, both theories have the same set of temporal equilibrium models. This information can be used to simplify the ground program either by removing rules or literals.

We begin defining several transformations on STL-programs. For any temporal rule r, we define $r^\wedge$ as the set of rules:

- If r has the form (1) then $r^\wedge \stackrel{\text{def}}{=} \{B \rightarrow p \mid \text{atom } p \text{ occurs in } H\}$
- If r has the form (2) then $r^\wedge \stackrel{\text{def}}{=} \{B \wedge \bigcirc B' \rightarrow \bigcirc p \mid \text{atom } p \text{ occurs in } H'\}$
- If r has the form (3) then $r^\wedge \stackrel{\text{def}}{=} \{\Box(B \wedge \bigcirc B' \rightarrow \bigcirc p) \mid \text{atom } p \text{ occurs in } H'\}$

In other words, $r^\wedge$ results from removing all negative literals in r and, informally speaking, transforming disjunctions in the head into conjunctions, so that $r^\wedge$ will imply *all* the original disjuncts in the disjunctive head of r. It is interesting to note that for any rule r with an empty head ($\bot$) this definition implies $r^\wedge = \emptyset$. Program $\Pi^\wedge$ is defined as the union of $r^\wedge$ for all rules $r \in \Pi$. As an example, $\Pi_7^\wedge$ consists of the rules:

$$\begin{array}{llll} \top & \rightarrow & q & \qquad \Box(q \rightarrow \bigcirc q) \qquad \Box(r \rightarrow \bigcirc r) \\ q & \rightarrow & \bigcirc p & \qquad\qquad\quad\ \ \Box(r \rightarrow \bigcirc q) \end{array}$$

whereas $\Pi_1^\wedge$ would be the program:

$$\Box(\ Driveto(x,a) \rightarrow \bigcirc At(x,a)\ ) \tag{13}$$
$$\Box(\ At(x,a) \wedge Road(a,b) \rightarrow Driveto(x,b)\ ) \tag{14}$$
$$\Box(\ At(x,a) \wedge Road(a,b) \rightarrow NoDriveto(x,b)\ ) \tag{15}$$
$$\Box(\ At(x,a) \rightarrow \bigcirc At(x,a)\ ) \tag{16}$$
$$\Box(\ At(x,b) \wedge City(a) \wedge a \neq b \rightarrow NoAt(x,a)\ ) \tag{17}$$

If we look carefully at this example program, we are now moving each car x so that it will be at several cities at the same time (constraint (12) has been removed) and, at each step, it will additionally locate car x in all adjacent cities to the previous ones "visited" by x. In this way, if we conclude $\bigcirc^i At(x,a)$ from this program this is actually representing that car x *can reach* city a in i steps or less. In some sense, $\Pi^\wedge$ looks like a heuristic simplification[7] of the original problem obtained by removing some constraints (this is something common in the area of Planning in Artificial Intelligence).

Notice that, by definition, $\Pi^\wedge$ is always a positive normal STL-program and, by Proposition 10, it has a unique temporal stable model, $LM(\Pi^\wedge)$.

▶ **Proposition 15.** *For any STL-program* Π, $CredFacts(\Pi) \subseteq LM(\Pi^\wedge)$. ◀

[7] We could further simplify $\Pi^\wedge$ removing rules (15) and (17) by observing that their head predicates never occur in a positive body of Π_1. However, for the formal results in the paper, this is not essential, and would complicate the definitions.

Unfortunately, using $\Delta = LM(\Pi^\wedge)$ as set of derivable facts is unfeasible for practical purposes, since this set contains infinite temporal facts corresponding to an "infinite run" of the transition system described by $\Pi^\wedge$. Take for instance $\Pi_1^\wedge$ for the cars scenario. Imagine a roadmap with thousands of connected cities. $LM(\Pi^\wedge)$ can tell us that, for instance, car 1 cannot reach Berlin in less than 316 steps, so that $\bigcirc^{315} At(1, Berlin)$ is non-derivable, although $\bigcirc^{316} At(1, Berlin)$ is derivable. However, in order to exploit this information for grounding, we would be forced to expand the program up to some temporal distance, and we have no hint on where to stop. Note that, on the other hand, when we represent the transition system as usual in ASP, using a bounded integer variable for representing time, then this fine-grained optimization for grounding can be applied, because the temporal path *always has a finite length.*

As a result, we will adopt a compromise solution taking a superset of $LM(\Pi^\wedge)$ extracted from a new theory, Γ_Π. This theory will collapse all the temporal facts from situation 2 on, so that all the states T_i for $i \geq 2$ will be repeated. We define Γ_Π as the result of replacing each rule $\Box(B \wedge \bigcirc B' \rightarrow \bigcirc p)$ in $\Pi^\wedge$ by the formulas:

$$B \wedge \bigcirc B' \rightarrow \bigcirc p \tag{18}$$

$$\bigcirc B \wedge \bigcirc^2 B' \rightarrow \bigcirc^2 p \tag{19}$$

$$\bigcirc^2 B \wedge \bigcirc^2 B' \rightarrow \bigcirc^2 p \tag{20}$$

and adding the axiom schema:

$$\bigcirc^2 \Box(p \leftrightarrow \bigcirc p) \tag{21}$$

for any ground atom $p \in At(D, P)$ in the signature of Π. As we can see, (18) and (19) are the first two instances of the original rule $\Box(B \wedge \bigcirc B' \rightarrow \bigcirc p)$ corresponding to situations $i = 0$ and $i = 1$. Formula (20), however, differs from the instance we would get for $i = 2$ since, rather than having $\bigcirc^3 B'$ and $\bigcirc^3 p$, we use $\bigcirc^2 B'$ and $\bigcirc^2 p$ respectively. This can be done because axiom (21) is asserting that from situation 2 on all the states are repeated.

In the cars example, for instance, rule (13) in $\Pi_1^\wedge$ would be transformed in Γ_{Π_1} into the three rules:

$$Driveto(x, a) \rightarrow \bigcirc At(x, a) \qquad \bigcirc Driveto(x, a) \rightarrow \bigcirc^2 At(x, a)$$

$$\bigcirc^2 Driveto(x, a) \rightarrow \bigcirc^2 At(x, a)$$

It is not difficult to see that axiom (21) implies that checking that some $\mathcal{M}$ is a temporal equilibrium model of Γ_Π is equivalent to check that $\{\bigcirc^i p \mid p \in T_i\,,\ i = 0, 1, 2\}$ is a stable model of $\Gamma_\Pi \setminus \{(21)\}$ and fixing $T_i = T_2$ for $i \geq 3$. This allows us to exclusively focus on the predicate extents in T_0, T_1 and T_2, so we can see the $\Box$-free program $\Gamma_\Pi \setminus \{(21)\}$ as a positive normal ASP (i.e., non-temporal) program for the propositional signature $\{p, \bigcirc p, \bigcirc^2 p \mid p \in At(D, P)\}$ that can be directly fed to DLV, after some simple renaming conventions.

▶ **Theorem 16.** *Γ_Π has a least LTL-model, $LM(\Gamma_\Pi)$ which is a superset of $LM(\Pi^\wedge)$.*

In other words $CredFacts(\Pi) \subseteq LM(\Pi^\wedge) \subseteq LM(\Gamma_\Pi) = \Delta$, i.e., we can use $LM(\Gamma_\Pi)$ as set of derivable facts and simplify the ground program accordingly. Note that this simplification does not mean that we first ground everything and then remove rules and literals: we simply do not generate the irrelevant ground cases.

A slight adaptation[8] is further required for this method: as we get ground facts of the form $p, \bigcirc p$ and $\bigcirc^2 p$ we have to unfold the original STL-program rules to refer to atoms in

[8] In fact, this means that we have to extend the definition of splitable temporal rule to cope with $\bigcirc^2$ atoms. This is not essential, but has forced to reprogram the translation into LTL performed by `STeLP`.

the scope of $\bigcirc^2$. For instance, given (9) we would first unfold it into:

$$At(x,a) \wedge Road(a,b) \rightarrow Driveto(x,b) \vee NoDriveto(x,b) \quad (22)$$

$$\bigcirc At(x,a) \wedge \bigcirc Road(a,b) \rightarrow \bigcirc Driveto(x,b) \vee \bigcirc NoDriveto(x,b) \quad (23)$$

$$\Box(\ \bigcirc^2 At(x,a) \wedge \bigcirc^2 Road(a,b) \rightarrow \bigcirc^2 Driveto(x,b) \vee \bigcirc^2 NoDriveto(x,b)\ ) \quad (24)$$

and then check the possible extents for the positive bodies we get from the set of derivable facts $\Delta = LM(\Gamma_\Pi)$. For instance, for the last rule, we can make substitutions for x, a and b using the extents of $\bigcirc^2 At(x,a)$ and $\bigcirc^2 Road(a,b)$ we have in Δ. However, this still means making a join operation for both predicates. We can also use DLV for that purpose by just adding a rule that has as body, the positive body of the original temporal rule r, and as head, a new auxiliary predicate $Subst_r(x,a,b)$ referring to all variables in the rule. In the example, for rule (24) we would include in our DLV program:

$$\bigcirc^2 At(x,a) \wedge \bigcirc^2 Road(a,b) \rightarrow Subst_{(24)}(x,a,b)$$

In this way, each tuple of $Subst_r(x_1,\ldots,x_n)$ directly points out the variable substitution to be performed on the temporal rule.

6 Conclusions

We have improved the grounding method for temporal logic programs with variables in different ways. First, we provided a safety condition that directly corresponds to extrapolating the usual concept of safe variable in Answer Set Programming as required, for instance, by the input language of DLV [9]. In this way, any variable occurring in a rule is considered to be safe if it also occurs in the positive body of the rule, regardless the possible scope of temporal operators. An interesting topic for future study is trying to extend [3, 8, 6] to the temporal case, providing a general safety condition for arbitrary quantified temporal theories. Second, we have designed a method for grounding the temporal logic program that consists in constructing a non-temporal normal positive program with variables that is fed to solver DLV to directly obtain the set of variable substitutions to be performed for each rule. The proposed method allows reducing in many cases the number of ground temporal rules generated as a result. For instance, in the cars scenario from Figure 2 and the small instance case described in the paper (2 cars and 6 cities) we reduce the number of generated ground rules in the scope of '$\Box$' from 160 using the current STeLP grounding method to 62 with the technique introduced here. Due to the combinatorial nature of this decrease, we do not include figures for other instances of the example. The reader may easily imagine that the higher degree of cities interconnection, the smaller obtained reduction of rule instances, as this approaches to the worst case of n^2, where the n cities are all pairwise connected. On the other hand, the example is general enough to illustrate the proposed technique, as rules for temporal predicates usually limit the possible combinations of variable values we must consider.

A stand-alone prototype for proving examples like the one in the paper has been constructed, showing promising results. The immediate future work is incorporating the new grounding method inside STeLP and analysing its performance on benchmark scenarios. We will also study different improvements like, for instance, detecting rules with variables that are irrelevant for grounding.

References

1 F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Strongly equivalent temporal logic programs. In *JELIA 2008*, volume 5293 of LNCS, pages 8–20.

2 F. Aguado, P. Cabalar, G. Pérez, and C. Vidal. Loop formulas for splitable temporal logic programs. In James P. Delgrande and Wolfgang Faber, editors, *LPNMR'11*, volume 6645 of LNCS, pages 80–92. Springer, 2011.

3 A. Bria, W. Faber, and N. Leone. Normal form nested programs. In S. Hölldobler *et al*, editor, *Proc. of the 11th European Conference on Logics in Artificial Intelligence (JELIA'08)*, Lecture Notes in Artificial Intelligence, pages 76–88. Springer, 2008.

4 P. Cabalar. A normal form for linear temporal equilibrium logic. In *JELIA'10*, volume 6341 of LNCS, pages 64–76. Springer, 2010.

5 P. Cabalar and M. Diéguez. STELP - a tool for temporal answer set programming. In *LPNMR'11*, volume 6645 of LNCS, pages 370–375, 2011.

6 P. Cabalar, D. Pearce, and A. Valverde. A revised concept of safety for general answer set programs. In *LPNMR'09*, volume 5753 of LNCS, pages 58–70. Springer, 2009.

7 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming: Proc. of the Fifth International Conference and Symposium (Volume 2)*, pages 1070–1080. MIT Press, 1988.

8 J. Lee, V. Lifschitz, and R. Palla. Safe formulas in the general theory of stable models. preliminary report. In *ICLP'08*, volume 5366 of *LNCS*, pages 672–676. Springer, 2008.

9 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.

10 Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the eleventh international conference on Logic programming*, pages 23–37, Cambridge, MA, USA, 1994. MIT Press.

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

12 Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Maths and AI*, 53(1-4):251–287, 2008.

13 I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

14 D. Pearce. A new logical characterisation of stable models and answer sets. In *Non monotonic extensions of logic programming. Proc. NMELP'96. (LNAI 1216)*. 1996.

15 David Pearce. Equilibrium logic. *Annals of Maths and AI*, 47(1-2):3–41, 2006.

16 A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.