Logical Foundations for Resolution-based Temporal Logic Programming Languages

J. Gaintzarain¹ and P. Lucio²

¹ The University of the Basque Country, 48012-Bilbao, Spain.
² The University of the Basque Country, 20080-San Sebastián, Spain.

Abstract. In this paper, we present a temporal logic programming language called TeDiLog that is an extension of the Horn clause language with temporal connectives. TeDiLog also allows disjunctions in clause heads, thus it is a combination of the temporal and disjunctive paradigms in Logic Programming. We formally define operational and declarative semantics for TeDiLog and prove their equivalence. The operational semantics of TeDiLog is based on resolution. Since TeDiLog is a sublanguage of the well-known Propositional Linear-time Temporal Logic (PLTL), the logical semantics of TeDiLog is supported by PLTL. TeDiLog allows both eventualities and always-formulas to occur in clause heads and also in clause bodies. To the best of our knowledge, TeDiLog is the first proposal of a pure sublanguage of PLTL that achieves this high degree of expressiveness in the field of resolution-based temporal logic programming. The model-theoretic semantics of a TeDiLog program is characterized by a collection of minimal models. TeDiLog is intended to be the propositional core language that captures the essence of our proposal with respect to temporal expressiveness. We hope that new temporal logic programming languages can be built on the top of TeDiLog—notably first-order languages—taking advantage of its logical foundations.

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

1 Introduction

Temporal logic is widely used in the specification, refinement, development and verification of software and hardware systems. Indeed, temporal logic constitutes the foundation of many formal methods and techniques whose central purpose is to improve the reliability of computer systems, in particular to verify their correctness property. Model checking and deductive reasoning are two major techniques for system validation. Model checking focuses on the problem of deciding whether a concrete model of a system satisfies a logical formula or not. In contrast to model checking, the deductive approach uses a logical formula to describe all possible executions of a system.

This work has been partially supported by Spanish Project TIN2007-66523 and by Basque Project LoRea GIU07/35.
and then attempts to prove the required property from this logical formula. A promising approach to automation is through use of restricted temporal languages that allow more efficient algorithms for the execution of logic specifications. A good trade-off between (efficient) implementability and expressiveness is highly desirable in this area. Temporal logic programming (TLP), in a broad sense, means programming in any language based on temporal logic. TLP languages enable to write executable specifications of systems in an underlying temporal logic. Such executable specifications are very interesting for prototyping, debugging and improving systems, before their final use. Additionally, the TLP aim is not only to obtain a yes/no answer (for satisfiability or deductive problems), but also to take into account how the problem has been solved. The information on how a query is deduced from a program is given in terms of substitutions for the variables of the query. These substitutions, called answers, constitute the output of a program regarding a query. TLP languages provide a single framework in which dynamic systems can be specified, developed, validated and verified. There are a huge variety of proposals for TLP languages that relies on features such as the underlying temporal logic (different variants of temporal logic, interval logic, etc), the view of program execution (as the construction of either a model or a refutation proof), the execution mechanism (translation to constraint logic programming, temporal resolution, etc), and even its capacity to express parallelism, concurrency, and so on. The reader is referred to \cite{9,12} as surveys of TLP languages.

The well-known Propositional Linear Temporal Logic (PLTL) is the most fundamental and basic logic that constitutes the core of most TLP languages. As a consequence, the logical foundations of the PLTL-core of TLP languages constitute an important area of investigation. Indeed, during the eighties of the last century, a big effort in this line was made, which gave rise to different PLTL-based languages whose execution mechanisms extend classical resolution for handling the temporal connectives of PLTL. The best known of those languages are Templog \cite{1}, Chronolog \cite{14} and Gabbay’s Temporal Prolog \cite{6}. In Chronolog, the next time connective (\(\circ\)) is the only temporal connective. Templog’s syntax allows the always connective (\(\Box\)) to occur in clause heads and the eventually connective (\(\Diamond\)) in clause bodies. However, Templog and Chronolog have the same expressive power and the same metalogical properties of existence of minimal model and fixpoint characterization. That is, roughly speaking, Templog and Chronolog programs are expressible using \(\circ\) as the unique temporal connective. This restriction is so strong that it allows reducing any temporal program to a (possibly infinite) classical logic program. On the contrary, Gabbay’s Temporal Prolog is a very expressive language that allows the eventuality connectives to appear in clause heads, but it does not allow \(\Box\) in clause heads. A resolution-based computation procedure is outlined in \cite{6} 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. To the best of our knowledge, during the last two decades, no other subsets of PLTL have been proposed as resolution-based languages. Hence, nowadays, Templog, Chronolog and Gabbay’s Temporal Prolog remain as the most expressive subsets of PLTL in the field of resolution-based TLP languages. In our opinion, research in this area was almost given up due to the troublesome solving (in the resolution sense) of the so-called eventualities.
In this paper, we contribute to the effort of increasing the expressive power of PLTL-resolution-based TLP languages. We propose a new approach to define TLP languages in the framework of the temporal extension of Horn clauses and provide an ad-hoc specialized resolution-based mechanism. We introduce a very expressive TLP language that allows both □ and ◇ in clause heads and bodies. In particular, with regard to temporal features, our language is strictly more expressive than Templog, Chronolog and Gabbay’s Temporal Prolog. The latter proposals are first-order languages, but TeDiLog is intended to be the propositional core language that captures the essence of our proposal with respect to temporal expressiveness. A first-order extension built on TeDiLog is beyond the scope of this paper, but it seems a natural and non-problematic way to obtain a very expressive temporal logic programming language. We provide the operational semantics of TeDiLog by means of a resolution-based mechanism. This underlying resolution mechanism is what makes TeDiLog very different from—and we hope more promising than—earlier proposals of TLP languages. Indeed TeDiLog has been designed as a specialization (for the TeDiLog language) of the invariant-free deduction method that is presented in detail in [8], where sound and complete methods of tableaux and sequents are defined for full PLTL. We should mention here the clausal resolution method introduced in [4] (see also [5]) which is also sound and complete for full PLTL. There are two crucial differences with our proposal: the clausal form and the treatment of eventualities. The so-called separated normal form (proposed in [4]) is very restricted for use as a form of program clause in a TLP language. The Fisher resolution rule for eventualities needs to generate an invariant formula from the set of clauses involved in the resolution step. Our resolution mechanism dispenses with invariant generation, i.e., does not require invariant generation.

We also endow TeDiLog with declarative semantics and prove the equivalence between operational and declarative semantics. The logical semantics given by the set of logical consequences of the program is based on the underlying logic PLTL and our resolution procedure is sound and complete with respect to this logical semantics. Regarding model-theoretic semantics, we cannot expect to have the classical minimal model property (MMP in short) that assigns to any program a minimal model, which is the intersection of all its models. The reason for this is twofold. First, the appearance of the non-conjunctive temporal connective ◇ in clause heads and of the non-finitary connective □ in clause bodies, both appearing separately, prevent the MMP from being enjoyed (see [13]). Gabbay’s Temporal Prolog also lacks the MMP for the first reason above. Second, our resolution mechanism produces (in computation time) disjunctive clauses, so TeDiLog is located in the disjunctive logic programming (DLP) paradigm, which does not enjoy the MMP even in the classical (non-temporal) case. As a consequence, in the DLP framework, the semantics of a program usually consists of the collection of all its minimal models (see e.g. [11]). Temporal disjunctive logic programming has previously been addressed in [10] where Chronolog is extended with DLP features. From the temporal point of view, Disjunctive Chronolog has the same limitations as Chronolog. The satisfaction of a Templog/Chronolog program can be reduced to the satisfaction of a classical logic program. As a consequence the minimal model characterization of Templog and Chronolog (see [2, 10, 14]) is a straightforward adaptation of the classical (disjunctive) case.
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 until connective \( U \). Hence, we directly formulate our language in terms of the temporal connectives \( U \) and its dual: the release connective \( R \).

We propose a complete algorithm which performs a systematic resolution of a goal with respect to a program. This algorithm is based on a natural extension of the classical rule for (binary) resolution (see Fig. 1) in two senses: temporal (\( \Box \) in front of clauses) and disjunctive (disjunction in head clauses). The algorithm not only performs the standard (linear) resolution between the current goal and a selected program clause, but also a controlled kind of resolution, called \( nx\text{-resolution} \), between some program clauses with a specific syntactic form. This \( nx\text{-resolution} \) is required for completeness and takes out only the clauses that are resolvents of program clauses and have a \( \Box \) connective in front of every literal in the clause.

Outline of the paper. In Section 2 we provide the basic background on PLTL and some notational conventions. In Section 3 we introduce the syntax of TeDiLog and show a program example. In Section 4 we introduce the system of rules that are the basis for the operational semantics of TeDiLog presented in Section 5 together with a sample derivation. Two declarative semantics –logical and model-theoretic– of TeDiLog are presented in Section 6, where the equivalence between the two declarative semantics is also proved. The equivalence between the operational and the declarative semantics is proved in Section 7. In the last section we summarize our contribution and outline some topics for future research.

2 Preliminaries and Notation

PLTL-formulas are built using propositional variables \( (p, q, \ldots) \) from a set Prop, the classical connectives \( \neg \) and \( \land \), and the temporal connectives \( \Diamond \) and \( U \). In the sequel, formula means PLTL-formula. A PLTL-structure \( M \) is a pair \( (S_M, V_M) \) where \( S_M \) is a denumerable sequence of states \( s_0, s_1, s_2, \ldots \) and \( V_M : S_M \to \text{Prop} \) is a mapping that specifies which atomic propositions are true in each state in \( S_M \). Formulas are interpreted in a state \( s_j \) of a PLTL-structure \( M \) as follows:

- \( \langle M, s_j \rangle \models p \) iff \( p \in V_M(s_j) \) for \( p \in \text{Prop} \)
- \( \langle M, s_j \rangle \models \neg \varphi \) iff \( \langle M, s_j \rangle \not\models \varphi \)
- \( \langle M, s_j \rangle \models \varphi \land \psi \) iff \( \langle M, s_j \rangle \models \varphi \) and \( \langle M, s_j \rangle \models \psi \)
- \( \langle M, s_j \rangle \models \Diamond \varphi \) iff \( \langle M, s_{j+1} \rangle \models \varphi \)
- \( \langle M, s_j \rangle \models \varphi U \psi \) iff there exists \( k \geq j \) such that \( \langle M, s_k \rangle \models \psi \) and \( \langle M, s_i \rangle \models \varphi \) holds for every \( i \in \{j, \ldots, k-1\} \).

The remaining linear-time connectives can be defined in terms of the previous ones. In particular, \( \varphi R \psi \equiv \neg (\neg \varphi U \neg \psi) \), \( \Diamond \varphi \equiv \neg \varphi U \varphi \), \( \Box \varphi \equiv \neg \Diamond \neg \varphi \). Note that \( \Box \varphi \equiv \neg \Diamond \neg \varphi \). Formulas of the form \( \varphi U \psi \) and \( \Diamond \varphi \) are called eventualities.

Given a set of formulas \( \Phi = \{ \varphi_1, \ldots, \varphi_n \} \), \( \neg \Phi \) stands for the disjunction of the negation of all the formulas in \( \Phi \), i.e., \( \neg \Phi \equiv \neg \varphi_1 \lor \ldots \lor \neg \varphi_n \).

For a set of formulas \( \Phi \), we say that \( \langle M, s_j \rangle \models \Phi \) iff \( \langle M, s_j \rangle \models \gamma \) for all \( \gamma \in \Phi \).

We say that \( M \) is a model of \( \Phi \), in symbols \( M \models \Phi \), iff \( \langle M, s_0 \rangle \models \Phi \). 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. The crucial idea behind the resolution mechanism of TeDiLog is based in the following equisatisfiability result that relates two eventualities.

**Proposition 1.** Let Δ be a set of formulas, Δ₁ = Δ ∪ {p₂ ∨ (p₁ ∧ o(p₁ U p₂))} and Δ₂ = Δ ∪ {p₂ ∨ (p₁ ∧ o((p₁ ∧ ¬Δ) U p₂))}. Then Δ₁ and Δ₂ are equisatisfiable.

**Proof.** Suppose that Δ₁ has a model M. If ⟨M, s₀⟩ |− Δ ∪ {p₂}, then M is also a model of Δ₂. Otherwise, if ⟨M, s₀⟩ |− p₁ ∧ o(p₁ U p₂) then p₂ should be satisfied in a later state s_j with j ≥ 1 and p₁ is true in all the states s_h such that 0 ≤ h < j. Let k be the greatest index in {0, . . . , j − 1} such that ⟨M, s_k⟩ |− Δ and Δ is not satisfied in the states s_k+1, . . . , s_j−1 of M. Then, we can construct a model M’ of Δ by simply deleting the states s_0, . . . , s_k−1 in M. It is easy to see that M’ is also a model of p₁ ∧ o((p₁ ∧ ¬Δ) U p₂). Hence, M’ |− Δ₂. In the converse direction, any model of Δ₂ is itself a model of Δ₁.

A formula χ is a logical consequence of a set of formulas ϕ̃, denoted as ϕ̃ |− χ, iff for every PLTL-structure M and every s_j ∈ S_M if ⟨M, s_j⟩ |− ϕ̃ then ⟨M, s_j⟩ |− χ.

As denotational convention, we use two kind of superscripts on unary temporal connectives. First, a superscript i varying on N represents the sequence formed by i identical connectives, in particular empty for i = 0. For instance, o^i represents the sequence o . . . o of length i. Second, the especial case of superscript b varying in {0, 1} that is written on the connectives □ and ◦. For instance, the expression □^0φ represents the formula φ, whereas □^1φ represents the formula □φ. Along the rest of the paper superscripts b (from bit) range in {0, 1}. Both kind of superscripts are notation, hence they are not part of the syntax.

### 3 The Language TeDiLog

In this section we introduce the syntax of TeDiLog and provide an illustrative program example. The programming language TeDiLog is a twofold extension of Horn clauses. On one hand TeDiLog introduces temporal connectives in atoms and, on the other hand, TeDiLog’s program clause heads are disjunctions of atoms.

First, we present the notion of atom denoted by the metavariable A as follows

\[ L ::= p | ¬p \quad T ::= LU p | LR p | ◦ p | □ p \quad A ::= o^i p | o^i T \quad (i ∈ N) \]

where p ∈ Prop, L stands for (classical) literal and T for temporal atom. Second, program clauses and goal clauses are defined as follows

\[ H ::= ⊥ | A ∨ H \quad B ::= T | A ∧ B \quad D ::= ◦ b(A ∨ H ↔ B) \quad G ::= □ b(⊥ ↔ B) \]

where ⊥ (resp. T) represents the empty disjunction (resp. conjunction), H stands for head, B for body, D for (disjunctive) program clause and G for goal clause.

Then, a program is a set of program clauses and a goal is a set of goal clauses. Due to the superscript b (see Section 2), the metavariable D represents two kinds of clauses. The expression □^b(H ↔ B), for b = 0, represents H ↔ B, which is called a now-clause,
whereas for $b = 1$, it represents $\Box (H \leftarrow B)$, which is called an always-clause. The same classification applies to the goal clauses denoted by $G$. In particular, $\Box^b (\perp \leftarrow \top)$ represents the two possible syntactic forms of the empty clause, as now- or always-clause. Semantically, $\Box^b (H \leftarrow B)$ is equivalent to $\Box^b (H \lor \neg B)$. For a set of (program and goal) clauses $\Phi$, the set $\text{alw}(\Phi)$ is formed by the always-clauses in $\Phi$ and $\text{now}(\Phi)$ is the set $\Phi \setminus \text{alw}(\Phi)$. We assume there is neither repetitions nor established order in the atoms of a head or a body.

Given a head, body, program clause or goal clause $\varphi$, we denote by $\circ \varphi$ the head, body, program clause or goal clause that is obtained by adding one $\circ$ connective to every atom in $\varphi$. As a consequence, $\circ \top$ denotes $\top$ and $\circ \bot$ denotes $\bot$, i.e., $\top$ and $\bot$ can be considered, respectively, as a body and a head in which every atom is of the form $\circ A$. For instance, $\Box ((p \lor q) \lor r \leftarrow \circ \circ \circ p \land \Box s)$ and $\circ (\circ p \lor (q \land r) \leftarrow \top)$ respectively denote the program clauses $\Box ((p \lor q) \lor r \leftarrow \circ \circ \circ p \land \Box s)$ and $\circ (\circ p \lor (q \land r) \leftarrow \top)$.

An atom is said to be $\circ$-free if it is a temporal atom or a classical propositional atom.

The example below illustrates the expressiveness of TeDiLog.

Example 1. The following set of clauses (partially) specifies a system where a device $dv$ and a system manager $sm$ interact with each other.

1. $\Box (\text{waiting}_{dv} \lor \text{ack}_{sm} \leftarrow \text{req}_{dv})$
2. $\circ (\circ \circ \text{ack}_{sm} \leftarrow \text{req}_{dv})$
3. $\Box (\text{waiting}_{sm} \lor \text{eop}_{dv} \leftarrow \text{ack}_{sm})$
4. $\Box (\text{working}_{dv} \lor \text{eop}_{dv} \leftarrow \text{ack}_{sm})$
5. $\Box (\neg \text{req}_{dv} \lor \text{eop}_{dv} \leftarrow \text{working}_{dv})$
6. $\Box (\neg \text{ack}_{sm} \lor \text{eop}_{dv} \leftarrow \text{req}_{dv})$
7. $\circ (\neg \text{req}_{dv} \lor \text{eop}_{dv} \leftarrow \text{req}_{dv})$
8. $\Box (\text{ctr}_{sm} \leftarrow \top)$
9. $\Box (\circ \text{ctr}_{sm} \leftarrow \text{ctr}_{sm})$
10. $\Box (\circ \text{conn}_{sm} \leftarrow \text{conn}_{sm})$
11. $\Box (\text{com}_{dv} \leftarrow \circ \text{conn}_{dv})$
12. $\Box (\text{com}_{sm} \leftarrow \circ \text{conn}_{sm})$
13. $\Box (\neg \text{waiting}_{dv} \lor \text{req}_{dv} \leftarrow \text{eop}_{dv})$

The system specified in Example 1 works as follows. Each time that the device $dv$ needs to execute a process, the device $dv$ sends a request $\text{req}_{dv}$ to the system manager $sm$ to get permission to execute the process and goes into waiting-state until the system manager $sm$ sends the acknowledgement signal $\text{ack}_{sm}$ giving permission to execute the process (clause 1). Whenever the device $dv$ asks for permission to execute a process, the system manager $sm$ will eventually give permission by sending the acknowledgement signal $\text{ack}_{sm}$ in a later state (clause 2). Once the system manager produces the acknowledgement signal $\text{ack}_{sm}$ to give permission to the device $dv$, the system manager $sm$ goes into waiting-state (clause 3) whereas the device $dv$ goes into working-state (clause 4) until the device $dv$ communicates the end of the process by means of the $\text{eop}_{dv}$ signal. Whenever the device $dv$ is in working-state, i.e., executing a process, it will not send another request signal until the execution of the process is ended (clause 5). Clause 6 states that whenever the device $dv$ generates the $\text{eop}_{dv}$ signal, then it will not be in working-state until it receives the $\text{ack}_{sm}$ signal giving permission to execute another process. Clause 7 states that once the device $dv$ sends a request signal $\text{req}_{dv}$ to ask for permission to execute a process, it will not send another request signal until the end of that process. Additionally, the system manager innerly
generates a control signal \( \text{ctr}_\text{sm} \) from time to time (clause 8) which is eventually followed by the signal \( \text{conn}_\text{sm} \) (clause 9) provoking an answer \( \text{conn}_\text{dv} \) from the device \( \text{dv} \) (clause 10). The interaction generated after the control signal \( \text{ctr}_\text{sm} \) corresponds to the fact that the system manager \( \text{sm} \) has to regularly control whether the device is correctly connected to the system. The device \( \text{dv} \) is considered to be in communicating-state \( \text{com}_\text{dv} \) while the arising of the \( \text{conn}_\text{dv} \) signal (now or in a future moment) is guaranteed (clause 11). Similarly for \( \text{com}_\text{sm} \) with respect to the system manager \( \text{sm} \) and the signal \( \text{conn}_\text{sm} \) (clause 12). Clause 13 states that whenever the device \( \text{dv} \) ends the execution of a process, it will not be in waiting-state until another request is generated.

Once the specification of the system is given, one could check, by means of goals, whether the system verifies some properties such as fairness, liveness, safety, mutual exclusion, etc. For instance we would be interested in checking whether the device \( \text{dv} \) and the system manager \( \text{sm} \) will always keep communicating with each other. The corresponding goal would be \( \{ \bot \leftarrow \Box \text{com}_\text{dv} \land \Box \text{com}_\text{sm} \} \). Note that this goal cannot be expressed in the languages in \([1, 14, 6]\). We also remark that the clauses 2, and 8-10 in Example 1, are also clauses of the language in \([6]\), but they are forbidden in \([14, 1]\) because of the eventualities in their heads.

4 The Rule System

In this section, we introduce the rule system that constitutes the basis of the operational semantics of TeDiLog. Our system includes a Resolution Rule, a collection of Temporal Rules for decomposing temporal atoms, and two auxiliary rules respectively for subsumption and jumping to the next state.

The TeDiLog’s Resolution Rule \((\text{Res})\), in Fig. 1, is a natural generalization of the classical rule for binary resolution.

\[
(\text{Res}) \quad \frac{\Box b (A \lor H \leftarrow B) \quad \Box b' (H' \leftarrow A \land B')}{} \quad \Box b \times b' (H \lor H' \leftarrow B \land B')
\]

**Fig. 1.** The Resolution Rule

The rule \((\text{Res})\) applies to two temporal clauses (the premises) that can be headed or not by an always connective and verify that one of the atoms in the head of the first clause is in the body of the second clause. Then, a new clause, called the resolvent, is added to the target set of clauses. The rule \((\text{Res})\) is written in the usual format of premises and resolvent separated by an horizontal line. Note that the resolvent is in general a program clause, but in particular when the premises respectively are a single-headed program clause and a goal clause, the resolvent is a goal clause. Note also that, by means of the product \( b \times b' \) in the superscript of the resolvent, only when both premises are always-clauses, the resolvent is also an always-clause.

The Temporal Rules serve to transform the set of clauses according to the inductive

---

3 The system checks the unsatisfiability of the eventuality \( \Diamond \neg \text{com}_\text{dv} \lor \Diamond \neg \text{com}_\text{sm} \) with respect to the specification.
Fig. 2. The Context-Free Rules

\[(U_{+}) \quad \Box \neg((p_1 \cup p_2) \vee H \leftarrow B)
\]

\[\equiv \{\Box (p_2 \vee p_1 \vee H \leftarrow B), \quad \Box (p_2 \vee \diamond(p_1 \cup p_2) \vee H \leftarrow B)\}\]

\[(U_{-}) \quad \Box \neg((p_1 \cup p_2) \vee H \leftarrow B)
\]

\[\equiv \{\Box (p_2 \vee H \leftarrow p_1 \wedge B), \quad \Box (p_2 \vee \diamond(p_1 \cup p_2) \vee H \leftarrow B)\}\]

\[(U_{B+}) \quad \Box \neg(H \leftarrow (p_1 \cup p_2) \wedge B)
\]

\[\equiv \{\Box (p_2 \wedge B), \quad \Box (p_1 \wedge \diamond(p_1 \cup p_2) \wedge B)\}\]

\[(U_{B-}) \quad \Box \neg(H \leftarrow (p_1 \cup p_2) \wedge B)
\]

\[\equiv \{\Box (p_1 \vee H \leftarrow \diamond(p_1 \cup p_2) \wedge B)\}\]

\[(R_{H+}) \quad \Box \neg((p_1 \cap p_2) \vee H \leftarrow B)
\]

\[\equiv \{\Box (p_2 \vee H \leftarrow B), \quad \Box (p_1 \vee \diamond(p_1 \cap p_2) \vee H \leftarrow B)\}\]

\[(R_{H-}) \quad \Box \neg((p_1 \cap p_2) \vee H \leftarrow B)
\]

\[\equiv \{\Box (p_2 \vee H \leftarrow B), \quad \Box (p_1 \vee \diamond(p_1 \cap p_2) \vee H \leftarrow p_1 \wedge B)\}\]

\[(R_{B+}) \quad \Box \neg(H \leftarrow (p_1 \cap p_2) \wedge B)
\]

\[\equiv \{\Box \neg((p_1 \cap p_2) \vee H \leftarrow p_1 \wedge B), \quad \Box (p_2 \wedge \diamond(p_1 \cap p_2) \wedge B)\}\]

\[(R_{B-}) \quad \Box \neg(H \leftarrow (p_1 \cap p_2) \wedge B)
\]

\[\equiv \{\Box (p_1 \vee H \leftarrow p_2 \wedge B), \quad \Box (p_2 \wedge \diamond(p_1 \cap p_2) \wedge B)\}\]

---

Fig. 3. The set of clauses \(\text{def}(a, L, \Delta)\)

\[
\text{def}(a, L, \emptyset) = \{\Box (\bot \leftarrow a)\}
\]

\[
\text{def}(a, p, \Delta) = \{\Box (p \leftarrow a)\} \cup \{\Box (H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset
\]

\[
\text{def}(a, \neg p, \Delta) = \{\Box (\bot \leftarrow p \wedge a)\} \cup \{\Box (H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset
\]

---

Fig. 4. The Context-Dependent Rules

\[
(U_{C+}) \quad \Omega \cup \{\Box \neg((p_1 \cup p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}
\]

\[\equiv \Omega \cup \{p_2 \vee p_1 \vee H_i \leftarrow B_i, \quad p_2 \vee \diamond(a \cup p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\}
\]

\[\cup \text{ def}(a, p_1, \text{now}(\Omega))\]

\[\cup \{\Box \neg((p_1 \cup p_2) \vee \diamond H_i \leftarrow \diamond B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\]

\[
(U_{C-}) \quad \Omega \cup \{\Box \neg((\neg p_1 \cup p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}
\]

\[\equiv \Omega \cup \{p_2 \vee H_i \leftarrow p_1 \wedge B_i, \quad p_2 \vee \diamond(a \cup p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\}
\]

\[\cup \text{ def}(a, \neg p_1, \text{now}(\Omega))\]

\[\cup \{\Box \neg((\neg p_1 \cup p_2) \vee \diamond H_i \leftarrow \diamond B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\]

\[
(R_{C+}) \quad \Omega \cup \{\Box \neg((p_1 \cap p_2) \wedge B_i) \mid 1 \leq i \leq n\}
\]

\[\equiv \Omega \cup \{H_i \leftarrow p_1 \wedge B_i, \quad H_i \leftarrow p_2 \wedge \diamond(\neg a \cap p_2) \wedge B_i \mid 1 \leq i \leq n\}
\]

\[\cup \text{ def}(a, \neg p_1, \text{now}(\Omega))\]

\[\cup \{\Box \neg((p_1 \cap p_2) \wedge \diamond B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\]

\[
(R_{C-}) \quad \Omega \cup \{\Box \neg((\neg p_1 \cap p_2) \wedge B_i) \mid 1 \leq i \leq n\}
\]

\[\equiv \Omega \cup \{p_1 \vee H_i \leftarrow p_2 \wedge B_i, \quad H_i \leftarrow p_2 \wedge \diamond(\neg a \cap p_2) \wedge B_i \mid 1 \leq i \leq n\}
\]

\[\cup \text{ def}(a, p_1, \text{now}(\Omega))\]

\[\cup \{\Box \neg((\neg p_1 \cap p_2) \wedge \diamond B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}\]

for \(n \geq 1, a \in \text{Prop}\) is fresh and def is in Fig. 3.
The set $\Gamma_0$ has been obtained from $\Gamma_1$ by splitting the clauses in $\{\Box^b((p_1 \cup \nabla_i \nabla_j) \cup H_i \leftarrow B_i) | 1 \leq i \leq n\}$ with $b_i = 1$ while clauses with $b_i = 0$ remain unchanged.

Note that, in the name of the rules, $H$ and $B$ respectively denote that the temporal atom occurs in the head and in the body. Also, $+$ and $-$ respectively stand for the positive and negative nature of the first propositional literal in the temporal atom.
Then, like in the case of the rule \((\cup H_\gamma)\), the set \(\Gamma_1\) is equisatisfiable to the set

\[
\Gamma_2 \equiv \Omega \cup \{ p_2 \lor p_1 \lor H_i \leftarrow B_i, \quad p_2 \lor \circ(p_1 \cup p_2) \lor H_i \leftarrow B_i \mid 1 \leq i \leq n \}
\]
\[
\cup \{ \Box b_i(\circ(p_1 \cup p_2) \lor \circ H_i \leftarrow \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n \}
\]

Now, by Proposition 1, the above set \(\Gamma_2\) is also equisatisfiable to the following set

\[
\Gamma_3 \equiv \Omega \cup \{ p_2 \lor p_1 \lor H_i \leftarrow B_i, \quad p_2 \lor \circ((p_1 \land \neg \Omega) \cup p_2) \lor H_i \leftarrow B_i \mid 1 \leq i \leq n \}
\]
\[
\cup \{ \Box b_i(\circ(p_1 \cup p_2) \lor \circ H_i \leftarrow \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n \}
\]

Due to the subformula \(\neg \Omega\), the set \(\Gamma_3\) is not exclusively formed by clauses. Now, the always-clauses in \(\Omega\) can be excluded from the negation of \(\Omega\) since, in general, the two sets \(\{ \Box \psi, \circ((\gamma \land (\varphi \lor \neg \Box \psi)) \cup \delta) \}\) and \(\{ \Box \psi, \circ((\gamma \land \varphi) \cup \delta) \}\) are logically equivalent. Hence, using \(\Delta\) to denote \(\text{now}(\Omega)\), which is called the context, \(\Gamma_3\) is logically equivalent to

\[
\Gamma_4 \equiv \Omega \cup \{ p_2 \lor p_1 \lor H_i \leftarrow B_i, \quad p_2 \lor \circ((p_1 \land \neg \Delta) \cup p_2) \lor H_i \leftarrow B_i \mid 1 \leq i \leq n \}
\]
\[
\cup \{ \Box b_i(\circ(p_1 \cup p_2) \lor \circ H_i \leftarrow \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n \}
\]

Fortunately, the negation of a set of now-clauses can be transformed (by distribution) into an equivalent set of now-clauses. That is, given a non-empty set of now-clauses \(\Phi = \{ N_1, \ldots, N_r \}\), the set \(\neg \Phi\) is the set of clauses obtained by applying the distribution laws to the formula \(\neg N_1 \lor \ldots \lor \neg N_r\). But \(\circ(p_1 \land \neg \Delta) \cup p_2\) is not an atom, therefore, in the conclusion of the rule \((\cup C_\gamma)\), the formula \(p_1 \land \neg \Delta\) is substituted by a new variable \(a\) and a set of clauses \(\text{def}(a, p_1, \Delta)\) that defines the meaning of \(a\) is added. The set of clauses \(\text{def}(a, L, \Delta)\) is defined in Fig. 3 and is the result of transforming the formula \(\Box(L \land \neg \Delta \leftarrow a)\) to a set of program and goal clauses. As a consequence, \(\Gamma_3\) and \(\Gamma_4\) are equisatisfiable. The clauses that correspond to the converse implication \(\Box(L \land \neg \Delta \leftarrow a)\) are not needed to achieve equisatisfiability of the two sets, hence we leave them out. This matter is shown in detail in the soundness proof (Section 7.1).

To round off the Temporal Rules we present the derived rules for \(\circ\) and \(\Box\) in Fig. 5. The context-free rules \((\Box H_\alpha), (\circ H_\alpha), (\Box B_\alpha), (\circ B_\alpha)\) and \((\Box C_\alpha), (\circ C_\alpha)\) are respectively derived from the context-free rules \((\cup H_\alpha), (\cup B_\alpha), (\Box H_\alpha), (\Box B_\alpha)\) and \((\Box C_\alpha), (\Box C_\alpha)\) and the definitions \(\Box \varphi \equiv \neg \varphi \cup \varphi\) and \(\circ \varphi \equiv \neg \varphi \cup \varphi\). In the same way, the context-dependent rules \((\circ C_\alpha)\) and \((\Box C_\alpha)\) are respectively derived from the context-dependent rules \((\cup C_\alpha)\) and \((\Box C_\alpha)\).

Finally, our resolution mechanism requires (for completeness) a subsumption rule \((Sbm)\) and a rule called unnext (namely \((Unx)\)) that, roughly speaking, serves to jump to the next state. The subsumption rule is formulated as follows

\[
(Sbm) \quad \{ \Box b(H \leftarrow B), \circ b(H' \leftarrow B') \} \rightarrow \{ \Box b(H' \leftarrow B') \}
\]

if \(H' \subseteq H\) and \(B' \subseteq B\)

It is said that the clause \(\Box b(H \leftarrow B)\) is subsumed by the clause \(\circ b(H' \leftarrow B')\).

In order to formulate the rule \((Unx)\), we first introduce the unnext operator that applies to a set of clauses. Given a set of clauses \(\Psi\), \(\text{unnext}(\Psi)\) is the set of clauses that should be satisfied at the next state of a state that satisfies the set \(\Psi\). Hence, we define

\[
\text{unnext}(\Psi) = \text{alw}(\Psi) \cup \{ H \leftarrow B \mid \Box b(\circ H \leftarrow \circ B) \in \Psi \}.
\]
Note that \( \text{unnext} \) implicitly uses the equivalence between \( \square (H \leftarrow B) \) and \( \{ H \leftarrow B, \square \square (H \leftarrow B) \} \). It is worth to remember that \( \top \) and \( \bot \) respectively represent the empty body and the empty head, and consequently, \( \circ \top \) denotes \( \top \) and \( \circ \bot \) denotes \( \bot \), i.e., \( \top \) and \( \bot \) can be considered, respectively, as a body and a head in which every atom is of the form \( \circ A \). For example, \( \text{unnext}(\{ \square (q \leftarrow r), \square (q \leftarrow \top) \}) \) is the set \( \{ \square (q \leftarrow \top), \circ (q \leftarrow r), \circ (q \leftarrow \top) \} \).

The rule \( \text{(Unx)} \) simply applies the unnext operator to the program \( \Pi \) and the goal \( \Gamma \).

\[
\text{(Unx)} \quad (\Pi, \Gamma) \longmapsto (\text{unnext}(\Pi), \text{unnext}(\Gamma))
\]

### 5 Operational Semantics

In this section we formulate the operational semantics of TeDiLog and give a sample derivation.

Our notion of IFT-derivation (Invariant-Free Temporal-derivation), not only captures the successive subgoals, but also the development of the program.

**Definition 1.** Let \( \Pi \) be a program and \( \Gamma \) a goal. An IFT-derivation from \( \Pi \) with top-goal \( \Gamma \) consists of a (possibly infinite) sequence of pairs (program, goal)

\[
(\Pi_0, \Gamma_0), (\Pi_1, \Gamma_1), \ldots, (\Pi_i, \Gamma_i), \ldots
\]

where \( (\Pi_0, \Gamma_0) = (\Pi, \Gamma) \) and each \( i \geq 1 \) satisfies one of the following conditions

(i) \( \Pi_i \cup \Gamma_i = \Pi_{i-1} \cup \Gamma_{i-1} \cup \{ \square \circ (H \leftarrow B) \} \) where \( \square \circ (H \leftarrow B) \) is the resolvent that results from applying the rule \( \text{(Res)} \) to some pair of clauses in \( \Pi_{i-1} \cup \Gamma_{i-1} \).

(ii) \( \Pi_i \cup \Gamma_i = ((\Pi_{i-1} \cup \Gamma_{i-1}) \setminus \Sigma) \cup \Sigma' \) where \( \Sigma \subseteq (\Pi_{i-1} \cup \Gamma_{i-1}) \) and \( \Sigma \mapsto \Sigma' \) according to a temporal rule or the subsumption rule.
The resolvent of $D$.

Now, resolution is applied to

By the rule

and $\Pi$.

In our case the underlying logic is

the set of bodies that are logical consequences (regarding the underlying logic) of

we prove the equivalence of these two declarative semantics.

We remark that the refutation procedure that ensures the completeness of our IFT-resolution mechanism (see Fig. 6) requires a controlled kind of resolution between two program clauses, not a complete closure (or saturation) with respect to resolvents. The following example shows how IFT-resolution deals with eventualities.

Example 2. Let $\Pi_0 = \{D_1\}$ be a program and $\Gamma_0 = \{G_1\}$ a goal where

By rule $(U C_+)$ we have $\Pi_1 = \{D_2, D_3, D_4\}$ and $\Gamma_1 = \{G_1, G_2\}$ where

Now, resolution is applied to $(D_2, G_1)$ and $(D_3, G_1)$ to yield $\Pi_2 = \Pi_1 \cup \{D_5, D_6\}$

and $\Gamma_2 = \Gamma_1 \cup \{G_1, G_2\}$. Then, the rule $(U \neg x)$ is applied to $(\Pi_2, \Gamma_2)$ to yield $\Gamma_3 = \Gamma_2$ and $\Pi_3 = \{D_7, D_8\}$ where

By the rule $(U C_+)$, $\Gamma_3$ is extended with $G_3 = \square (\bot \leftarrow z)$ and $D_8$ is replaced with

The resolvent of $D_9$ and $G_2$ is $b \leftarrow \top$, whose resolution with $G_1$ produces $\bot \leftarrow \top$.

6 Declarative Semantics

In this section we present the logical and the model-theoretic semantics of TeDiLog and we prove the equivalence of these two declarative semantics.

On one hand, the ordinary logical characterization of a logic program is given by the set of bodies that are logical consequences (regarding the underlying logic) of the program. In our case the underlying logic is PLTL and a goal $\Gamma = \{\square b_1 (\bot \leftarrow B_1), \ldots, \square b_n (\bot \leftarrow B_n)\}$ is understood as the conjunction of the goal clauses in $\Gamma$.

Since a goal clause $\square b (\bot \leftarrow B)$ represents the formula $\neg (\top ^b B)$, the set $\Gamma$ is logically equivalent to the formula $\neg (\top ^b B_1 \land \ldots \land \neg (\top ^b B_n)$, i.e., $\Gamma$ is equivalent to the formula $\neg (\top ^b B_1 \lor \ldots \lor \top ^b B_n)$).

Definition 2. The semantics of a program $\Pi$ is logically characterized as the set of all formulas of the form $\top ^{b_1} B_1 \lor \ldots \lor \top ^{b_n} B_n$ that are logical consequences of $\Pi$.

On the other hand, the model-theoretic characterization of a program $\Pi$ is given by the collection of all the minimal models of $\Pi$ (as it is usual in DLP). Next we will define the notion of minimal model which are cyclic structures.
Definition 3. A PLTL-structure $\mathcal{M}$ is cyclic iff there exist $j \geq 0$ and $k \geq j$ such that $V(s_{j+h * (k-j+1) + \ell}) = V(s_j + \ell)$ for every $h \geq 0$ and every $\ell \in \{0, \ldots, k-j\}$.

Definition 4. A model $\mathcal{M}$ of a program $\Pi$ is minimal iff $\mathcal{M}$ is cyclic and there is no a distinct cyclic model $\mathcal{M}'$ of $\Pi$ that verifies: $V_{\mathcal{M}'}(s_i') \subseteq V_{\mathcal{M}}(s_i)$ for all $i \in \mathbb{N}$.

Now, both declarative semantics can be proved to be equivalent.

Theorem 1. Let $\Pi$ be a program and $\Gamma = \{\Box b_1(\perp \leftarrow B_1), \ldots, \Box b_n(\perp \leftarrow B_n)\}$ be a goal. The formula $\varphi = \Diamond b_1 B_1 \lor \ldots \lor \Diamond b_n B_n$ is a logical consequence of $\Pi$ iff every minimal model of $\Pi$ is also a model of $\varphi$.

Proof. The left to right implication is trivial. For the converse, suppose that $\varphi$ is true in all the minimal models of $\Pi$. Then $\varphi$ is true in all the cyclic models of $\Pi$, since the connectives used in $\varphi$ are monotonic$^5$. Now, suppose that there is a non-cyclic model $\mathcal{M}$ of $\Pi$ that is not a model of $\varphi$, by the expressiveness of PLTL (see [15]), there exists a cyclic model $\mathcal{M}'$ of $\Pi$ with $S_{\mathcal{M}'} = (s_0, s_1, \ldots, s_{j-1}) \cdot (s_j, \ldots, s_k)\omega$ where $s_0, s_1, \ldots, s_j, \ldots, s_k$ is an initial segment of $S_{\mathcal{M}}$ and $0 \leq j \leq k$, such that $\mathcal{M}'$ is not a model of $\varphi$.

7 Equivalence between Operational and Declarative Semantics

In this section we prove the equivalence between the operational and the declarative semantics of TeDiLog by proving the equivalence between the operational and the logical semantics. The equivalence between the operational and the logical semantics is divided into soundness and completeness. The first subsection is devoted to soundness which is proved by ensuring that each rule application preserves satisfiability. In the second subsection we provide an algorithm for systematic construction of IFT-derivations. On the basis of this algorithm we prove, in Subsection 7.3, that the operational semantics is complete with respect to the logical semantics. That is, the systematic resolution algorithm in Fig. 6 obtains a refutation for any unsatisfiable pair consisting of a program and a goal. Closely related technical details can be found in [7] where invariant-free clausal resolution is presented for the case of general clauses.

7.1 Soundness

Soundness is a consequence of the fact that each rule (used by the resolution method) preserves satisfiability.$^6$ The soundness of our system can be guaranteed rule by rule. A rule is sound whenever it preserves the satisfiability from the initial set to the target set of every application of the rule. It is common that some rules preserve the stronger property of logical equivalence, although satisfiability is enough for soundness. Next, we analyze the properties that our rules preserve in order to prove that our resolution system is sound.

---

$^5$ That is, the extension of a model of $\varphi$ cannot falsify $\varphi$. For an interesting discussion of monotonicity in temporal logic programs we refer to [13]

$^6$ Some of them preserve even logical equivalence.
**Proposition 2.** Every application of the rule (Res), the context-free rules and the rule (Sbm) yields a new set of clauses that is logically equivalent to the initial set.

**Proof.** When (Res) is applied to a pair of sets of clauses \((\Pi, \Gamma)\), it adds a logical consequence of the premises (the resolvent) giving a new pair \((\Pi', \Gamma')\) verifying \(\Pi' \cup \Gamma \subseteq \Pi' \cup \Gamma'\) and that is logically equivalent to \((\Pi, \Gamma)\).

Given a pair of sets of clauses \((\Pi, \Gamma)\), the context-free rules for the connectives \(\land\) and \(\lor\) replace a clause \(C\) in \(\Pi \cup \Gamma\) with two clauses \(C_1\) and \(C_2\) obtaining a new pair \((\Pi', \Gamma')\) such that \(\Pi' \cup \Gamma' = ((\Pi \cup \Gamma) \setminus \{C\}) \cup \{C_1, C_2\}\). The sets \(\Pi' \cup \Gamma'\) and \(\Pi \cup \Gamma\) are logically equivalent since the clauses containing literals of the form \(\square\) and \(\diamond\) are replaced with the clauses obtained taking into account the inductive definitions of the connectives \(\land\) and \(\lor\). In particular, every application of the context-free rules for the connectives \(\square\) and \(\diamond\) yields a new set of clauses that is logically equivalent to the initial set. Hence, they are also sound.

For soundness of (Sbm), suppose that \(\square^b(H \leftarrow B)\) and \(\square^b(H' \leftarrow B')\) are two different clauses in \(\Pi \cup \Gamma\) and that \(H' \subseteq H\) and \(B' \subseteq B\). It is trivial that any model of \(\Pi \cup \Gamma\) is also a model of \(\Pi \cup \Gamma \setminus \{\square^b(H \leftarrow B)\}\) and vice-versa. \(\square\)

**Proposition 3.** The initial set and the target set of every application of the context-dependent rules are equisatisfiable.

**Proof.** Given an initial set of clauses \(\Sigma = \Omega \cup \Phi\) where

\[
\Phi = \{\square^b(p_1 \land p_2 \lor H_i \leftarrow B_i) \mid 1 \leq i \leq n\}
\]

with \(n \geq 1\) and \(\Omega \cap \Phi = \emptyset\), the application of the rule (\(U C_+\)) replaces the non-empty set \(\Phi\) with a set of clauses \(\forall \) such that

\[
\Psi_1 = \{p_2 \lor p_1 \lor H_i \leftarrow B_i, p_2 \lor \diamond(a \land p_2) \lor H_i \leftarrow B_i \mid 1 \leq i \leq n\}
\]

\[
\Psi_2 = \text{def}(a, p_1, \Delta)
\]

\[
\Psi_3 = \{\square(\diamond(p_1 \land p_2) \lor \diamond H_i \leftarrow \diamond B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\]

where \(\Delta = \text{now}(\Omega)\), \(a \in \text{Prop}\) is fresh, \(\text{def}(a, p, \Delta) = \{\square(p_1 \leftarrow a)\} \cup \{\square(H \leftarrow B \land a) \mid H \leftarrow B \in \neg \Delta\}\) if \(\Delta \neq \emptyset\) and \(\text{def}(a, p, \Delta) = \square(\bot \leftarrow a)\) if \(\Delta = \emptyset\). So the new set \(\Sigma'\) is \(\Omega \cup \Psi\). The set \(\Phi\) is equivalent, by the inductive definition of \(U\) and the distribution laws, to the set \(\Phi_1 \cup \Phi_2\) where

\[
\Phi_1 = \{p_2 \lor p_1 \lor H_i \leftarrow B_i, p_2 \lor \diamond(p_1 \land p_2) \lor H_i \leftarrow B_i \mid 1 \leq i \leq n\}
\]

and \(\Phi_2 = \Psi_3\). Now we show that (1) if \(\Sigma\) is satisfiable then \(\Sigma'\) is satisfiable and (2) if \(\Sigma'\) is satisfiable then \(\Sigma\) is satisfiable.

(1) Let \(\mathcal{M}\) be a model of \(\Sigma\), i.e., \(\langle \mathcal{M}, s_0 \rangle \models \Sigma\). Since \(a\) does not appear in the \(p_2 \lor H_i \leftarrow B_i\), we build a model \(\mathcal{M}'\) of \(\Sigma'\) in the following two cases. First, consider that \(\langle \mathcal{M}, s_0 \rangle \models p_2 \lor H_i \leftarrow B_i\) for all \(i \in \{1, \ldots, n\}\). Then we define

- \(a \notin V_{\mathcal{M}'}(s_k)\) for every \(k \in \mathbb{N}\)
- \(p \in V_{\mathcal{M}'}(s_k)\) if \(p \in V_{\mathcal{M}}(s_k)\) for all \(k \in \mathbb{N}\) and all \(p \in \text{Prop}\) such that \(p \neq a\).
Second, if \( \langle M, s_0 \rangle \not\models p_2 \lor H_i \leftarrow B_i \) for some \( i \in \{1, \ldots, n\} \), then we can ensure that \( \langle M, s_0 \rangle \models \{p_1, \circ(p_1 \cup p_2)\} \). Let \( x \) be the least \( z \geq 1 \) such that \( \langle M, s_z \rangle \models p_2 \) and let \( y \) be the greatest \( z \) such that \( 0 \leq z < x \) and \( \langle M, s_z \rangle \models \Omega \cup \{p_1, \circ(p_1 \cup p_2)\} \). Since \( \langle M, s_j \rangle \models \text{alw}(\Omega) \) for every \( j \geq 0 \), we can ensure that \( y \) is the greatest \( z \in \{0, \ldots, x-1\} \) such that \( \langle M, s_z \rangle \models \text{now}(\Omega) \cup \{p_1, \circ(p_1 \cup p_2)\} \).

As a consequence of the choice of \( x \) and \( y \), it holds that

\[
\langle M, s_y \rangle \models \text{now}(\Omega) \cup \{p_1, \circ((p_1 \land \lnot\text{now}(\Omega)) \cup p_2)\}.
\]

So that, we can define a model \( M' \) for \( \Sigma' \) as follows

- \( p \in V_{M'}(s'_k) \) iff \( p \in V_M(s_{k+y}) \) for all \( k \in IN \) and all \( p \in \text{Prop} \) such that \( p \neq a \)
- \( a \in V_{M'}(s'_k) \) for every \( k \in \{1, \ldots, x-y-1\} \)
- \( a \notin V_{M'}(s'_k) \) for \( k = 0 \) and every \( k \geq x-y \).

(2) Let \( M \) be a model of \( \Sigma' \), i.e., \( \langle M, s_0 \rangle \models \Sigma' \). Note that \( a \) does not appear in \( \Sigma \). We differentiate two cases. First, if \( \langle M, s_0 \rangle \models p_2 \lor H_i \leftarrow B_i \) for all \( i \in \{1, \ldots, n\} \) then \( M \) is also a model of \( \Sigma \). Second, if \( \langle M, s_0 \rangle \not\models p_2 \lor H_i \leftarrow B_i \) for some \( i \in \{1, \ldots, n\} \), then \( \langle M, s_0 \rangle \models \{p_1, \circ(a \cup p_2)\} \). In such a case, there exists \( j \geq 1 \) such that \( \langle M, s_j \rangle \models p_2 \) and \( \langle M, s_h \rangle \models a \) for every \( h \in \{1, \ldots, j-1\} \). If \( j = 1 \) then \( \langle M, s_0 \rangle \models \{p_1, \circ(p_1 \cup p_2)\} \) and \( M \) is a model of \( \Sigma \). If \( j > 1 \) then the clause \( \Box(p_1 \leftarrow a) \) is necessarily in \( \Psi_2 \) and, consequently, \( \langle M, s_h \rangle \models p_1 \) for every \( h \in \{1, \ldots, j-1\} \). So that, \( \langle M, s_0 \rangle \models \{p_1, \circ(p_1 \cup p_2)\} \) and \( M \) itself is a model of \( \Sigma \).

The proof for the rule \( (\cup C_-) \) is identical to the previous one. For the rules \( (\cap C_+) \) and \( (\cap C_-) \) the proof is obtained straightforwardly by considering that \( p_1 \cap p_2 \) and \( \lnot p_1 \cap p_2 \) in the right-hand side are respectively equivalent to \( \lnot p_1 \cup \lnot p_2 \) and \( p_1 \cup \lnot p_2 \) in the left-hand side. Finally, the derived rules \( (\circ C_+) \) and \( (\Box C_+) \) are particular cases of \( (\cup C_-) \) and \( (\cap C_-) \), respectively.

**Proposition 4.** The rule \( (\cup nx) \) preserves satisfiability.

**Proof.** If \( M \) is a model of \( \Sigma \) then \( \text{unnext}(\Sigma) \) is true in the state \( s_1 \) of \( M \), which obviously gives a model for \( \text{unnext}(\Sigma) \).

Note that, in general, the equi-satisfiability of initial and target sets of the \text{unnext} operator cannot be ensured. For example, in the case of the unsatisfiable set \( \Phi = \{p \leftarrow \top, \perp \leftarrow p, \circ q \leftarrow \top\} \) the set \( \text{unnext}(\Phi) = \{q \leftarrow \top\} \) is satisfiable.

**Theorem 2.** (Soundness) If there exists an IFT-refutation from \( \Pi \) with top-goal \( \Gamma \), then \( \Pi \cup \Gamma \) is unsatisfiable.

**Proof.** If \( \Box \bot (\perp \leftarrow \top) \in \Gamma' \) for some \( (\Pi', \Gamma') \) in an IFT-derivation from \( (\Pi, \Gamma) \), then \( \Pi' \cup \Gamma' \) is unsatisfiable. Therefore, by Propositions 2, 3 and 4, \( \Pi \cup \Gamma \) is also unsatisfiable.
7.2 Systematic IFT-Derivation

In this section we introduce the algorithm $S$ that, for any pair formed by a program $\Pi$ and a goal $\Gamma$, obtains an IFT-derivation (see Definition 1). If $\Pi \cup \Gamma$ is unsatisfiable then $S$ produces an IFT-refutation in a finite number of iterations. We call eventuality atoms to atoms of the form $\neg p$ and $\square p$ when they occur in a head and also to atoms of the form $L \neg p$ and $\square p$ when they occur in a body. Given a program or goal clause $C$ of the form $\square (A_1 \lor \ldots \lor A_m \leftarrow A_1' \lor \ldots \lor A_n')$ with $m \geq 0$ and $n \geq 0$, we denote by atoms$(C)$ the set $\{A_1, \ldots, A_m\} \cup \{A_1', \ldots, A_n'\}$.

Given a pair $(\Pi, \Gamma)$ we denote by atoms$(\Pi, \Gamma)$ the set of all the atoms that occur in $\Pi \cup \Gamma$, i.e., the set $\bigcup_{C \in \Pi \cup \Gamma} \text{atoms}(C)$. We denote by $(\Pi, \Gamma) \mid \{T\}$ the set of all the clauses in $\Pi \cup \Gamma$ that contain the temporal atom $T$ as eventuality atom, i.e., the set $\{C \in \Pi \cup \Gamma \mid T \in \text{atoms}(C)\}$ and $T$ is an eventuality atom in $C$. The subset of atoms$(\Pi, \Gamma)$ formed by all its eventuality atoms is denoted by EventAt$(\Pi, \Gamma)$.

The algorithm $S$ (see Fig. 6) uses, by means of the auxiliary selection function selffun, a marking strategy for applying exactly one context-dependent rule between each two consecutive applications of the rule $(Unx)$. The function selffun associates to every pair $(\Pi_i, \Gamma_i)$ of the resolution derivation the set that contains the eventuality atom that is selected in $(\Pi_i, \Gamma_i)$. Hence the targets of selffun are either the empty set or a singleton. Furthermore, when an eventuality atom $T$ is selected and the corresponding context-dependent rule is applied, the selection of the so-called descendants of $T$ is prioritized in the posterior applications of the context-dependent rules. Due to this marking strategy and due also to the fact that the context-dependent rules are always applied just after the application of the $(Unx)$ rule, the fresh variables introduced by different applications of the context-dependent rules never appear as part of the context when a context-dependent rule is applied. Consequently, the number of possible different contexts is finite and this fact, together with the above mentioned selection strategy, are the keys that warrant the termination of $S$ for any unsatisfiable pair $(\Pi, \Gamma)$.

We say that an eventuality atom $T'$ is a direct descendant of an eventuality atom $T$ in an IFT-derivation $D$ produced by $S$ if selffun$(\Pi_i, \Gamma_i) = \{T\}$ and selffun$(\Pi_j, \Gamma_j) = \{T'\}$ for some $i$ and $j$ where $i \geq 0$, $j > i$, $T \neq T'$ and $(\Pi_{k+1}, \Gamma_{k+1})$ is obtained from

```plaintext
1 D := < (\Pi, \Gamma) >; selffun(\Pi, \Gamma) := \emptyset;
2 while \( \Box^b(\neg \top \leftarrow \top) \notin \text{last}(D) \) loop
3   if selffun(last(D)) \cap EventAt(last(D)) = \emptyset then fair select(D, selffun); end if
4   apply_ctx_depl(D, selffun);
5   apply_ctx_free(D, selffun);
6   apply_goal_res(D, selffun);
7   if \( \Box^b(\neg \top \leftarrow \top) \notin \text{last}(D) \) then apply_unx_res(D, selffun);
8   apply_sbm(D, selffun);
9   apply_unx(D, selffun);
10  end if
11 end loop
```

Fig. 6. The Algorithm $S$
(Π_k, Γ_k) without applying the (Unx) rule for every k ∈ {i, . . . , j − 1}. In other words, the eventuality atom T′ is the direct descendant of the eventuality atom T if ∪T′ is the fresh atom introduced by the application of the context-dependent rule that corresponds to T. The sequence of descendants of T in D, is the longest sequence T_0, T_1, . . . such that T_0 = T and T_{i+1} is a direct descendant of T_i for every i ≥ 0. Additionally, we say that an eventuality atom T′ is a descendant of an eventuality atom T in D if T′ ≠ T and T′ belongs to the sequence of descendants of T in D.

The auxiliary function last gives the last pair of a given derivation.

The construction of the resolution proof for (Π, Γ) is as follows. The algorithm S initializes (see line 1) both the sequence of pairs D and the function selfun. Concretely, D is the sequence formed by the initial pair (Π, Γ) and the function selfun on (Π, Γ) is the empty set, which indicates that none eventuality atom is initially selected. Second, S iterates extending the derivation D with new pairs and stopping only if a refutation is obtained. At the beginning of each iteration step (line 3), one eventuality atom is fixed as selected if there are any available eventuality atoms, i.e., if EventAt(last(D)) ≠ ∅.

In such a case, if selfun(last(D)) ∩ EventAt(last(D)) = ∅, then a new eventuality atom is fairly selected by means of the fair_select function, where fairness means that an eventuality atom cannot be indefinitely unselected. On the contrary, if selfun(last(D)) ∩ EventAt(last(D)) ≠ ∅, the selected eventuality atom, which is the direct descendant of the previous selected eventuality atom in D, remains unchanged. The line 4 has no effect if there is no any selected eventuality atom, since in such a case none of the context-dependent rules will be applied in the current iteration step. Otherwise, the corresponding context-dependent rule is applied considering that Ω = (Π_i ∪ Γ_i) \ (Π_i, Γ_i | selfun(Π_i, Γ_i)) and the value of selfun for the new pair is the singleton that contains the new eventuality atom T (the new atom without ∪) introduced by the applied context-dependent rule, prioritizing in this way the selection of the direct descendant T for the next iteration step. So that, the new eventuality atom T has the highest priority in order to be the selected eventuality atom for the application of a context-dependent rule in the next iteration step of the algorithm, after the application of the rule (Unx). However, T will lose the condition of selected eventuality atom after the application of the rule (Unx) if T does not appear as eventuality atom in the new pair. Then, in line 5, the context-free rules are repeatedly applied while possible. In line 6, the resolution rule is repeatedly applied choosing a clause from the program and a clause from the goal for each application of the rule. We refer to this specific kind of application of the rule (Res) as goal-resolution. The process of applying goal-resolution goes on until either a refutation is obtained or every resolvent that is obtainable by means of goal-resolution is already in the last pair of D. Finally, if D is not a refutation, the resolution rule is repeatedly applied (apply_nx_res) (line 7) by choosing each time a program clause of the form □β(A ∨ H ← B) where A is ∪-free and a program clause of the form □γ (⋄H′ ← A ∧ B′) and applying the resolution rule with respect to the atom A. We refer to this specific kind of application of the rule (Res) as nx-resolution. The program {□(⋄a ← a), a ← ⊤} provides a hint of the need of nx-resolution to get completeness. The process of applying nx-resolution continues until every resolvent that is obtainable by means of nx-resolution is already in the program of the last pair of D. Then the rule
(Sbm) is applied while possible (line 8) and the iteration step ends with the application of the rule (Unx) in line 9.

The value of selfun for each new pair obtained by means of the procedures in S (except for the procedure apply_unx_depred) is inherited from its previous pair in D and in the case of pairs obtained by means of the procedure apply_unx the initially inherited value of selfun can change afterwards by means of the procedure fair_select.

7.3 Completeness

In order to prove that the operational semantics is complete with respect to the logical semantics, we show that the systematic resolution algorithm S in Fig. 6 builds a refutation for any unsatisfiable pair consisting of a program \( \Pi \) and a goal \( \Gamma \). This result is obtained by proving that there exists a model of \( \Pi \cup \Gamma \) whenever an infinite IFT-derivation is built by the algorithm S.

In logic programming, completeness proofs are usually addressed through the immediate consequence operators. In the case of TeDiLog, although a continuous mapping \( T_{\Pi} \) can be associated to every program \( \Pi \) –similarly as in [10]– there are many difficulties for using \( T_{\Pi} \) in a customary completeness proof. The main reason is that structural induction cannot be straightforwardly applied in TeDiLog because the contexts used in \( T_{\Pi}^{n+1}(\emptyset) \) and \( T_{\Pi}^{n}(\emptyset) \) are different. Therefore, the refutability of the goals that correspond to the clauses in \( T_{\Pi}^{n+1}(\emptyset) \) cannot be directly obtained from the refutability of clauses in \( T_{\Pi}^{n}(\emptyset) \) (see e.g. Lemma 4.6 in [2]). This problem is closely related to the problem of syntactical cut elimination that was left open in [3] and [8]. On the contrary, TeDiLog’s completeness result shows that whenever a set of clauses \( \Pi \cup \Gamma \) is unsatisfiable there exists an IFT-refutation for \( (\Pi, \Gamma) \) that is constructed by the algorithm S. More precisely, if the initial set of clauses \( \Pi \cup \Gamma \) is satisfiable, then the algorithm S constructs an infinite derivation, namely \( D_{\Pi \cup \Gamma} \), which allows to define a model of \( \Pi \cup \Gamma \), namely \( M_{\Pi \cup \Gamma} \). To construct \( M_{\Pi \cup \Gamma} \) from \( D_{\Pi \cup \Gamma} \) the main difficulty are eventuality atoms. In particular, we must ensure that satisfaction of eventuality atoms in \( D_{\Pi \cup \Gamma} \) cannot be infinitely delayed in \( M_{\Pi \cup \Gamma} \). The construction of \( M_{\Pi \cup \Gamma} \) is based on the following auxiliary lemma that ensures that clauses containing eventuality atoms can be (finitely) satisfied in \( M_{\Pi \cup \Gamma} \).

It is worth to note that (1) only the context-dependent rules introduce new variables and (2) the eventuality atoms that contain those fresh variables only occur in now-clauses and never occur in always-clauses. More technically, we use the notion of closure of a set of clauses. Let univAt(\( \Pi, \Gamma \)) denote the set of all the atoms that could appear in any IFT-derivation from \( (\Pi, \Gamma) \) without any application of a context-dependent rule. Hence, univAt(\( \Pi, \Gamma \)) is finite and does not contain fresh variables. Then, closure(\( \Pi, \Gamma \)) is defined as the set of all the clauses that can be constructed from univAt(\( \Pi, \Gamma \)). The number of clauses in closure(\( \Pi, \Gamma \)) is \( 2^{O(n)} \) where \( n \) is the size of

---

\(^7\) A practical implementation of TeDiLog requires a slight refinement of the algorithm S. Such refinement, whose details are out of the scope of this paper, is focused on enabling finite derivations for satisfiable inputs whenever no loops are involved.

\(^8\) Under the assumption that the strategy for selecting eventuality atoms is fair in the sense that every eventuality atom is selected at some time.
Due to the way in which the rules presented in Section 4 operate, for every \( \Gamma \) rule (Lemma 1. auxiliary result. p. dnants of univAt)

In addition, the selected eventuality atom in (selected eventuality works when the rule that let \( (p_0 \cup p) \) be an eventuality atom in \( T \) of the form \( U \) be an IFT-derivation for \( (\Pi, \Gamma) \) be an eventuality atom in \( \Pi \cup \Gamma \). In order to prove the completeness theorem we first prove the following

**Lemma 1.** Let \( D_{\Pi \cup \Gamma} \) be an IFT-derivation for \( (\Pi, \Gamma) \) constructed by the algorithm \( S \), and let \( T \) be an eventual atom in \( D_{\Pi \cup \Gamma} \). The sequence of descendants of \( T \) in \( D_{\Pi \cup \Gamma} \) is finite.

**Proof.** Consider any \( T \) of the form \( p_0 \cup p \) such that selfun(\( \Pi_0, \Gamma_0 \)) = \{ \( p_0 \cup p \) \}. Then, the rule \( (U+\) replaces the set \( (\Pi_0, \Gamma_0) \uparrow \{ p_0 \cup p \} \) with the union of the following five disjoint sets of clauses

\[
\begin{align*}
\Psi_0^0 &= \{ p \lor p_0 \lor H_0 \leftarrow B_0 | \Box^{b}(p_0 \cup p \lor H_0 \leftarrow B_0) \in \Pi_0 \} \\
\Psi_0^2 &= \{ p \lor \Box(a_i \cup p) \lor H_0 \leftarrow B_0 | \Box^{b}(p_0 \cup p \lor H_0 \leftarrow B_0) \in \Pi_0 \} \\
\Psi_0^3 &= \{ \Box \lor (p_0 \cup p \lor H_0 \leftarrow B_0) | \Box(p_0 \cup p \lor H_0 \leftarrow B_0) \in \Pi_0 \} \\
\Psi_0^4 &= \{ \Box \lor (p_0 \leftarrow a_1) \} \\
\Psi_0^5 &= \text{deriv}(a_1, p_0, \text{now}(\Pi_0 \cup \Gamma_0) \uparrow (\{ p_0 \cup p \} \cup \{ \Phi_0^4 \}))) \setminus \Phi_0^0
\end{align*}
\]

In addition, the selected eventuality atom in \( (\Pi_1, \Gamma_1) \) is \( a_1 \cup p \). The same reasoning works when the rule \( (U+\) is applied to any \( a_i \cup p \) (i \( \geq 1 \)) that is a descendant of \( p_0 \cup p \), and \( a_{i+1} \cup p \) is obtained as direct descendant. Therefore, a sequence of descendants of \( p_0 \cup p \) of the form \( (a_1 \cup p, a_2 \cup p, \ldots, a_i \cup p, \ldots) \) where for any \( i \geq 1 \) the rule \( (U+\) is applied to a pair \( (\Pi_i, \Gamma_i) \) to obtain a pair \( (\Pi_{i+1}, \Gamma_{i+1}) \) such that \( \Gamma_{i+1} = \Gamma_i \) and \( \Pi_{i+1} \) is the union of the following six disjoint sets

\[
\begin{align*}
\Phi_1^0 &= \{ p \lor a_i \lor H_{j_i} \leftarrow B_{j_i} | \Box^{b}(a_i \cup p \lor H_{j_i} \leftarrow B_{j_i}) \in \Pi_{j_i} \} \\
\Phi_1^2 &= \{ p \lor \Box(a_{i+1} \cup p) \lor H_{j_i} \leftarrow B_{j_i} | \Box^{b}(a_i \cup p \lor H_{j_i} \leftarrow B_{j_i}) \in \Pi_{j_i} \} \\
\Phi_1^3 &= \{ \Box \lor (a_i \cup p \lor H_{j_i} \leftarrow B_{j_i}) | \Box(a_i \cup p \lor H_{j_i} \leftarrow B_{j_i}) \in \Pi_{j_i} \} \\
\Phi_1^4 &= \{ \Box \lor (p_0 \leftarrow a_1), \Box(a_1 \leftarrow a_2), \ldots, \Box(a_{i-1} \leftarrow a_i), \Box(a_i \leftarrow a_{i+1}) \} \\
\Phi_1^5 &= \text{deriv}(a_{i+1}, a_i, \text{now}(\Pi_{j_i}, \Gamma_{j_i}) \uparrow (\{ a_i \cup p \} \cup \{ a_i \cup p \}))) \setminus \Phi_0^4 \\
\Phi_1^6 &= \Pi_{j_i} \setminus \{ \Box^{b}(a_i \cup p \lor H_{j_i} \leftarrow B_{j_i}) \in \Pi_{j_i} \}
\end{align*}
\]

Due to the way in which the rules presented in Section 4 operate, for every \( i \geq 1 \) the selected eventuality \( a_i \cup p \) appears in \( (\Pi_{j_i}, \Gamma_{j_i}) \) only in now-clauses. Hence the set \( \Phi_1^0 \) is empty. Moreover, the literals \( a_i \cup p \) as well as the variables \( a_i \) never appear in the now-clauses in \( (\Pi_{j_h}, \Gamma_{j_h}) \) for all \( h > i \) and all \( \ell \geq 1 \). Consequently now(\( (\Pi_{j_h}, \Gamma_{j_h}) \uparrow (\{ a_i \cup p \} \cup \{ a_i \cup p \} \)) is in closure(\( (\Pi, \Gamma) \) for every \( i \geq 0 \).

Now, let us suppose that the sequence of descendants of \( p_0 \cup p \) is infinite. Since closure(\( (\Pi, \Gamma) \) is finite there must exist two pairs \( (\Pi_{j_h}, \Gamma_{j_h}) \) and \( (\Pi_{j_h}, \Gamma_{j_n}) \) such that

\[
\text{now}(\( (\Pi_{j_h}, \Gamma_{j_n}) \setminus (\{ a_{j_h} \cup p \} \)) = \text{now}(\( (\Pi_{j_h}, \Gamma_{j_n}) \setminus (\{ a_{j_h} \cup p \} \))
\]

Without loss of generality, we consider \( g = 0 \) and \( h = i \). Since

\[
\text{now}(\( (\Pi_{j_h}, \Gamma_{j_i}) \setminus (\{ a_i \cup p \} \)) = \text{now}(\( (\Pi_{j_h}, \Gamma_{j_i}) \setminus (\{ a_i \cup p \} \))
\]

the algorithm \( S \), by repeatedly applying goal-resolution to the clauses in

\[
\text{now}(\( (\Pi \cup \Gamma_0) \setminus (\{ p_0 \cup p \} \)) \cup
\]

19
A sequence of sets of clauses

**Definition 6.**

obtains \( \downarrow \leftarrow a_1 \) which resolves with \( \Box (a_1 \leftarrow a_2) \) producing \( \downarrow \leftarrow a_2 \). Then \( \downarrow \leftarrow a_2 \) resolves with \( \Box (a_2 \leftarrow a_3) \). At the end of this process \( \downarrow \leftarrow a_{i-1} \) resolves with \( \Box (a_{i-1} \leftarrow a_i) \) producing \( \downarrow \leftarrow a_i \). This clause resolves with every clause in

\[
\{ p \lor a_i \lor H_{j_i} \leftarrow B_{j_i}, \ a_i U p \lor H_{j_i} \leftarrow B_{j_i}, \ i \in \Pi_j \}\]

producing the clauses in

\[
\{ p \lor H_{j_i} \leftarrow B_{j_i}, \ a_i U p \lor H_{j_i} \leftarrow B_{j_i}, \ i \in \Pi_j \}\}

which subsume the clauses in

\[
\{ p \lor \circ (a_{i+1} U p) \lor H_{j_i} \leftarrow B_{j_i}, \ a_i U p \lor H_{j_i} \leftarrow B_{j_i}, \ i \in \Pi_j \}\}

Hence, \( a_{j+1} U p \) cannot be the selected eventuality atom in the pair \( (I', \Gamma') \) obtained after the following application of the rule \( (Unx) \), since \( a_{j+1} U p \notin \text{atoms}(C) \) for every clause \( C \in \Pi' \cup \Gamma' \). This is a contradiction because the sequence of descendants for \( p_0 U p \) has been supposed to be infinite.

The other context-dependent rules work similarly to \( (U C_+) \).

In the above proof, we have considered that \( (U C_+) \) is always applied with a non-empty context. The proof for possibly empty contexts is just a special case.

Note that the application of the subsumption rule \( (Sbm) \) and the subsequent use of the rule \( (Unx) \) is essential to prove the above lemma.

The following two notions of *successor* and *fulfilling* are used in the completeness proof we give next.

**Definition 5.** Let \( \Sigma \) and \( \Sigma' \) be two sets of clauses respectively associated to two pairs \( z \) and \( z' \) in an IFT-derivation \( D \), such that \( z \) is a pair obtained just previously to an application of \( (Unx) \) and \( z' \) is the pair obtained just previously to the next application of \( (Unx) \). We say that \( \Sigma' \) is the successor of \( \Sigma \) (and \( \Sigma \) is the predecessor of \( \Sigma' \)) iff they satisfy the following conditions

1. if \( \circ p \leftarrow T \in \Sigma \) then \( p \leftarrow T \in \Sigma' \) and if \( \downarrow \leftarrow \circ p \in \Sigma \) then \( \downarrow \leftarrow p \in \Sigma' \)
2. if \( \circ A \leftarrow T \in \Sigma \) then \( \circ A \leftarrow T \in \Sigma' \) and if \( \downarrow \leftarrow \circ A \in \Sigma \) then \( \downarrow \leftarrow \circ A \in \Sigma' \)
3. if \( \circ (p_1 U p_2) \leftarrow T \in \Sigma \) and \( p_1 U p_2 \notin \text{selfun}(x) \) then \( p_2 \leftarrow T \in \Sigma' \) or \( \{ p_1 \leftarrow T, \circ (p_1 U p_2) \leftarrow T \} \subset \Sigma' \), where \( x \) is the pair in \( D \) obtained from \( z \) using \( (Unx) \)
4. if \( \circ (p_1 U p_2) \leftarrow T \in \Sigma \) and \( p_1 U p_2 \in \text{selfun}(x) \) then \( p_2 \leftarrow T \in \Sigma' \) or \( \{ p_1 \leftarrow T, \circ (a U p_2) \leftarrow T \} \subset \Sigma' \) where \( a U p_2 \in \text{selfun}(z') \) and \( x \) is as in the above condition
5. if \( \downarrow \leftarrow \circ (p_1 U p_2) \in \Sigma \) then \( \downarrow \leftarrow p_2 \in \Sigma' \) or \( \downarrow \leftarrow p_1 \wedge \circ (p_1 U p_2) \in \Sigma' \).

The remaining cases of \( \circ (\neg p_1 U p_2) \leftarrow T \), \( \downarrow \leftarrow \circ (\neg p_1 U p_2) \), \( \circ (L R p) \leftarrow T \), \( \downarrow \leftarrow \circ (L R p) \), \( \circ \circ p \leftarrow T \), \( \downarrow \leftarrow \circ \circ p \leftarrow T \) and \( \downarrow \leftarrow \circ \circ p \) follow straightforwardly from the corresponding equivalences (see figures 2, 4 and 5 for a hint).

**Definition 6.** A sequence of sets of clauses \( \Sigma_0, \Sigma_1, \ldots \) is fulfilling for some \( \Sigma_j \) and some clause \( p_1 U p_2 \leftarrow T \in \Sigma_j \) iff there exists \( k \geq j \) such that \( \Sigma_k \) contains \( p_2 \leftarrow T \) and \( \Sigma_h \) contains \( p_1 \leftarrow T \) for every \( h \in \{ j, \ldots, k - 1 \} \).
The distance \( k - j \) is called the number of steps of that fulfilling assertion. The fulfilling notion is extended to clauses \( \neg p_1 \cup p_2 \leftarrow T, \bot \leftarrow L \neg p, \Diamond p \leftarrow T \) and \( \bot \leftarrow \Box p \) in the obvious manner.

Lemma 1 enables the construction of a model \( \mathcal{M}_{\Pi \cup \Gamma} \) from the infinite derivation \( \mathcal{D}_{\Pi \cup \Gamma} \) that the algorithm \( \mathcal{S} \) produces for any satisfiable pair \((\Pi, \Gamma)\).

**Theorem 3. (Completeness)** For any program \( \Pi \) and any goal \( \Gamma \), if \( \Pi \cup \Gamma \) is unsatisfiable then there exists an IFT-refutation for \((\Pi, \Gamma)\).

**Proof.** Consider \( \Pi \) and \( \Gamma \) such that there does not exist an IFT-refutation for \((\Pi, \Gamma)\). Therefore, the systematic resolution algorithm \( \mathcal{S} \), with input \((\Pi, \Gamma)\), produces an infinite derivation \( \mathcal{D} = (\Pi_0, \Gamma_0), (\Pi_1, \Gamma_1), \ldots \). We are going to show that there exists a model of \( \Pi \cup \Gamma \). We denote by \( S = \Omega_0, \Omega_1, \ldots, \Omega_j, \ldots \) and \( S^* = \Omega^*_0, \Omega^*_1, \ldots, \Omega^*_j, \ldots \) the two infinite sequences of sets such that every \( \Omega_j \) contains all the program and goal clauses that appear in \( \mathcal{D} \) in the pair obtained after the \( j \)th application of the rule \((Unx)\) and every \( \Omega^*_j \) contains all the program and goal clauses that appear in \( \mathcal{D} \) in the pair obtained just before the \( j + 1 \)th application of the rule \((Unx)\). In particular \( \Omega_0 = \Pi_0 \cup \Gamma_0 = \Pi \cup \Gamma \).

Now, let \( \mathcal{G}_\mathcal{D} \) be the set of all the infinite sequences \( \widehat{S}^j = \Omega^*_0, \Omega^*_1, \ldots, \Omega^*_j, \ldots \) such that every \( \Omega^*_j \) is a (minimal) saturation of \( \Omega^*_j \) such that \( \Omega^*_j \) is a minimal (with respect to the subset relation) superset of \( \Omega^*_j \) that satisfies the following five conditions:

1. \( \Omega^*_j \) is locally consistent. That is, there does not exist any IFT-refutation from \( \Omega^*_j \) that does not use the rule \((Unx)\).
2. For every clause \( \Box^b (H \leftarrow B) \in \Omega^*_j \) there exists at least one atom \( A \) in \( H \) such that \( A \leftarrow \top \) is in \( \Omega^*_j \) or there exists at least one atom \( A \) in \( B \) such that \( \bot \leftarrow A \) is in \( \Omega^*_j \).
3. For every \( \Diamond A \leftarrow \top \in \Omega^*_j \) there exists a clause of the form \( \Box^b (A \lor H \leftarrow \Diamond B) \) in \( \Omega^*_j \) and for every \( \bot \leftarrow \Diamond A \in \Omega^*_j \) there exists a clause of the form \( \Box^b (H \leftarrow \Diamond A \land \Diamond B) \) in \( \Omega^*_j \).
4. For every \( j \geq 0 \), the set \( \Omega^*_j+1 \) is a successor of \( \Omega^*_j \).
5. For every \( j \geq 0 \), the sequence \( S^j \) is fulfilling for \( \Omega^*_j \) and for every \( A \leftarrow \top \) and every \( \bot \leftarrow A \) in \( \Omega^*_j \) such that \( A \) is an eventuality atom.

We will prove the following two facts:

1. \( \mathcal{G}_\mathcal{D} \) is non-empty. That is, there exists at least one sequence that belongs to \( \mathcal{G}_\mathcal{D} \).
2. Any sequence in \( \mathcal{G}_\mathcal{D} \) enables to construct a model of \( \Pi \cup \Gamma \).

The fact 1 relies not only on the properties of the sequences in \( \mathcal{G}_\mathcal{D} \) but also on the systematic resolution made by algorithm \( \mathcal{S} \). After the execution of both procedures \( \text{apply}_{\text{ctx}_{\text{dep}}} \) and \( \text{apply}_{\text{ctx}_{\text{free}}} \), every temporal atom is affected by a \( \Diamond \) symbol. In particular, no more decomposition is possible regarding inductive temporal definitions and, hence, the atoms can be seen as propositional atoms. The procedure \( \text{apply}_{\text{goal}_{\text{res}}} \) yields the closure with respect to classical negative resolution. In particular, since \( \mathcal{D} \) is
not a refutation, by completeness of classical resolution, every pair obtained after an execution of the procedure apply\textunderscore goal\textunderscore res is locally consistent. Consequently, for each iteration step \( j \) of the algorithm \( S \), the set \( \Omega_j^\ast \) verifies (i). In general, there are finitely many possibilities to minimally complete \( \Omega_j^\ast \) for satisfying (ii) and (iii) while preserving (i). The existence of extensions of \( \Omega_j^\ast \) that verify (ii) is related to the application of the procedure apply\textunderscore goal\textunderscore res. The existence of extensions of \( \Omega_j^\ast \) that additionally verify (iii) is related to the application of the procedure apply\textunderscore res.

Now, let us consider all the minimal sets \( \Phi_j^1, \Phi_j^2, \ldots, \Phi_j^m \) that contain \( \Omega_j^\ast \) (for some arbitrary \( j \geq 0 \)) and satisfy (i)-(iii). First of all we know that \( m \geq 1 \). Consider also any minimal set \( \Phi_{j+1} \) that contains \( \Omega_{j+1}^\ast \) and satisfies (i)-(iii). Then, at least one of the sets \( \Phi_j^\ell \) (with \( \ell \in \{1, \ldots, m\} \)) must be a predecessor of \( \Phi_{j+1} \) because otherwise every \( \Phi_j^\ell \) contains a clause of the form \( \neg A \leftarrow \top^{9} \) that causes \( \Phi_{j+1} \) not to be a successor of \( \Phi_j^\ell \). In such a case, by classical resolution techniques, it can be proved that there is a clause of the form \( \Box (H \leftarrow \top) \) in \( \Omega_j^\ast \) such that \( H \subseteq \{A_1, \ldots, A_m\} \). On the other hand, by condition (iii), there is a clause \( \Box (A \leftarrow H \leftarrow \neg B) \) in \( \Omega_j^\ast \) and, by construction, there is a clause \( A \lor H \leftarrow B \) in \( \Omega_{j+1}^\ast \) for all \( \ell \in \{1, \ldots, m\} \). By structural induction on \( \neg A \) and considering the cases depicted in the definition of the successor (predecessor) relation between sets of clauses, a contradiction arises.

Summarizing, there exists at least one infinite sequence \( \hat{S}^\ast \) that satisfies (i)-(iv).

Finally, at least one of those infinite sequences \( \hat{S}^\ast = \Omega_0^\ast, \Omega_1^\ast, \ldots, \Omega_j^\ast, \ldots \) that satisfy (i)-(iv) is fulfilling for every \( j \geq 0 \) and every \( p_1 \cup p_2 \leftarrow \top \) in \( \Omega_j^\ast \) (the proof is similar for the remaining cases of unitary clauses with eventuality atom). By fairness, \( p_1 \cup p_2 \) must be selected at some step \( k \). By Lemma 1, the atom \( p_1 \cup p_2 \) has a finite number \( n \) of descendants. Then, by minimality of \( \Omega_j^\ast \), at least one of the sequences\( ^{10} \) prefixed by \( \Omega_0^\ast, \ldots, \Omega_j^\ast \) must be fulfilling for \( \Omega_j^\ast \) and \( p_1 \cup p_2 \leftarrow \top \) in \( n \) steps. Otherwise, \( \Omega_{j+n+1}^\ast \) is not a successor of \( \Omega_j^\ast \), contradicting the fact that for every \( \Omega_h^\ast \) in the sequence \( \hat{S}^\ast \) the set \( \Omega_{h+1}^\ast \) is a successor of \( \Omega_h^\ast \). This ensures that \( G_D \) is non-empty.

Finally, to prove the above fact 2, let \( \hat{S}^\ast = \Omega_0^\ast, \Omega_1^\ast, \ldots, \Omega_j^\ast, \ldots, \) be any sequence in \( G_D \). We define the PLTL-structure \( M = (\hat{S}^\ast, V_M) \) such that for every \( i \geq 0 \)

\[
V_M(\hat{\Omega}_i^\ast) = \{p \in \text{Prop} | p \leftarrow \top \in \hat{\Omega}_i^\ast\}.
\]

It is routine to see that \( \langle M, \hat{\Omega}^\ast \rangle \models \Box (H \leftarrow B) \) holds for all \( \Box (H \leftarrow B) \in \hat{\Omega}_i^\ast \). Since any \( \hat{\Omega}_i^\ast \) contains at least one clause \( C = A \leftarrow \top \) such that \( A \in H \) or one clause \( C = \bot \leftarrow A \) such that \( A \in B \), this is made by structural induction on the form of \( A \), using the Definition 5 and the fact that the sequence \( \hat{S}^\ast \) is fulfilling for \( \hat{\Omega}_i^\ast \) and \( C \) (by the above proved fact 1 (v)) whenever \( A \) is an eventuality atom. In particular, \( M \) is a model of \( \Omega_0^\ast \). Hence, since \( \Pi \cup \Gamma = \Omega_0^\ast \), the set of clauses \( \Pi \cup \Gamma \) is satisfiable.  

\footnote{or of the form \( \bot \leftarrow \neg A \), whose proof is very similar.}

\footnote{which satisfies (i)-(iv).}
8 Conclusions

TeDiLog is a very expressive temporal logic programming language with a purely declarative nature and mathematically defined semantics. The operational semantics of TeDiLog is based on a resolution mechanism that is powerful enough to deal with eventualities and dispenses with invariant generation. The latter is a crucial difference of our method with respect to the clausal resolution method introduced in [4] (see also [5]) which needs to generate invariant formulas for solving eventualities. The other main difference is that the clausal normal form proposed in [4] is not an extension of Horn clauses and it is very restricted for use as a form of program clause in a TLP language. We see TeDiLog as the propositional kernel of a new generation of temporal logic languages based on the so-called invariant-free temporal resolution. In this sense we hope that TeDiLog could influence the design of future temporal logic programming languages for incorporating more expressive temporal features and new resolution procedures for temporal reasoning.

References