A New Approach to Temporal Logic Programming

Jose Gaintzarain [1]

Lenguajes y Sistemas Informáticos
The University of the Basque Country
Bilbao, Spain

Paqui Lucio[2]

Lenguajes y Sistemas Informáticos
The University of the Basque Country
San Sebastián, Spain

Abstract

In this paper, we propose a new approach to define temporal logic programming languages based on a temporal extension of resolution. We introduce the very expressive language TeDiLog that allows both eventualities and always formulas to occur in the head and also in the body of clauses. The operational semantics of TeDiLog is formulated on the basis of a resolution mechanism that dispenses with invariants, but produces new disjunctive clauses. As a consequence TeDiLog combines the temporal and the disjunctive paradigms in logic programming. In this contribution, we restrict our presentation to the propositional setting where the underlying logic PLTL is complete. Hence, the equivalence between operational and the logical semantics can be fully achieved.

Keywords: Temporal Logic Programming, Linear Temporal Logic, Invariant-free Temporal Resolution, Disjunctive Logic Programming, Refutation Procedure, Operational Semantics, Logical Semantics.

1 Introduction

Temporal logic programming (TLP) in a broad sense means programming in any language that is based in a temporal logic. TLP is an important tool for describing dynamic systems, therefore TLP is a central issue for many applications in computer science and artificial intelligence. The different approaches to TLP can be grouped into two categories depending on their view of a program execution as, either a construction of a model, or a refutation proof. The earliest languages TEMPURA and TOKIO, respectively proposed in [14] and [2], are based on interval temporal logic and belong to the former class. Their objective is to construct a Kripke model of a given formula and, to this end, the formula is interpreted similarly to an imperative program. Some other similar languages are proposed in [3,10,13].

[1] Email: jose.gaintzarain@ehu.es
[2] Email: paqui.lucio@ehu.es

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

The second category includes all languages that extends Horn clauses with temporal connectives. Depending on the underlying execution mechanism these temporal Horn languages can be grouped into two sub-families. On one hand, the execution of the languages proposed in [6,8,12] relies on a translation into the constraint logic programming (CLP) paradigm using special temporal theories. On the other hand, there are some proposals of TLP languages whose underlying execution mechanism extends classical resolution proof system for handling temporal connectives. The best known of that languages are Templog [1], Chronolog [15] and Gabbay's Temporal Prolog [9]. The implementation of both Templog and Chronolog are based in two different temporal extensions of SLD-resolution, called TSLD- and TiSLD-resolution, which respectively define their operational semantics. The only allowed temporal operator in Chronolog clauses is the next time operator ($\circ$). In Templog the always operator ($\Box$) is allowed in clause heads and the eventually operator ($\Diamond$) is allowed in clause bodys. However, Templog and Chronolog have the same expressive power and the same metalogical properties of existence of minimal model and fixpoint characterization. Roughly speaking Templog and Chronolog programs are expressible using $\circ$ as the only temporal operator. This restriction is so strong that allows to reduce any temporal program to a (possibly infinite) classical logic program. Gabbay's Temporal Prolog is a very expressive language that allows two kinds of eventually connectives (future and past), but does not allow $\Box$ in clause heads. This great expressiveness causes the lack of the minimal model property. A resolution-based computation procedure is outlined in [9] where it is also proved to be sound. As far as we know, the completeness property of Gabbay's Temporal Prolog has not been addressed.

In this paper, we propose a new approach to define TLP languages in the framework of temporal extension of resolution. We introduce a very expressive TLP language that allows both $\Box$ and $\Diamond$ in clause heads and bodys. In particular, our language is strictly more expressive than Templog, Chronolog and Gabbays's Temporal Prolog. We provide a system of resolution rules and a computation mechanism. As a first step, and for clarity, the presentation of the system is restricted to the propositional setting. Our refutation procedure is sound and complete with respect to the logical semantics of the logical consequences of the program. We cannot expect the minimal model property because of the same reasons of Gabbay's Temporal Prolog. Due to technical reasons related to our resolution system, we choose to combine the paradigms of temporal and disjunctive logic programming (DLP). Temporal disjunctive logic programming has previously been addressed in [11] where Chronolog is extended with DLP features. From the temporal point of view, Disjunctive Chronolog has the same limitations as Chronolog. We conjecture that our temporal disjunctive logic programming language naturally enjoys the minimal model property that is the metalogical roof in DLP. Our resolution system requires the expressive power of full temporal logic. That is, the resolution of a $\Diamond$-goal, necessarily generates subgoals involving the strictly more expressive operator "until". Hence, we directly formulate our language in terms of the temporal operators "until" and "release".

2 The Logic PLTL

A PLTL-formula is built using propositional variables (denoted by lowercase letters $p, q, \ldots$) from a set Prop, the classical connectives $\neg$ and $\wedge$, and the temporal connectives $\circ$ and $\mathcal{U}$. A lowercase Greek letter ($\varphi, \psi, \chi, \gamma, \ldots$) denotes a formula and an uppercase one

$(\Phi, \Delta, \Gamma, \Psi, \Omega, \Pi, \ldots)$ denotes a finite set of PLTL-formulas. As usual other connectives can be defined in terms of the previous ones: $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, $\diamond \varphi \equiv \neg\varphi \mathcal{U} \varphi$, $\Box\varphi \equiv \neg\diamond\neg\varphi$. Note that $\Box\varphi \equiv \neg\varphi \mathcal{R} \varphi$. PLTL-formulas of the form $\varphi \mathcal{U} \psi$ and $\diamond\varphi$ are called *eventualities*. In the sequel, *formula* means PLTL-formula.

We use two kind of superscripts on unary connectives namely $\otimes$. First, $\otimes^i$ with i varying on $\mathbb{N}$ represents the sequence formed by i $\otimes$, in particular empty for $i = 0$. Second, the especial case $\otimes^b$ for b varying in $\{0, 1\}$ which allows to represent a formula with or without a prefixed $\otimes$. Along the rest of the paper superscripts b and b' range in $\{0, 1\}$.

A PLTL-*structure* $\mathcal{M}$ is a pair $(S_\mathcal{M}, V_\mathcal{M})$ such that $S_\mathcal{M}$ is a denumerable sequence of states $s_0, s_1, s_2, \ldots$ and $V_\mathcal{M}$ is a map $V_\mathcal{M} : S_\mathcal{M} \to 2^{\mathsf{Prop}}$. Intuitively, $V_\mathcal{M}(s)$ specifies which atomic propositions are (necessarily) true in the state s. The formal semantics of PLTL-formulas is given by the truth of a formula φ in the state s_j of a PLTL-structure $\mathcal{M}$, which is denoted by $\langle \mathcal{M}, s_j \rangle \models \varphi$, which is inductively defined as follows:

- $\langle \mathcal{M}, s_j \rangle \models p$ iff $p \in V_\mathcal{M}(s_j)$ for $p \in \mathsf{Prop}$
- $\langle \mathcal{M}, s_j \rangle \models \neg\varphi$ iff $\langle \mathcal{M}, s_j \rangle \not\models \varphi$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \wedge \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ and $\langle \mathcal{M}, s_j \rangle \models \psi$
- $\langle \mathcal{M}, s_j \rangle \models \circ\varphi$ iff $\langle \mathcal{M}, s_{j+1} \rangle \models \varphi$
- $\langle \mathcal{M}, s_j \rangle \models \varphi \mathcal{U} \psi$ iff there exists $k \geq j$ such that $\langle \mathcal{M}, s_k \rangle \models \psi$ and for every $j \leq i < k$ it holds $\langle \mathcal{M}, s_i \rangle \models \varphi$.

This formal semantics is extended using the respective abbreviations to the defined connectives. The semantics is extended from formulas to sets of formulas in the usual way: $\langle \mathcal{M}, s_j \rangle \models \Phi$ iff $\langle \mathcal{M}, s_j \rangle \models \gamma$ for all $\gamma \in \Phi$. We say that $\mathcal{M}$ is a model of Φ, in symbols $\mathcal{M} \models \Phi$, iff $\langle \mathcal{M}, s_0 \rangle \models \Phi$. A satisfiable set of formulas has at least one model, otherwise it is unsatisfiable. Two sets of formulas Φ and Ψ are equisatisfiable whenever Φ is satisfiable iff Ψ is satisfiable. A formula χ is a *logical consequence* of a set of formulas Φ , denoted as $\Phi \models \chi$, iff for every PLTL-structure $\mathcal{M}$ and every $s_j \in S_\mathcal{M}$: if $\langle \mathcal{M}, s_j \rangle \models \Phi$ then $\langle \mathcal{M}, s_j \rangle \models \chi$.

3 The Logic Programming Language

Our notion of atom extends the classical notion of propositional atom with temporal atoms and (possibly empty) prefixed chains of the connective $\circ$. Using the usual BNF-notation:

$$L ::= C \mid \neg C \qquad T ::= L \mathcal{U} C \mid L \mathcal{R} C \mid \diamond C \mid \Box C \qquad A ::= \circ^i C \mid \circ^i T \quad (i \in \mathbb{N})$$

where C stands for classical atom, L for literal, T for temporal atom and A for atom. In the sequel, we say atom in the latter sense, otherwise we specify between propositional or temporal atom. Program clauses and query clauses are defined using atoms as usual

$$H ::= \bot \mid A \vee H \qquad B ::= \top \mid A \wedge B \qquad D ::= \Box^b(A \vee H \leftarrow B) \qquad Q ::= \Box^b(\bot \leftarrow B)$$

where $\bot$ represents the empty disjunction and $\top$ represents the empty conjunction. H stands for head, B for body, D for (disjunctive) program clause and Q for query clause.

A program is a set of program clauses and a goal is a set of query clauses. The intended meaning of a program Π is the conjunction of the program clauses in Π and the intended meaning of a goal Γ is the conjunction of the query clauses in Γ. A clause of the form $H \leftarrow B$ is called a *now-clause* and a clause of the form $\Box(H \leftarrow B)$ is called an *always-*

clause. Note that $\Box^b(\bot \leftarrow \top)$ represents the two possible syntactic forms of the empty clause, as now- or always-clause. We identify heads and bodies (finite disjunctions and conjunctions of atoms, respectively) with sets of atoms. Hence, we assume there is neither repetitions nor established order in the atoms of a head or a body.

In the next example we show a TeDiLog specification of a system. We would like to remark that many clauses in the below example contain eventualities in their heads. Hence, this clauses cannot be written in the programming languages [1,15] that we mention in Section 1, although they are allowed in the language of [9].

Example 3.1

1. $\Box(\circ\Diamond\, ack\_sm \leftarrow req\_dv)$
2. $\Box(\circ\Diamond\, eop\_dv \leftarrow ack\_sm)$
3. $\Box(\Diamond\, ctr\_sm \leftarrow \top)$
4. $\Box(\Diamond\, ack\_sm \leftarrow ctr\_sm)$
5. $\Box(com\_dv \leftarrow \Diamond\, eop\_dv)$
6. $\Box(com\_sm \leftarrow \Diamond\, ack\_sm)$
7. $\Box(\neg waiting\_dv\,\mathcal{U}\, req\_dv \vee \neg waiting\_dv\,\mathcal{U}\, ack\_sm \leftarrow eop\_dv)$
8. $\Box(waiting\_dv\,\mathcal{U}\, ack\_sm \leftarrow req\_dv)$
9. $\Box(working\_dv\,\mathcal{U}\, eop\_dv \leftarrow ack\_sm)$
10. $\Box(waiting\_sm\,\mathcal{U}\, eop\_dv \leftarrow ack\_sm)$
11. $\Box(\circ(\neg req\_dv\,\mathcal{U}\, eop\_dv) \leftarrow working\_dv)$
12. $\Box(\neg working\_dv\,\mathcal{U}\, ack\_sm \leftarrow eop\_dv)$
13. $\Box(\circ(\neg req\_dv\,\mathcal{U}\, eop\_dv) \leftarrow req\_dv)$

In this example a system where a device dv and a system manager sm interact with each other is (partially) specified by means of a set of program clauses. The device dv sends a request req_dv to sm to get permission to execute a process and goes into waiting-state until sm sends the acknowledgement ack_sm giving permission to execute the process. Then sm goes into waiting-state whereas dv goes into working-state until dv communicates the end of the process by means of eop_dv. Besides, the system manager innerly generates a control signal ctr_sm from time to time which is eventually followed by ack_sm provoking an answer eop_dv from the device dv. The interaction generated after the control signal ctr_sm corresponds to the fact that the system manager sm has to control regularly whether the device dv is correctly connected to the system. The device dv is considered to be in communicating-state (com_dv) while the arising of the eop_dv signal (now or in a future moment) is guaranteed. Similarly for com_sm whith respect to sm and ack_sm.

With this TeDiLog specification in hand, one could check if the system verifies some properties such as fairness, liveness, safety, mutual exclusion, etc. It suffices to express the property as a goal. For instance we would be interested in checking whether the device dv and the system manager sm will always keep communicating with each other. This could be accomplished by checking whether dv is always in communicating-state (com_dv) and checking whether sm is always in communicating-state (com_sm). The corresponding goal would be $\{\bot \leftarrow \Box com\_dv \wedge \Box com\_sm\}$. Note that this goal cannot be expressed neither in the languages of [1,15] nor in the language [9]. In Section 4 we give a computation example using a simplification of this TeDiLog specification. ∎

4 Procedural Semantics

In this section, we present the refutation procedure underlying TeDiLog. First we introduce some necessary technical notation. Then we provide the rules of our temporal resolution system and finally the proof method is showed.

Given a set of clauses Δ, we define $\mathsf{alw}(\Delta) = \{\Box N \mid \Box N \in \Delta\}$ and $\mathsf{now}(\Delta) =$

$\Delta \setminus \mathsf{alw}(\Delta)$. Given a now-clause $N = A_1 \vee \ldots \vee A_n \leftarrow A'_1 \wedge \ldots \wedge A'_m$, with $n \geq 0$ and $m \geq 0$, we abbreviate by $\circ N$ the now-clause $\circ A_1 \vee \ldots \vee \circ A_n \leftarrow \circ A'_1 \wedge \ldots \wedge \circ A'_m$.

The operator unnext applies to a set of clauses Ψ for obtaining the clauses that should be satisfied at any state which is next to any state that satisfies Ψ, that is, $\mathsf{unnext}(\Psi) = \mathsf{alw}(\Psi) \cup \{N \mid \Box^b(\circ N) \in \Psi\}$. Note that unnext implicitly uses the equivalence between $\Box N$ and $\{N, \circ\Box N\}$. Given a non-empty set of now-clauses $\Delta = \{N_1, \ldots, N_r\}$, the set $\Delta^\neg$ is the set of clauses obtained by applying the distribution laws to the formula $\neg N_1 \vee \ldots \vee \neg N_r$. For augmenting a set of now-clauses, i.e., for adding an atom to the body of each clause and converting the new now-clauses in always-clauses, we use the function $\mathsf{aug}(a, \Delta) = \{\Box(H \leftarrow B \wedge a) \mid H \leftarrow B \in \Delta\}$. The following function def associates a set of clauses to a fresh variable a w.r.t. a set of clauses Δ and a set of literals S. In some sense, the clauses in $\mathsf{def}(a, S, \Delta)$ give the meaning of a in a context given by Δ and S.

$$\mathsf{def}(a, S, \Delta) = \begin{cases} \{\Box(\bot \leftarrow a)\} & \text{if } \Delta = \emptyset \\ \mathsf{aug}(a, \Delta^\neg) & \text{if } \Delta \neq \emptyset \text{ and } S = \emptyset \\ \{\Box(p \leftarrow a)\} \cup \mathsf{aug}(a, \Delta^\neg) & \text{if } \Delta \neq \emptyset \text{ and } S = \{p\} \\ \{\Box(\bot \leftarrow p \wedge a)\} \cup \mathsf{aug}(a, \Delta^\neg) & \text{if } \Delta \neq \emptyset \text{ and } S = \{\neg p\} \end{cases}$$

Now, we give the rule system underlying the operational semantics of TeDiLog. Besides a classical-like resolution rule (Res), our system includes a subsumption rule (Sbm) and also the temporal rules for decomposing temporal atoms. The so called context-free rules are based directly on the inductive definition of the corresponding connective. The so called context-dependent rules are a bit peculiar in the sense that they apply to a set of clauses and the conclusion is a set constructed using also the clauses that are not in the premise set.

This section is split into four subsections. The first subsection is devoted to the rules (Res) and (Sbm), in the second subsection the context-free rules are introduced. The idea underlying the context-dependent rules is explained in the third subsection. In the fourth subsection the details about the refutation procedure are dealt with.

4.1 Resolution and Subsumption Rules

$$(Res)\ \frac{\Box^b(A \vee H \leftarrow B) \qquad \Box^{b'}(H' \leftarrow A \wedge B')}{\Box^{b*b'}(H \vee H' \leftarrow B \wedge B')}$$

(Sbm) $\Box^b(H \leftarrow B)$ is subsumed by $\Box^b(H' \leftarrow B')$ if $H' \subsetneq H$ and $B' \subseteq B$ or $H' \subseteq H$ and $B' \subsetneq B$

Fig. 1. The Resolution Rule (Res) and the Subsumption rule (Sbm)

The resolution rule (Res) in Figure 1 applies to two clauses (the premises) that verify that one of the atoms in the head of the first clause is in the body of the second clause, and a new clause is obtained (the resolvent). The premises remain in and the resolvent is added to the target set of clauses. The rule (Res) constitutes a very natural generalization of classical resolution to the case of TeDiLog clauses.

Similarly, the subsumption rule (Sbm) in Figure 1 generalizes classical subsumption to TeDiLog clauses. Note that the same superscript b occurs in both clauses. Unlike classical logic programming, we cannot ensure refutational completeness for our resolution procedure in absence of (Sbm).

4.2 *Context-Free Temporal Rules*

$$(\mathcal{U}\,L_1)\ \frac{\Box^b((p_1\,\mathcal{U}\,p_2)\vee H\leftarrow B)}{\{\Box^b(p_2\vee p_1\vee H\leftarrow B),\ \Box^b(p_2\vee\circ(p_1\,\mathcal{U}\,p_2)\vee H\leftarrow B)\}}$$

$$(\mathcal{U}\,L_2)\ \frac{\Box^b((\neg p_1\,\mathcal{U}\,p_2)\vee H\leftarrow B)}{\{\Box^b(p_2\vee p_1\vee H\leftarrow B),\ \Box^b(p_2\vee\circ(p_1\,\mathcal{U}\,p_2)\vee H\leftarrow B)\}}$$

$$(\mathcal{U}\,R_1)\ \frac{\Box^b(H\leftarrow(p_1\,\mathcal{U}\,p_2)\wedge B)}{\{\Box^b(H\leftarrow p_2\wedge B),\ \Box^b(H\leftarrow p_1\wedge\circ(p_1\,\mathcal{U}\,p_2)\wedge B)\}}$$

$$(\mathcal{U}\,R_2)\ \frac{\Box^b(H\leftarrow(\neg p_1\,\mathcal{U}\,p_2)\wedge B)}{\{\Box^b(H\leftarrow p_2\wedge B),\ \Box^b(p_1\vee H\leftarrow\circ(\neg p_1\,\mathcal{U}\,p_2)\wedge B)\}}$$

Fig. 2. The Context-Free Rules $(\mathcal{U}\,L_1)$, $(\mathcal{U}\,L_2)$, $(\mathcal{U}\,R_1)$ and $(\mathcal{U}\,R_2)$

$$(\mathcal{R}\,L_1)\ \frac{\Box^b((p_1\,\mathcal{R}\,p_2)\vee H\leftarrow B)}{\{\Box^b(p_2\vee H\leftarrow B),\ \Box^b(p_1\vee\circ(p_1\,\mathcal{R}\,p_2)\vee H\leftarrow B)\}}$$

$$(\mathcal{R}\,L_2)\ \frac{\Box^b((\neg p_1\,\mathcal{R}\,p_2)\vee H\leftarrow B)}{\{\Box^b(p_2\vee H\leftarrow B),\ \Box^b(\circ(p_1\,\mathcal{R}\,p_2)\vee H\leftarrow p_1\wedge B)\}}$$

$$(\mathcal{R}\,R_1)\ \frac{\Box^b(H\leftarrow(p_1\,\mathcal{R}\,p_2)\wedge B)}{\{\Box^b(H\leftarrow p_2\wedge p_1\wedge B),\ \Box^b(H\leftarrow p_2\wedge\circ(p_1\,\mathcal{R}\,p_2)\wedge B)\}}$$

$$(\mathcal{R}\,R_2)\ \frac{\Box^b(H\leftarrow(\neg p_1\,\mathcal{R}\,p_2)\wedge B)}{\{\Box^b(p_1\vee H\leftarrow p_2\wedge B),\ \Box^b(H\leftarrow p_2\wedge\circ(\neg p_1\,\mathcal{R}\,p_2)\wedge B)\}}$$

Fig. 3. The Context-Free Rules $(\mathcal{R}\,L_1)$, $(\mathcal{R}\,L_2)$, $(\mathcal{R}\,R_1)$ and $(\mathcal{R}\,R_2)$

The context-free rules $(\mathcal{U}\,L_1)$, $(\mathcal{U}\,L_2)$, $(\mathcal{U}\,R_1)$ and $(\mathcal{U}\,R_2)$ of Figure 2 replace a clause of the form $\Box^b((L\,\mathcal{U}\,p)\vee H\leftarrow B)$ or a clause of the form $\Box^b(H\leftarrow(L\,\mathcal{U}\,p)\wedge B)$ by a logically equivalent set of clauses using the well known inductive definition $L\,\mathcal{U}\,p\equiv p\vee(L\wedge\circ(L\,\mathcal{U}\,p))$ of the connective $\mathcal{U}$. Likewise, the rules $(\mathcal{R}\,L_1)$, $(\mathcal{R}\,L_2)$, $(\mathcal{R}\,R_1)$ and $(\mathcal{R}\,R_2)$ of Figure 3 use the inductive definition $L\,\mathcal{R}\,p\equiv p\wedge(L\vee\circ(L\,\mathcal{R}\,p))$ of the connective $\mathcal{R}$. A simple distribution gives the equivalent set of clauses that appears in the conclusion of each rule.

4.3 *Context-Dependent Temporal Rules*

The rules $(\mathcal{U}\,L_3)$, $(\mathcal{U}\,L_4)$, $(\mathcal{R}\,R_3)$ and $(\mathcal{R}\,R_4)$ consider a more complex inductive definition of the corresponding connective taking into account the *context* of the involved set of clauses. For instance, in the case of the rule $(\mathcal{U}\,L_3)$ the underlying idea is that a set of clauses $\Omega\cup\{\Box^{b_i}((p_1\,\mathcal{U}\,p_2)\vee H_i\leftarrow B_i)\mid 1\leq i\leq n\}$ (where $n\geq 1$) is satisfiable if and only if the following set of clauses is satisfiable

$$\Omega\cup\{p_2\vee p_1\vee H_i\leftarrow B_i,\ p_2\vee\circ(a\,\mathcal{U}\,p_2)\vee H_i\leftarrow B_i\ \mid\ 1\leq i\leq n\}\cup\mathsf{def}(a,\{p_1\},\Delta)$$
$$\cup\{\Box(\circ(p_1\,\mathcal{U}\,p_2)\vee\circ H_i\leftarrow\circ B_i)\ \mid\ b_i=1\text{ and }1\leq i\leq n\}$$

where $\Delta=\mathsf{now}(\Omega)$ and the fresh propositional variable a is always equivalent to the formula $p1\wedge\bigvee_{\zeta\in\Delta}\neg\zeta$. Intuitively, if $p_1\,\mathcal{U}\,p_2$ is true at the first state s_0 of a model but the model does not satisfy p_2 at s_0, then p_2 should be true in a later state, namely s_1, and the states in between s_0 and s_1 must satisfy p_1 but not the context Ω. When the number of possible contexts is finite, this property prevents from postponing indefinitely

$$(\mathcal{U}\,L_3)\quad \frac{\Omega, \{\Box^{\,b_i}((p_1\,\mathcal{U}\,p_2)\vee H_i \leftarrow B_i) \mid 1\leq i\leq n\}}{\{p_2\vee p_1\vee H_i\leftarrow B_i,\ p_2\vee \circ(a\,\mathcal{U}\,p_2)\vee H_i\leftarrow B_i \ \mid\ 1\leq i\leq n\}\ \cup\ \mathsf{def}(a,\{p_1\},\Delta)\ \cup\ \{\Box\,(\circ(p_1\,\mathcal{U}\,p_2)\vee\circ H_i\leftarrow\circ B_i)\ \mid\ b_i=1 \text{ and } 1\leq i\leq n\}}$$

$$(\mathcal{U}\,L_4)\quad \frac{\Omega, \{\Box^{\,b_i}((\neg p_1\,\mathcal{U}\,p_2)\vee H_i \leftarrow B_i) \mid 1\leq i\leq n\}}{\{p_2\vee H_i\leftarrow p_1\wedge B_i,\ p_2\vee \circ(a\,\mathcal{U}\,p_2)\vee H_i\leftarrow B_i \ \mid\ 1\leq i\leq n\}\ \cup\ \mathsf{def}(a,\{\neg p_1\},\Delta)\ \cup\ \{\Box\,(\circ(\neg p_1\,\mathcal{U}\,p_2)\vee\circ H_i\leftarrow\circ B_i)\ \mid\ b_i=1 \text{ and } 1\leq i\leq n\}}$$

$$(\mathcal{R}\,R_3)\quad \frac{\Omega, \{\Box^{\,b_i}(H_i \leftarrow (p_1\,\mathcal{R}\,p_2)\wedge B_i) \mid 1\leq i\leq n\}}{\{H_i\leftarrow p_2\wedge p_1\wedge B_i,\ H_i\leftarrow p_2\wedge \circ(\neg a\,\mathcal{R}\,p_2)\wedge B_i \mid 1\leq i\leq n\}\ \cup\ \mathsf{def}(a,\{\neg p_1\},\Delta)\ \cup\ \{\Box\,(\circ H_i\leftarrow\circ(p_1\,\mathcal{R}\,p_2)\wedge\circ B_i) \mid b_i=1 \text{ and } 1\leq i\leq n\}}$$

$$(\mathcal{R}\,R_4)\quad \frac{\Omega, \{\Box^{\,b_i}(H_i \leftarrow (\neg p_1\,\mathcal{R}\,p_2)\wedge B_i) \mid 1\leq i\leq n\}}{\{p_1\vee H_i\leftarrow p_2\wedge B_i,\ H_i\leftarrow p_2\wedge \circ(\neg a\,\mathcal{R}\,p_2)\wedge B_i \mid 1\leq i\leq n\}\ \cup\ \mathsf{def}(a,\{p_1\},\Delta)\ \cup\ \{\Box\,(\circ H_i\leftarrow\circ(p_1\,\mathcal{R}\,p_2)\wedge\circ B_i) \mid b_i=1 \text{ and } 1\leq i\leq n\}}$$

where $n\geq 1$, $\Delta=\mathsf{now}(\Omega)$ and $a\in\mathsf{Prop}$ is fresh.

Fig. 4. The Rules $(\mathcal{U}\,L_3)$, $(\mathcal{U}\,L_4)$, $(\mathcal{R}\,R_3)$ and $(\mathcal{R}\,R_4)$

the satisfaction of $p_1\,\mathcal{U}\,p_2$ and makes possible to get a complete resolution method that does not require invariant generation at all.

Besides, the always-clauses in Ω can be excluded from the context (preserving equivalence) because of the easy equivalence between the set $\{\varphi, \Box\psi, \circ((\gamma\wedge(\neg\varphi\vee\neg\Box\psi))\,\mathcal{U}\,\delta)\}$ and the set $\{\varphi, \Box\psi, \circ((\gamma\wedge\neg\varphi)\,\mathcal{U}\,\delta)\}$. Hence, the context Δ is given by $\mathsf{now}(\Omega)$.

With respect to the clauses that state that a is always equivalent to the formula $p1\wedge\bigvee_{\zeta\in\Delta}\neg\zeta$, since the left-to-right implication is enough for equisatisfiability, we do not add the clauses for the reverse implication in the rule.

The idea behind the other context-dependent rules is the same as in $(\mathcal{U}\,L_1)$. In the case of the rule $(\mathcal{U}\,L_2)$ since $\neg p_1$ is not an atom, $\neg p_1$ cannot remain on the left-hand side of the new clauses and therefore p_1 appears on the right-hand side. The rules $(\mathcal{R}\,R_3)$ and $(\mathcal{R}\,R_4)$ are better understood considering that having $p_1\,\mathcal{R}\,p_2$ and $\neg p_1\,\mathcal{R}\,p_2$ on the right-hand side of a clause is like having (respectively) $\neg p_1\,\mathcal{U}\,\neg p_2$ and $p_1\,\mathcal{U}\,\neg p_2$ on the left-hand side.

4.4 Refutation Procedure

A *derivation* for a program Π and a goal Γ is a sequence of pairs $\mathcal{D}=(\Pi_0,\Gamma_0)\mapsto(\Pi_1,\Gamma_1)\mapsto\ldots\mapsto(\Pi_i,\Gamma_i)\mapsto\ldots$ where $(\Pi_0,\Gamma_0)=(\Pi,\Gamma)$ and each (Π_{i+1},Γ_{i+1}) is obtained from (Π_i,Γ_i) by applying one of the previously presented rules. If the last pair of a derivation $\mathcal{D}$ contains the empty query clause $\Box^b(\bot\leftarrow\top)$, then $\mathcal{D}$ is called a *refutation*. The clause $\Box^b(\bot\leftarrow\top)$ can only appear in the last pair of a derivation.

The resolution procedure consists in repeating Algorithm 1 starting from a pair (Π,Γ) where the empty clause does not appear in the goal

Algorithm 1 *Input: a program and a goal. Output: a derivation*
Step 1 Fix fairly a distinguished temporal atom
Step 2 Apply the context-dependent rule that corresponds to the distinguished atom
Step 3 Apply the context-free rules
Step 4 Saturate the set of program clauses using the resolution rule
Step 5 Apply linear resolution between clauses in the program and in the goal
Step 6 Apply the Subsumption rule
Step 7 Apply the unnext *operator*

For a temporal atom to be distinguishable it must appear as atom in at least one program clause or query clause and must accept the application of a context-dependent rule. In each iteration it is mandatory to distinguish a unique temporal atom if distinguishable atoms are available. At the beginning of a new iteration, we say that a temporal atom T' is the direct descendant of T if T was the distinguished atom in the previous iteration and $\circ T'$ is the fresh atom obtained after the application of the corresponding context-dependent rule to the set of all the clauses that contained T as an atom that accepted the application of such a rule. If T' is distinguishable in the present iteration, then T' must be the distinguished one. If T' is not distinguishable, one of the distinguishable temporal atoms is distinguished fairly. If following the mentioned procedure a temporal atom T'' has been distinguished in the present pair (Π_j, Γ_j), the corresponding context-dependent rule is applied to the set Ψ of all the clauses in $\Pi_j \cup \Gamma_j$ that contain T'' as an atom that accepts such a rule and the clauses in Ψ are replaced with the clauses in the conclusion set, obtaining a new pair $(\Pi_{j+1}, \Gamma_{j+1})$. Then the context-free rules are applied to the remaining temporal atoms replacing each affected clause with the corresponding new clauses. When a pair where no temporal rule can be applied is obtained, the program set is saturated by means of the resolution rule, adding all the program clauses that can be obtained applying the rule (Res) to pairs of program clauses. When the set of program clauses has been saturated, the linear resolution step begins. At this step the rule (Res) is applied while possible but in each application one of the premises must be a program clause and the other one must be a query clause. Note that in the linear resolution step an application of (Res) can give as conclusion either a program clause or a query clause and that clause must be added either to the program or to the goal. After the linear resolution step, the rule (Sbm) must be applied while possible. And finally, the application of the unnext operator yields a new pair and another iteration must begin. It is worthwhile to note that the the unnext operator is applied when no other rule can be applied. A derivation ends if during the linear resolution step the empty query clause $\Box^b(\bot \leftarrow \top)$ is obtained.

Now, let us give a hint about how TeDiLog works by means of a simplified version of the example introduced in Section 3.

Example 4.1 Consider the goal $\Gamma_0 = \{\bot \leftarrow \Box com\_dv\}$ w.r.t. the program

$$\Pi_0 = \{C_1 = \Box(\circ\Diamond eop\_dv \leftarrow ack\_sm), \quad C_2 = \Box(\Diamond ctr\_sm \leftarrow \top),$$
$$C_3 = \Box(\Diamond ack\_sm \leftarrow ctr\_sm), \quad C_4 = \Box(com\_dv \leftarrow \Diamond eop\_dv)\}$$

If we distinguish $\Box com\_dv$ in Γ_0, since $\Box p$ abbreviates $\neg p \,\mathcal{R}\, p$, then we apply the rule $(\mathcal{R}\, R_4)$ with empty context. Hence, Γ_0 is replaced with the set of two clauses

$$\Gamma_1 = \{\bot \leftarrow com\_dv \wedge \circ(\neg x \,\mathcal{R}\, com\_dv), \;\; \Box(\bot \leftarrow x)\} \text{ where } x \text{ is fresh.}$$

Now, in the clauses obtained by saturation there is only one now-clause: $\circ\Diamond eop\_dv \vee \circ\Diamond ack\_sm \vee \circ\Diamond ctr\_sm \leftarrow \top$. Therefore, after the first application of the unnext operator, we obtain the clause $C_5 = \Diamond eop\_dv \vee \Diamond ack\_sm \vee \Diamond ctr\_sm \leftarrow \top$ and the goal

$$\Gamma_2 = \{\bot \leftarrow \Diamond eop\_dv \wedge \neg x \,\mathcal{R}\, com\_dv, \;\; \Box(\bot \leftarrow x)\}$$

where $\neg x \,\mathcal{R}\, com\_dv$ becomes distinguished. Hence, by application of the rule $(\mathcal{R}\, R_4)$ to the first query in Γ_2 with context $\{C_5\}$ and successive resolution steps, the goal becomes $\Gamma_3 = \{\leftarrow ack\_sm, \;\; \Box(\bot \leftarrow x)\}$. Then, $\Diamond ack\_sm$ in C_3 is distinguished, to yield the goal $\Gamma_4 = \{\leftarrow ctr\_sm, \;\; \Box(\bot \leftarrow x)\}$. Finally, $\Diamond ctr\_sm$ in C_2 must be distinguished. Then the empty query is achieved. ∎

5 Equivalence between Operational and Logical Semantics

In this section we sketch the idea behind the equivalence of the operational and logical semantics for our system. This equivalence means that the resolution procedure using Algorithm 1 obtains a refutation for (Π, Γ) iff $\Pi \cup \Gamma$ is unsatisfiable. This equivalence is divided in soundness and completeness.

Soundness means that whenever a refutation exists for a pair (Π, Γ), then $\Pi \cup \Gamma$ is unsatisfiable. The soundness of our system can be guaranteed rule by rule. A rule is sound whenever it preserves the satisfiability from the premises to the conclusion. The rules presented in Sections 4.1, 4.2 and 4.3 are sound. Moreover, the initial and the target sets of every application of these rules are equisatisfiable. Besides, the operator unnext preserves satisfiability. Note that the equisatisfiability, in general, of initial and target sets of unnext cannot be ensured. By satisfiability preservation of the TeDiLog rules and the unnext operator, we can prove that whenever $\Box^b(\bot \leftarrow \top)$ occurs in a derivation, then the first pair of that derivation should be unsatisfiable. Hence, the introduced system is sound.

Completeness means that whenever a set of clauses $\Pi \cup \Gamma$ is unsatisfiable, a refutation for (Π, Γ) can be constructed. Under the assumption that the resolution algorithm uses a fair strategy for distinguishing atoms, the resolution system underlying the proof procedure of TeDiLog is complete.The aim of the algorithm presented in the previous Section 4.4 is to obtain a pair that contains the clause $\Box^b(\bot \leftarrow \top)$, however if the initial set of clauses $\Pi \cup \Gamma$ is satisfiable, then this never occurs and an infinite derivation is built. In order to see this fact let us consider the set $\mathsf{atoms}(\Pi, \Gamma)$ as the set that contains the atoms that could appear in the clauses obtained from (Π, Γ) by means of the unnext operator and all the TeDiLog rules with the exception of the context-dependent rules. The closure $\mathsf{closure}(\Pi, \Gamma)$ of a pair (Π, Γ) is defined as the (minimal) set that contains all the clauses that could be generated from atoms in $\mathsf{atoms}(\Pi, \Gamma)$. The number of clauses in $\mathsf{closure}(\Pi, \Gamma)$ is $2^{O(n)}$, where n is the number of atoms in $\mathsf{atoms}(\Pi, \Gamma)$.

Given a pair of sets of clauses (Π, Γ) as input, only the context-dependent rules introduce new variables and due to the way that the unnext operator and the context-dependent rules operate, the atoms that contain those fresh variables are always part of now-clauses. Hence a distinguished atom that contains a new variable and all its descendants are in now-clauses. Since following the algorithm the context-dependent rules must be applied to all the clauses that contain the distinguished atom, the context is always a set of clauses in $\mathsf{closure}(\Pi, \Gamma)$ and consequently the number of different contexts is finite. Moreover, the sequence of distinguished atoms generated from an atom (the sequence of descendants) is always finite because otherwise, since the closure is finite there would be necessarily two contexts repeated and since the distinguished atoms state that no previous context can be repeated, it is easy to see that clauses that do not contain the distinguished atom and that at the same time subsume all the clauses with distinguished atoms are produced eventually. So the sequence of descendants would finish contradicting its infiniteness. When the sequence of descendants of a distinguished temporal atom finishes because after an application of the unnext operator there is no a direct descendant of the previous one, that means that the satisfaction of that eventuality has succeeded or is not needed. If an infinite derivation is obtained there exists a model. Therefore, whenever there is no any refutation for (Π, Γ), then the Algorithm 1 in Section 4.4 produces an infinite derivation and a model for $\Pi \cup \Gamma$ can be constructed from that infinite derivation.

6 Future Work

In [4,11,15] the minimal model and the fixpoint characterization of the declarative semantics is a straightforward adaptation of the classical (disjunctive) case since in all of them the satisfaction of a program is reduced to the satisfaction of a classical logic program and the semantics is based on interpretations that contain only the so called temporally ground atoms (atoms that only contain $\circ$ as temporal operator). But we are not able to do such a reduction due to the fact that the $\mathcal{U}$ and the $\mathcal{R}$ operator can appear in heads and bodies. Our aim is to define the declarative semantics of TeDiLog programs using interpretations that can contain temporally non-ground atoms, i.e., atoms of the form $L\,\mathcal{U}\,p$ and $L\,\mathcal{R}\,p$. Our intention is to use the s-semantics approach as introduced in [5,7].

We plan to study TeDiLog in the first-order setting. Unlike PLTL, first order temporal logic is incomplete. In spite of the incompleteness, the equivalence between procedural and logical semantics could be preserved likewise in Templog. Besides we are also interested in the extension of the model theoretic and fixpoint characterization to the first-order setting.

References

[1] M. Abadi and Z. Manna. Temporal logic programming. *J. Symb. Comput.*, 8(3):277–295, 1989.

[2] T. Aoyagi, M. Fujita, and T. Moto-Oka. Temporal logic programming language tokio - programming in tokio. In E. Wada, editor, *Logic Programming '85, Proceedings of the 4th Conference, Tokyo, Japan, July 1-3, 1985*, volume 221 of *Lecture Notes in Computer Science*, pages 128–137. Springer, 1986.

[3] H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. Metatem: A framework for programming in temporal logic. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands, May 29 - June 2, 1989, Proceedings*, volume 430 of *Lecture Notes in Computer Science*, pages 94–129. Springer, 1990.

[4] M. Baudinet. Temporal logic programming is complete and expressive. In *POPL*, pages 267–280, 1989.

[5] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *J. Log. Program.*, 19/20:149–197, 1994.

[6] C. Brzoska. Programming in metric temporal logic. *Theor. Comput. Sci.*, 202(1-2):55–125, 1998.

[7] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli. Declarative modeling of the operational behavior of logic languages. *Theor. Comput. Sci.*, 69(3):289–318, 1989.

[8] T. W. Frühwirth. Annotated constraint logic programming applied to temporal reasoning. In M. V. Hermenegildo and J. Penjam, editors, *Programming Language Implementation and Logic Programming, 6th International Symposium, PLILP'94, Madrid, Spain, September 14-16, 1994, Proceedings*, volume 844 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 1994.

[9] D. M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics And Their Application*, pages 197–237. Academic Press, 1987.

[10] D. M. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer, 1989.

[11] M. Gergatsoulis, P. Rondogiannis, and T. Panayiotopoulos. Temporal disjunctive logic programming. *New Generation Comput.*, 19(1):87–102, 2000.

[12] T. Hrycej. Temporal prolog. In *8th European Conference on Artificial Intelligence, ECAI 88, Munich, Germany, August 1-5, 1988, Proceedings.*, pages 296–301. Pitmann Publishing, London, 1988.

[13] S. Merz. Efficiently executable temporal logic programs. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics, IJCAI '93, Workshop, Chambéry, France, August 28, 1993, Proceedings*, volume 897 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 1995.

[14] B. C. Moszkowski. Executing temporal logic programs. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA, July 9-11, 1984*, volume 197 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 1985.

[15] W. W. Wadge. Tense logic programming: a respectable alternative. In *Proc. of the 1988 International Symposium on Lucid and Intensional Programming, Sidney, B.C., Canada, Apr. 7-8*, pages 26–32, 1988.