Logical Foundations for More Expressive Declarative Temporal Logic Programming Languages

JOSE GAINTZARAIN and PAQUI LUCIO, University of the Basque Country

In this paper, we present a declarative propositional temporal logic programming language called TeDiLog that is a combination of the temporal and disjunctive paradigms in Logic Programming. TeDiLog is, syntactically, a sublanguage of the well-known Propositional Linear-time Temporal Logic (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 declarative temporal logic programming language that achieves this high degree of expressiveness. We establish the logical foundations of our proposal by formally defining operational and logical semantics for TeDiLog and by proving their equivalence. The operational semantics of TeDiLog relies on a restriction of the *invariant-free temporal resolution* procedure for PLTL that was introduced by Gaintzarain et al. in 2013. We define a fixpoint semantics that captures the reverse (bottom-up) operational mechanism and prove its equivalence with the logical semantics. We also provide illustrative examples and comparison with other proposals.

Categories and Subject Descriptors: D.1.6 [**Programming Techniques**]: Logic Programming; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*; *Syntax*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Temporal Logic*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Logic Programming*; *Resolution*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Disjunctive Logic Programming, Invariant-Free Clausal Temporal Resolution, Linear-time Temporal Logic, Operational and Logical Semantics, Refutation Procedure, Temporal Logic Programming

ACM Reference Format:
Jose Gaintzarain, and Paqui Lucio, 2013. Logical Foundations for More Expressive Declarative Temporal Logic Programming Languages. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 39 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

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. For a recent and extensive monograph on temporal logic techniques and tools, we refer to [Fisher 2011]. Temporal Logic Programming (TLP) deals with the direct execution of temporal logic formulas. Hence TLP provides a single framework in which dynamic systems can be specified, developed, validated and verified by means of executable specifications that make possible to prototype, debug and improve systems before their final use. A different approach for the validation of dynamic systems is model checking [Clarke et al. 1986; Clarke et al. 2001]. Model checking focuses on the problem of deciding whether a concrete model

This work has been partially supported by Spanish Project TIN2007-66523, and by the University of the Basque Country under Project LoRea GIU07/35 and grant UFI11/45.
Author's addresses: J. Gaintzarain, Department of Computer Languages and Systems, University of the Basque Country, EUITI de Bilbao, Paseo Rafael Moreno Pitxitxi 3, 48013-Bilbao, Spain; P. Lucio, Department of Computer Languages and Systems, University of the Basque Country, Facultad de Informática, Paseo Manuel de Lardizábal 1, 20018-San Sebastián, Spain.

(or run) of a system satisfies a logical formula or not. In model checking, temporal logic is used for specification purposes, whereas the system is often implemented in a different language, hence verification requires to manage different semantic domains. A lot of work has been carried out in the field of model checking. This approach is reasonably efficient for finite state systems. The interested reader is referred to [Fisher 2011] (Section 4.4.7 and Chapter 5) for a recent work that describes model checking techniques. A brief and clarifying discussion about model checking versus deductive temporal verification can be found in [Dixon et al. 2006].
In TLP, the direct execution of a formula corresponds to building a model for that formula. The idea of directly executing logic formulas has been thoroughly studied in (classical) Logic Programming (LP). Given a program Π, the computation of a goal $\bot \leftarrow \gamma$ with respect to Π in an LP system is a search for a refutation proof of $\Pi \cup \{\bot \leftarrow \gamma\}$.[1] However, this proof search can also be seen as an attempt to build a model of $\Pi \cup \{\gamma\}$. This model is (in general) partially specified, because it only determines the truth value of the atoms (from Π) that are involved in the refutation proof. We illustrate this view (of LP) in the next example.

Example 1.1. Let us consider the following (classical) logic program:[2]

$$\begin{array}{l} q(0) \leftarrow \top \\ q(X) \leftarrow q(Y) \wedge X = Y + 1 \\ r(X) \leftarrow q(Y) \wedge X = Y + 2 \\ w(X) \leftarrow q(Y) \wedge X = Y + 3 \end{array}$$

The computation of the goal $\bot \leftarrow r(Z)$ gives rise to the infinite sequence of answer substitutions $\{Z \leftarrow 2\}, \{Z \leftarrow 3\}, \{Z \leftarrow 4\}, \dots$ that partially shows the implicit step by step construction of the infinite minimal model $\{q(j), r(j+2) \mid j \geq 0\}$ for the body of the goal (i.e. $r(Z)$) and the subprogram that contains the first three program clauses. However, this model does not specify which instances of $w(X)$ are true. ■

TLP, in a broad sense, means programming in any language based on temporal logic. In TLP two different approaches have arisen: the *imperative future* approach and the *declarative* approach. In the imperative future approach a program is a set of rules of the form $\varphi \rightarrow \circ\psi$ asserting that whenever the formula φ is true in a state s, the next state s' satisfies the formula ψ. Using a forward chaining process, the imperative future approach tries to construct a model of the whole input program. By contrast, the declarative approach to TLP is based on extending classical resolution for dealing with temporal connectives. Hence the (implicit) attempt of constructing a model is driven by the goal. As the above Example 1.1 shows, such model determines only the predicates involved in the refutational process.

It is well known that one of the features of temporal logic is the ability to express eventualities and invariants. An eventuality is a formula that asserts that something does eventually hold. For example, to fulfill the formula $\varphi \mathcal{U} \psi$, the formula ψ must eventually be satisfied. Invariants state that a property will always be true (from some moment onwards). Syntactically, eventualities are easily detectable but invariants can be expressed in intricate ways by means of loops. Consequently, we say that invariants can be "hidden". Since a "hidden" invariant can prevent the fulfillment of an eventuality, the way in which the issue of eventualities and "hidden" invariants is dealt with becomes a relevant characteristic of TLP languages. The use of the customary inductive definitions of the temporal connectives as the only mechanism for detecting the existence of an invariant that prevents the fulfillment of an eventuality, leads to incompleteness. The reason is that such customary inductive definitions make possible to indefinitely postpone the fulfillment of an eventuality and, consequently, they make possible to indefinitely postpone the contradiction between an eventuality that states that a property ψ will eventually hold and an invariant that states that ψ will never hold. Therefore, more elaborated mechanisms are needed.

[1] $\bot$ stands for the empty head (equivalent to falsehood).
[2] $\top$ stands for the empty body (equivalent to truth).

Next, we briefly review the most significant proposals in the literature for both the imperative future approach and the declarative approach. More discussion and references about programming languages with capabilities for reasoning about time can be found e.g. in [Gergatsoulis 2001; Fisher 2011; Orgun 1994; Orgun and Ma 1994].

Imperative future TLP languages. The most significant representatives of this approach are Tempura [Moszkowski 1986] and MetateM [Barringer et al. 1989]. The language Tempura is based on a fragment of Interval Temporal Logic with a restricted use of eventualities. The Tempura approach has been continued ([Cau et al. 1996; Moszkowski 1998]) and extended to Framed Tempura and Projection Temporal Logic Programming [Duan et al. 2005; 2008; Yang and Duan 2008; Yang et al. 2010]. However, regarding eventualities, these extensions keep the same limitations as the original Tempura. The language MetateM develops the methodology outlined in [Gabbay 1987a]. MetateM is based on First-order Linear-time Temporal Logic (FLTL) and formulas are written in the Separated Normal Form (SNF) presented in [Fisher 1991; 1992]. The propositional fragment of MetateM is complete, however, since FLTL is incomplete ([Merz 1992; Szalas and Holenderski 1988; Szalas 1995]), the execution of a first-order MetateM program attempts to build a model, but the success of such construction is not guaranteed (see Example 1.4). In MetateM disjunctions are seen as choices and one disjunct is selected from each disjunction as part of the process of building a model. If a choice is later shown to be inappropriate, because it leads to inconsistency, then backtracking is used to return to the last point where a choice was made. In propositional MetateM the termination is addressed by explicitly considering the finite model property, which allows to calculate an upper bound of forward chaining steps. If a model is not obtained bellow this upper bound, then the attempt is given up and the procedure backtracks. MetateM was extended to Concurrent MetateM in [Fisher 1993]. Among its applications we can mention, e.g., the development of agent systems ([Fisher 1997; Fisher and Ghidini 2010]). More references on MetateM, Concurrent MetateM and their applications can be found in [Fisher 2011]. A fragment of Linear-time Temporal logic is presented as imperative future TLP language in [Merz 1995]. This language, for efficiency, restricts the use of eventualities (and also disjunctions). The clausal normal form and the idea of forward chaining construction of models introduced in MetateM are used in [Aguado et al. 2008; Aguado et al. 2011] to obtain a temporal extension of the Answer Set Programming paradigm (non-monotonic reasoning).
Finally, we also mention the assembly-like TLP language XYZ/E that was presented in [Tang 1983] as a vehicle for providing temporal semantics to programs written in conventional imperative programming languages. An imperative program is expressed in XYZ/E on the basis of the execution sequences that it generates along the timeline. A similar approach can be found in Chapter 3 of [Fisher 2011].

Declarative TLP languages. There are several works on extending classical LP (in particular Prolog) for reasoning about time. Some proposals are purely based on temporal logic and extensions of SLD resolution, but the incompleteness of FLTL becomes a delicate issue for using fragments of FLTL as TLP languages. Also the complexity result is a drawback even for the propositional fragment (see [Sistla and Clarke 1985]). Additionally, the interaction between the $\Box$ ("always") and the $\circ$ ("next") connectives makes possible to encode the so-called *induction on time* by means of loops that, in an indirect way, state that a formula is satisfied in every moment in time. The presence of these loops –"hidden" invariants– makes necessary to consider quite intricate mechanisms for detecting (un)satisfiable eventualities. Many temporal extensions of LP are not purely founded on temporal logic due to their extra-logical features for handling eventualities. Next, we summarize representative published work concerning the variety of proposals in declarative TLP languages (including some approaches that are not purely based on temporal logic).
The language Tokio [Fujita et al. 1986; Kono 1995; Kono et al. 1985; Nakamura et al. 1989] extends Prolog by adding temporal reasoning capabilities inspired by both Linear-time Temporal Logic and Interval Temporal Logic. In Tokio there are restrictions regarding the use of temporal connectives and, unlike Prolog variables, the so-called temporal variables used in Tokio have state,

what makes possible to express properties like $\circ Y = Y + 1$ stating that the value of the variable Y in the next time instant will be its present value plus one. Obviously, this kind of expressions are no supported by conventional Temporal Logic.

A different temporal extension of Prolog was introduced by Hrycej in [Hrycej 1988; 1993] where time intervals are considered as conceptual primitives. Hrycej's language is a non-modal approach based on first-order logic with capabilities to deal with time intervals. More precisely, the first-order "reified" logic ([Reichgelt 1987; Shoham 1986]) is considered as the basis for the implementation of the language.

Metric temporal operators and dense time are considered in [Brzoska 1991; 1993; 1995b; 1995a; 1998; Brzoska and Schäfer 1995] where execution is based on translating temporal logic programs into Constraint Logic Programming. Temporal Annotated Constraint Logic Programming is presented in [Frühwirth 1995; 1994; 1996; Raffaetà and Frühwirth 1999].

The Temporal Prolog presented in [Sakuragawa 1986] extends Prolog by introducing linear-time temporal connectives. Programs are transformed into a normal form that is similar to the Separated Normal Form used in MetateM. This transformation removes most temporal connectives by introducing fresh predicates. The transformation of eventualities yields negated atoms. If negated atoms (i.e., eventualities) are involved in a program, then the Herbrand universe must be finite and, in this case, computation is performed on the basis of a nondeterministic finite automaton that corresponds to the program. Two implementation options are devised: first, by translating programs into Prolog (if the program contains negation, then a pure Prolog program is not obtained) and second, asserting the facts which are true at each point in time (although this implementation option is not explained in detail, it resembles, at first sight, the imperative future approach).

A sequent-based proposal for establishing logical foundation for declarative TLP is presented in [Pliuskevicius 1992]. This approach considers a complete fragment of FLTL where eventualities are allowed. In order to handle eventualities, the sequent system contains an invariant-based rule.

We finally review the three existing declarative TLP languages that are based on pure extensions of classical logic programming languages and resolution, which are Chronolog [Wadge 1988; Orgun 1991; 1995], Templog [Abadi and Manna 1987; 1989; Baudinet 1988; 1989a; 1989b; 1992; 1995] and Gabbay's Temporal Prolog [Gabbay 1987b]. Chronolog and Templog are the most studied and the most representative languages in the purely declarative approach. The underlying logic for the languages Templog and Chronolog is FLTL. In the case of Gabbay's Temporal Prolog, the presented system is intended for both branching-time and linear-time temporal logic. In Chronolog, the connectives first (to refer to the initial state s_0) and next (to refer to the next state with respect to the current one) are the only temporal connectives. Templog's syntax allows the always connective ($\Box$) to occur in clause heads and the eventually connective ($\Diamond$) in clause bodies. Additionally, the next time connective $\circ$ is allowed in the atoms and the connective $\Box$ can prefix an entire clause. However, Templog programs are expressible by using $\circ$ as the unique temporal connective ([Baudinet 1989b]) and consequently it has the same expressive power as Chronolog. This restriction is so strong that it allows reducing any temporal program to a (possibly infinite) classical logic program. Templog and Chronolog have also the same metalogical properties of existence of minimal model and fixpoint characterization. Gabbay's Temporal Prolog is a more expressive language that allows eventualities in clause heads (although it does not allow $\Box$ in clause bodies). The resolution-based computation procedure outlined in [Gabbay 1987b] is proved to be sound, however its completeness has not been addressed. The development of these three declarative languages was mainly done in the early nineties, in contrast to the imperative future approach (e.g. Tempura and MetateM) which has been evolving until present days. During the last two decades, no other clausal sublanguage of linear-time temporal logic has been proposed as declarative TLP language. Hence, nowadays, Templog, Chronolog and Gabbay's Temporal Prolog remain as the most expressive proposals of declarative TLP languages. Later extensions of Chronolog (e.g. [Orgun et al. 1993; Orgun and Wadge 1994; Rondogiannis et al. 1997; 1998; Gergatsoulis et al. 2000]) did not add significant temporal expressiveness. In the case of Gabbay's Temporal Prolog, although the expressive power was considerably high, it seems that the lack of completeness was a handicap for further study and development.

In general, it seems that the troublesome solving (in the resolution sense) of the eventualities (whose fulfillment can be prevented by "hidden" invariants) has been blocking the steps toward more expressive resolution-based declarative TLP languages. Indeed, even in the propositional fragment –i.e. in PLTL– the solving of eventualities is the most intricate part that often requires techniques such as invariant generation ([Fisher 1991; Fisher et al. 2001]). In this paper, we contribute to the effort of increasing the temporal expressiveness of declarative TLP languages on the basis of a new temporal resolution-based mechanism (see [Gaintzarain et al. 2013]) that is complete (in the propositional setting). The main novelty of this temporal resolution lies in a new approach to handle eventualities. We introduce a purely declarative propositional TLP language, called TeDiLog, that allows both $\Box$ and $\Diamond$ in clause heads and bodies. Hence, TeDiLog is strictly more expressive than the propositional fragments of the above mentioned purely declarative proposals: Templog [Abadi and Manna 1989; Baudinet 1989b], Chronolog [Wadge 1988; Orgun 1995] and Gabbay's Temporal Prolog [Gabbay 1987b]. Additionally TeDiLog is as expressive as propositional MetateM [Barringer et al. 1989]. From the operational point of view, MetateM follows the imperative future approach, i.e. it is not based on resolution. For deciding whether an eventuality is satisfied, the MetateM procedure performs backtracking whenever the upper bound of the size of the (hypothetical) model is exceeded. However, the resolution mechanism of TeDiLog directly manages unsatisfiable eventualities. Temporal logic can be seen as an instance of modal logic where the set of possible worlds represents a collection of moments in time. The earliest proposals on Modal Logic Programming (MLP) come from the mid and late 1980's (see e.g. [Balbiani et al. 1988; Fariñas del Cerro 1986]). The most expressive MLP language is MProlog ([Nguyen 2000; 2003; 2006; 2009]) which is as expressive as the general modal Horn fragment. More references on MLP can be found in [Gergatsoulis 2001; Orgun 1994; Orgun and Ma 1994; Nguyen 2003; 2009]. The main concern in the operational semantics of TeDiLog is the problem of eventualities and "hidden" invariants, which is a specific TLP problem that does not appear in MLP. This makes TeDiLog to be very far from any MLP proposal. However, results and advances in MLP can help, to some extent, in the achievement of new results for TLP. In particular, MProlog is based on extending classical resolution to the modal framework. In [Nguyen 2003], the fixpoint semantics for MProlog is first introduced and then the SLD-Resolution calculus is defined as the reverse operation of the fixpoint operator. We have been inspired by this relation between the operational and the fixpoint semantics to define a fixpoint semantics for TeDiLog.

Along the paper, we compare TeDiLog with its most closely related proposals: Templog, Chronolog, the linear-time Gabbay's Temporal Prolog and MetateM. The technical content of this paper is focused on the propositional language TeDiLog. However, for a better illustration of the aim of our proposal, we next discuss some first-order program examples. They are written in the natural extension of TeDiLog with predicates and variables.

Example 1.2. Consider the following program (on Fibonacci numbers):

$$\begin{array}{l} fib(0) \leftarrow \top \\ \circ fib(1) \leftarrow \top \\ \Box\,(\circ^2 fib(V) \leftarrow fib(X) \wedge \circ fib(Y) \wedge V = X + Y) \end{array}$$

The goal $\bot \leftarrow \circ^3 fib(Z)$ yields the answer substitution $\{Z \leftarrow 2\}$. The goal $\bot \leftarrow \Diamond fib(Z)$ produces an infinite sequence of answer substitutions $\{Z \leftarrow 0\}$, $\{Z \leftarrow 1\}$, $\{Z \leftarrow 1\}$, $\{Z \leftarrow 2\}$, ..., that is, the sequence of Fibonacci numbers. Now, consider the goal $\bot \leftarrow \Box fib(Z)$ which is not expressible in Templog, Chronolog and Gabbay's Temporal Prolog. The TeDiLog computation does not finish and does not produce any answer. Note that $\Box fib(j)$ is not a logical consequence of the program for any $j \geq 0$.
The above program is expressible in MetateM through a simple transformation. The

MetateM program execution, which does not need a goal, builds the infinite model $\{fib(0), \circ fib(1), \circ^2 fib(1), \circ^3 fib(2), \ldots\}$. ■

Example 1.3. The following program encodes the so-called induction on time (for $q(a)$):

$$\begin{array}{l} q(a) \leftarrow \top \\ \Box\,(\circ q(X) \leftarrow q(X)) \end{array}$$

Hence, $q(a)$ is true at every instant along the time. The goal $\bot \leftarrow \circ^3 q(Z)$ yields the answer substitution $\{Z \leftarrow a\}$. The goal $\bot \leftarrow \Diamond\, q(Z)$ generates the infinite sequence of answer substitutions $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, $\{Z \leftarrow a\}$, ... The goal $\bot \leftarrow \Box\, q(Z)$ also yields the answer substitution $\{Z \leftarrow a\}$. The latter goal is not expressible in Templog, Chronolog and Gabbay's Temporal Prolog. The MetateM system builds the infinite model $\{q(a), \circ q(a), \circ^2 q(a), \ldots\}$ for the above program. ■

Example 1.4. The following program shows that, as expected, the natural first-order extension of TeDiLog gives rise to an incomplete system:

$$\begin{array}{l} q(0) \leftarrow \top \\ \Box\,(\circ q(X) \leftarrow q(X)) \\ \Box\,(\circ q(X) \leftarrow q(Y) \wedge X = Y + 1) \\ \Box\,(w(X) \leftarrow \Box\, q(X)) \end{array}$$

This fact is due to the interaction between the infinite domain and the connective $\Box$ in the body of the last clause. By means of the first three clauses, for every $i \geq 0$, $q(i)$ holds in all states s_j such that $j \geq i$. As a consequence, $w(i)$ holds in a state s_j if $i \geq j$. Indeed, the atoms $w(0), \circ w(0), \circ w(1), \circ^2 w(0), \circ^2 w(1), \circ^2 w(2), \ldots$ are logical consequences of the program. However, the first-order extension of our resolution method will not yield any answer either for the goal $\bot \leftarrow \Diamond\, w(Z)$ or for any goal $\bot \leftarrow \circ^k w(Z)$ where $k \geq 0$. The reason is that, by contrast with the previous Example 1.3, here the goal $\bot \leftarrow \Box\, q(V)$ does not give any answer (due to the infinite domain), and consequently the last program clause cannot be used to produce $w(V)$.
In order to obtain a MetateM program, the last program clause above is translated into SNF giving rise to two clauses: $\Box\,(\circ r(X) \leftarrow q(X) \wedge \neg w(X))$ and $\Box\,(\Diamond \neg q(X) \leftarrow r(X))$, where r is a fresh predicate symbol. Consequently MetateM attempts to construct a model for the following program[3]:

$$\begin{array}{l} q(0) \leftarrow \top \\ \Box\,(\circ q(X) \leftarrow q(X)) \\ \Box\,(\circ q(X) \leftarrow q(Y) \wedge X = Y + 1) \\ \Box\,(\circ r(X) \leftarrow q(X) \wedge \neg w(X)) \\ \Box\,(\Diamond \neg q(X) \leftarrow r(X)) \end{array}$$

Then, the atoms in $\{q(0), \circ q(0), \circ q(1), \circ^2 q(0), \circ^2 q(1), \circ^2 q(2), \ldots\}$ are successively obtained. In addition, since there is no clause with head $w(Z)$, we can suppose that $\neg w(X)$ succeeds in a time instant for any X such that $q(X)$ is true at that time instant. Therefore, the atoms

$$\{\circ r(0), \circ^2 r(0), \circ^2 r(1), \circ^3 r(0), \circ^3 r(1), \circ^3 r(2), \ldots\}$$

are also generated. According to the last program clause, the system attempts to satisfy $\Diamond \neg q(X)$, however at each step the system must delay this task for the next step. Therefore, MetateM (as TeDiLog) is not able to generate a model for this program. ■

In the rest of the paper we restrict ourselves to the propositional setting. Hence, the logic that underlies TeDiLog is the well-known Propositional Linear-time Temporal Logic (PLTL), which

[3]Actually this program is not in pure SNF yet (see e.g. [Fisher 1992]). Some minor syntactical changes are still needed, but they are irrelevant for our discussion.

is complete and decidable. We endow TeDiLog with logical and operational semantics and prove their equivalence. The logical semantics is given by the set of all the (finite) formulas of the form $\alpha_1 \vee \ldots \vee \alpha_n$ that are logical consequences (in PLTL) of the program and where each α_j is either a body or a body prefixed by the connective $\diamond$. The operational semantics of TeDiLog is based on the *invariant-free resolution method* that is presented in detail in [Gaintzarain et al. 2013] and dispenses with invariant generation. [4] 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 non-conjunctive temporal connective $\diamond$ appearing in clause heads, and also the non-finitary connective $\Box$ appearing in clause bodies, both (separately) prevent from holding the MMP (see [Orgun and Wadge 1992]). For Gabbay's Temporal Prolog the MMP does not hold because of the use of the connective $\diamond$ in clause heads. The second reason is that 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. In the DLP framework, the semantics of a program consists of the collection of all its minimal models (see e.g. [Lobo et al. 1992]). Temporal disjunctive logic programming has previously been addressed in [Gergatsoulis et al. 2000] where Chronolog is extended with DLP features. The satisfiability of a Templog/Chronolog program can be reduced to the satisfiability of a classical logic program. As a consequence the minimal model characterization of Templog and Chronolog (see [Baudinet 1989b; Gergatsoulis et al. 2000; Wadge 1988; Orgun 1995]) is a straightforward adaptation of the classical (disjunctive) case. We define a fixpoint semantics (inspired by [Nguyen 2003]) by associating an operator T_Π to any TeDiLog program Π. However, there is a great difficulty for using T_Π in the customary way for proving the equivalence between the operational and the fixpoint semantics. Concretely, the context handling mechanism (which is the basis of our operational semantics) prevents us from obtaining a refutation of $T^n_\Pi(\emptyset)$ from a refutation of $T^{n-1}_\Pi(\emptyset)$. [5] Indeed, this difficulty is closely related to the problem of syntactical cut elimination for PLTL, which is an open problem (see [Brünnler and Lange 2008] and [Gaintzarain et al. 2009]). As a consequence, in this paper, we prove, first, the equivalence between operational and logical semantics and, second, the equivalence between fixpoint and logical semantics.

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* $\mathcal{U}$. Hence, we directly formulate our language in terms of the temporal connectives $\mathcal{U}$ and its dual: the *release connective* $\mathcal{R}$. We present a notion of derivation, called IFT-derivation, of a goal with respect to a program. IFT-derivations are based on a natural extension of the classical LP rule for (binary) resolution in two senses: temporal ($\Box$ in front of clauses) and disjunctive (disjunction in clause heads). Additionally, resolution application is restricted not only to the standard (linear) resolution between the current goal and a selected program clause, but also to a controlled kind of resolution called *next-resolution*. This next-resolution is performed to infer (from program clauses) all the (program) clauses that have a $\circ$ connective in front of every literal. Intuitively, next-resolution allows to extract all the implicit information about the next state that is crucial to achieve completeness. We also define a fixpoint semantics that captures the reverse (bottom-up) of the operational mechanism given by IFT-derivations. We prove the equivalence between the operational, logical and fixpoint semantics.

Outline of the paper. In Section 2 we provide the basic background on PLTL. In Section 3 we introduce the syntax of TeDiLog, some preliminary definitions and a sample TeDiLog specification of a reactive system. In Section 4 we present the system of rules that are the basis for the operational semantics of TeDiLog. Section 5 is devoted to the operational, logical and fixpoint semantics and their equivalence. In Subsection 5.1 we present the operational semantics of TeDiLog. Then, in

[4]The basis of this *deduction method* is presented in [Gaintzarain et al. 2009], where sound and complete methods of tableaux and sequents are defined for full PLTL.
[5]This customary technique can be found in e.g. Lemma 4.6 in [Baudinet 1989b] and Lemma 7.8 in [Nguyen 2003].

Subsection 5.2 we detail some sample derivations. The logical semantics is described in Subsection 5.3. We prove the equivalence between operational and logical semantics in Subsection 5.4. In Subsection 5.5 we introduce a fixpoint semantics and prove its equivalence with the logical semantics. In Section 6 we compare in more detail TeDiLog with the most similar proposals in the literature, though some puntual comparisons have been previously made. Finally, we summarize our contribution in Section 7.

A very preliminary version of this work was presented at the Spanish Workshop PROLE 2009 (see [Gaintzarain and Lucio 2009]).

2. PRELIMINARIES AND NOTATION

PLTL-formulas are built using propositional variables ($p, q, \ldots$) from a set Prop, the classical connectives, and the temporal connectives $\circ$ ("next") and $\mathcal{U}$ ("until"). In the sequel, *formula* means PLTL-formula. A lowercase Greek letter ($\varphi, \psi, \chi, \gamma, \ldots$) denotes a formula and an uppercase one ($\Phi, \Delta, \Gamma, \Psi, \Omega, \ldots$) denotes a finite set of formulas. 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 Φ, i.e., $\neg\Phi = \neg\varphi_1 \vee \ldots \vee \neg\varphi_n$.
A PLTL-*structure* $\mathcal{M}$ is a pair $(S_\mathcal{M}, V_\mathcal{M})$ where $S_\mathcal{M}$ is a denumerable sequence of states $s_0, s_1, s_2, \ldots$ and $V_\mathcal{M} : S_\mathcal{M} \rightarrow 2^{\mathsf{Prop}}$ is a mapping that specifies which atomic propositions are true in each state in $S_\mathcal{M}$. Formulas are interpreted in the states of PLTL-structures. The formal semantics of formulas is given by the truth of a formula φ in a state s_j of a PLTL-structure $\mathcal{M}$, which is denoted by $\langle \mathcal{M}, s_j \rangle \models \varphi$. This semantics 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 $\langle \mathcal{M}, s_j \rangle \models \psi$
— $\langle \mathcal{M}, s_j \rangle \models \varphi \vee \psi$ iff $\langle \mathcal{M}, s_j \rangle \models \varphi$ or $\langle \mathcal{M}, s_j \rangle \models \psi$
— $\langle \mathcal{M}, s_j \rangle \models \circ\varphi$ 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 $\langle \mathcal{M}, s_i \rangle \models \varphi$ holds for every $i \in \{j, \ldots, k-1\}$.

The remaining linear-time connectives that appear in this paper, i.e., $\mathcal{R}$ ("release"), $\diamond$ ("eventually") and $\Box$ ("always"), can be defined in terms of the connective $\mathcal{U}$. In particular, $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$, $\diamond \varphi \equiv \neg\varphi \mathcal{U} \varphi$, $\Box \varphi \equiv \neg\diamond\neg\varphi$. Note that $\Box \varphi \equiv \neg\varphi \mathcal{R} \varphi$. The extension of the above formal semantics to the derived connectives yields:

— $\langle \mathcal{M}, s_j \rangle \models \varphi \mathcal{R} \psi$ iff for every $k \geq j$ it holds either $\langle \mathcal{M}, s_k \rangle \models \psi$ or $\langle \mathcal{M}, s_i \rangle \models \varphi$ for some i such that $j \leq i < k$
— $\langle \mathcal{M}, s_j \rangle \models \diamond \varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for some $k \geq j$
— $\langle \mathcal{M}, s_j \rangle \models \Box \varphi$ iff $\langle \mathcal{M}, s_k \rangle \models \varphi$ for every $k \geq j$.

For a set of formulas Φ, we say that $\langle \mathcal{M}, s_j \rangle \models \Phi$ 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 $\langle \mathcal{M}, s_j \rangle \models \chi$.

Note that the satisfaction of $\varphi \mathcal{U} \psi$ and $\diamond \psi$ requires that ψ must eventually be satisfied, and also that the eventual satisfaction of $\neg\psi$ is required to satisfy $\neg\Box \psi$ and $\neg(\varphi \mathcal{R} \psi)$. Consequently

Definition 2.1. An *eventuality* is a formula of the form $\varphi \mathcal{U} \psi$ or $\diamond \psi$ or $\neg\Box \psi$ or $\neg(\varphi \mathcal{R} \psi)$.

Definition 2.2. A formula χ is an *invariant* if and only if the formula $\neg\chi \vee \circ\chi$ is true at every state of every PLTL-structure.

In order to illustrate that an invariant χ may assert, often in an intricate way, that some formula ψ is always true, let us consider some examples. The formulas $\chi_1 = \Box\,\psi$, $\chi_2 = \psi \wedge \Box\,(\neg\psi \vee \circ\psi)$ and $\chi_3 = \psi \wedge \Box\,(\neg\psi \vee \circ\varphi) \wedge \Box\,(\neg\varphi \vee \psi)$ are three invariants. The formula χ_1 is the easiest way of stating that ψ is always true. However χ_2 and χ_3 also assert that ψ is always true, since $\Box\,\psi$ is a logical consequence of both $\{\chi_2\}$ and $\{\chi_3\}$. It is worth noting that loops like the one in χ_3 can be quite complex. Let us consider, for instance, the following three sets of formulas

$$\begin{aligned}
\Delta_1 &= \{\Box\,(\neg\varphi_0 \vee \circ\psi_0), \ldots, \Box\,(\neg\varphi_n \vee \circ\psi_n)\}\\
\Delta_2 &= \{\Box\,(\neg\psi_0 \vee \neg\gamma), \ldots, \Box\,(\neg\psi_n \vee \neg\gamma)\}\\
\Delta_3 &= \{\Box\,(\neg\psi_0 \vee \varphi_0 \vee \ldots \vee \varphi_n), \ldots, \Box\,(\neg\psi_n \vee \varphi_0 \vee \ldots \vee \varphi_n)\}
\end{aligned}$$

The formulas $\Box\,\circ\neg\gamma$ and $\Box\,(\varphi_0 \vee \ldots \vee \varphi_n)$ are logical consequences of the set

$$\Sigma = \{\varphi_0 \vee \ldots \vee \varphi_n\} \cup \Delta_1 \cup \Delta_2 \cup \Delta_3$$

Additionally, for the formula $\chi = \bigwedge_{\alpha\in\Sigma} \alpha$, it holds that $\neg\chi \vee \circ\chi$ is true in every state of every PLTL-structure. Therefore χ is an invariant that states, in an intricate way, that the eventuality $\Diamond\,\gamma$ cannot be true from the next state onwards. Note also that the formula $\Box\,(\neg(\varphi_0 \vee \ldots \vee \varphi_n) \vee \circ(\varphi_0 \vee \ldots \vee \varphi_n))$ is a logical consequence of $\Delta_1 \cup \Delta_2 \cup \Delta_3$. So that, if we restrict ourselves to the set of models of $\Delta_1 \cup \Delta_2 \cup \Delta_3$, we could say that the formula $\varphi_0 \vee \ldots \vee \varphi_n$ is an invariant with respect to such models. Since the set Σ can be formed by an arbitrary number of formulas, the invariant χ (unlike eventualities) cannot be trivially detected. Additionally, Σ could just be a subset of another set of formulas. More details about invariants can be found in e.g. [Fisher et al. 2001; Paech 1988; Pliuskevicius 1991; 1992].

3. THE LANGUAGE TEDILOG

In this section we introduce the syntax of TeDiLog along with an illustrative example of a TeDiLog specification for a reactive system.

$$\begin{array}{ll}
L ::= p \mid \neg p & H ::= \bot \mid A \vee H\\
T ::= L\,\mathcal{U}\,p \mid L\,\mathcal{R}\,p \mid \Diamond p \mid \Box p & B ::= \top \mid A \wedge B\\
A ::= \circ^i p \mid \circ^i T & D ::= \Box^b(A \vee H \leftarrow B)\\
 & G ::= \Box^b(\bot \leftarrow B)
\end{array}$$

where $p \in$ Prop, $i \geq 0$, $\bot$ is the empty disjunction,
$\top$ is the empty conjunction and $b \in \{0, 1\}$.

Fig. 1. TeDiLog's Syntax

The programming language TeDiLog is a twofold extension of propositional Horn clauses that incorporates temporal connectives in atoms and disjunctions in clause heads. It is the Temporal Disjunctive Logic programming language given in Figure 1, where the metavariable A denotes *atom*, L stands for (classical) literal, T for temporal atom, H for head, B for body, D for (disjunctive) program clause, and G for goal clause. Let n be a natural number, we respectively denote by $\circ^n$, $\Box^n$ and $\Diamond^n$ the sequences of n connectives $\circ$, $\Box$ and $\Diamond$. Since $\Box\Box\,\psi$ and $\Diamond\Diamond\,\psi$ are respectively equivalent to $\Box\,\psi$ and $\Diamond\,\psi$, we use only $\Box^n$ and $\Diamond^n$ with $n = 0$ or $n = 1$. Along the rest of the paper superscripts b (from bit) range in $\{0, 1\}$. These kinds of superscripts are notation, hence they are not part of the syntax. Due to the superscript b, the metavariable D represents two kinds of clauses. The expression $\Box^b(H \leftarrow B)$, for $b = 0$, represents $H \leftarrow B$, 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, $\Box^b(\bot \leftarrow \top)$ represents the two possible syntactic forms of the empty clause, as now- or always-clause.

Definition 3.1. Given a set of clauses Φ, the set $\mathsf{alw}(\Phi)$ is formed by all the always-clauses in Φ, i.e. all the clauses of the form $\Box\,(H \leftarrow B)$. In addition, the set $\mathsf{now}(\Phi)$ is $\Phi \setminus \mathsf{alw}(\Phi)$.

A program is a set of program clauses and a goal is a set of goal clauses.

The set of atoms of a clause $C = \Box^b(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$ is $\{A_1, \ldots, A_m, A'_1, \ldots, A'_n\}$. We assume that there is neither repetitions nor established order in the atoms of a head or a body. An atom is said to be $\circ$-free if it is a temporal atom or a classical propositional atom. The connective $\circ$ is distributive over every other connective and, consequently, $\circ\Box\,(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$ is equivalent to $\Box\,(\circ A_1 \vee \ldots \vee \circ A_m \leftarrow \circ A'_1 \wedge \ldots \wedge \circ A'_n)$. Given a head, body, program clause or goal clause ψ, we denote by $\circ\psi$ the head, body, program clause or goal clause that is obtained by adding one $\circ$ connective to every atom in ψ. For instance, $\circ\Box\,(p \vee q \leftarrow \circ r)$ denotes $\Box\,(\circ p \vee \circ q \leftarrow \circ\circ r)$ and $\circ\Box\,(\bot \leftarrow \circ r)$ denotes $\Box\,(\bot \leftarrow \circ\circ r)$. Note that $\circ\bot$ is written just $\bot$ and $\circ\top$ is written $\top$.

A clause $\Box^b(H \leftarrow B)$ is semantically equivalent to the formula $\Box^b(H \vee \neg B)$. Consequently, not only the temporal atoms of the form $\Diamond p$ and $L\,\mathcal{U}\,p$ that occur in the head H of the clause behave as eventualities, but also the temporal atoms $\Box\,p$ and $L\,\mathcal{R}\,p$ in the body B, which respectively correspond to (temporal) literals $\neg\Box\,p$ and $\neg(L\,\mathcal{R}\,p)$. Hence, we define the eventuality literals of a clause, on the basis of the notion of eventuality (see Definition 2.1).

Definition 3.2. Let C be a clause $\Box^b(A_1 \vee \ldots \vee A_m \leftarrow A'_1 \wedge \ldots \wedge A'_n)$. $\mathsf{Lits}(C)$ denotes the set $\{A_1, \ldots, A_m, \neg A'_1, \ldots, \neg A'_n\}$ whose elements are called the *temporal literals* of C. Additionally, $\mathsf{EventLits}(C)$ denotes the set of all the *eventuality literals* in C, i.e. $\{N \mid N \in \mathsf{Lits}(C)$ and N is an eventuality$\}$.
Both notations are extended to a set of clauses Ψ in the obvious manner: $\mathsf{Lits}(\Psi) = \bigcup_{C\in\Psi} \mathsf{Lits}(C)$ and $\mathsf{EventLits}(\Psi) = \bigcup_{C\in\Psi} \mathsf{EventLits}(C)$.

Note that eventuality literals from clauses have one of the following four forms: $\Diamond p$, $L\,\mathcal{U}\,p$, $\neg\Box\,p$ and $\neg(L\,\mathcal{R}\,p)$, where p is a propositional variable and L a classical literal.

TeDiLog is syntactically a sublanguage of PLTL, but every PLTL-formula can be translated into TeDiLog by using, in general, new propositional variables. The translation yields an equisatisfiable set of (program and goal) clauses. For example, the PLTL-formula $\Box\,\neg p \leftarrow q$ can be translated into TeDiLog as the goal clause $\bot \leftarrow q \wedge \Diamond p$ but also as the set formed by the program clause $\Box\,r \leftarrow q$ and the goal clause $\Box\,(\bot \leftarrow r \wedge p)$ where r is a fresh propositional variable. For the PLTL-formula $\Box\,(x \vee y) \leftarrow z$ we obtain the program clauses $\Box\,w \leftarrow z$ and $\Box\,(x \vee y \leftarrow w)$ where w is a fresh propositional variable. A detailed translation method is presented in [Gaintzarain et al. 2013].

To finish this section, let us illustrate (with an example) how TeDiLog can be used to specify reactive systems and to verify properties that are satisfied by these systems. We also use the next example to compare the expressiveness of TeDiLog with the more closely related proposals in the literature.

Example 3.3. Let us consider a system where a device (dv) and a system manager (sm) interact with each other. When the device dv needs to execute a process, it sends a request req_dv to the system manager sm to get permission and goes into waiting-state until the system manager sm sends the acknowledgement signal ack_sm giving permission to execute the process.

$$\Box\,(waiting\_dv\,\mathcal{U}\,ack\_sm \leftarrow req\_dv) \qquad (1)$$

Whenever dv asks for permission, the system manager sm will eventually give permission by sending the acknowledgement signal ack_sm in a later state.

$$\Box\,(\circ\Diamond\, ack\_sm \leftarrow req\_dv) \tag{2}$$

Once the system manager produces the signal ack_sm (giving permission), the device dv goes into working-state until it communicates the end of the process by means of the eop_dv signal.

$$\Box\,(working\_dv\,\mathcal{U}\,eop\_dv \leftarrow ack\_sm) \tag{3}$$

Whenever the device generates the eop_dv signal, then it will not be in working-state until it receives the ack_sm signal giving permission to execute another process.

$$\Box\,(\neg working\_dv\,\mathcal{U}\,ack\_sm \leftarrow eop\_dv) \tag{4}$$

From time to time, the system manager generates a control signal ctr_sm

$$\Box\,(\Diamond\, ctr\_sm \leftarrow \top) \tag{5}$$

The interaction generated after the control signal ctr_sm corresponds to the fact that the system manager has to regularly control whether the device is correctly connected to the system. This signal ctr_sm is always eventually followed by the signal $conn\_sm$ which is received by the device.

$$\Box\,(\Diamond\, conn\_sm \leftarrow ctr\_sm) \tag{6}$$

After receiving the signal $conn\_sm$, the device dv answers by sending the signal $conn\_dv$ to the system manager.

$$\Box\,(\circ\Diamond\, conn\_dv \leftarrow conn\_sm) \tag{7}$$

The device dv is considered to be in communicating-state (com_dv) while the arising of the $conn\_dv$ signal (now or in a future moment) is guaranteed.

$$\Box\,(com\_dv \leftarrow \Diamond\, conn\_dv) \tag{8}$$

We would like to remark that the clauses (2) and (5)-(7) cannot be expressed neither in Chronolog nor in Templog because of the eventualities in their heads. However, all of them are syntactically correct in Gabbay's Temporal Prolog. As for the clauses (1), (3) and (4), they contain the $\mathcal{U}$ connective which is not allowed in the above mentioned three declarative TLP languages.

Now, we can check whether the system specified by the TeDiLog clauses (1)-(8) verifies some properties such as fairness, liveness, safety, mutual exclusion, etc. This is made by writing the intended property as a TeDiLog goal and then checking if that goal can be inferred from the program. For example we would be interested in checking whether the device dv will always keep in communicating-state. The corresponding goal would be $\{\bot \leftarrow \Box\, com\_dv\}$. Actually, the refutational mechanism of TeDiLog checks the unsatisfiability of the eventuality $\Diamond\,\neg com\_dv$ with respect to the specification.
None of the just above mentioned three languages (Chronolog, Templog and Gabbay's Temporal Prolog) allows always-atoms in clause bodies, hence the previous goal is not expressible in any of these declarative TLP languages.
The program clauses (1)-(8) can be expressed in propositional MetateM, although some translation into SNF is needed. For the resulting specification, the MetateM execution system builds a model step by step in the imperative future style. The process will stop when a loop that gives rise to an ultimately periodic model for the program is detected. If we add to the specification the SNF clauses that correspond to the goal $\bot \leftarrow \Box\, com\_dv$, then MetateM finitely detects the unsatisfiability of the extended specification. ■

$$(Res)\quad \frac{\Box^{b}(A \vee H \leftarrow B) \qquad \Box^{b'}(H' \leftarrow A \wedge B')}{\Box^{b\times b'}(H \vee H' \leftarrow B \wedge B')} \qquad b, b' \in \{0,1\}$$

Fig. 2. The Resolution Rule

$$\begin{array}{ll}(\mathcal{U}\,H_{+}) & \Box^{b}((p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B)\\ & \longmapsto \{\Box^{b}(p_2 \vee p_1 \vee H \leftarrow B),\ \ \Box^{b}(p_2 \vee \circ(p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B)\}\end{array}$$

Fig. 3. The Context-Free Rule $(\mathcal{U}\,H_{+})$

$$\begin{array}{ll}(\mathcal{U}\,H_{-}) & \Box^{b}((\neg p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B)\\ & \longmapsto \{\Box^{b}(p_2 \vee H \leftarrow p_1 \wedge B),\ \ \Box^{b}(p_2 \vee \circ(\neg p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B)\}\\[1em] (\mathcal{U}\,B_{+}) & \Box^{b}(H \leftarrow (p_1\,\mathcal{U}\,p_2) \wedge B)\\ & \longmapsto \{\Box^{b}(H \leftarrow p_2 \wedge B),\ \ \Box^{b}(H \leftarrow p_1 \wedge \circ(p_1\,\mathcal{U}\,p_2) \wedge B)\}\\[1em] (\mathcal{U}\,B_{-}) & \Box^{b}(H \leftarrow (\neg p_1\,\mathcal{U}\,p_2) \wedge B)\\ & \longmapsto \{\Box^{b}(H \leftarrow p_2 \wedge B),\ \ \Box^{b}(p_1 \vee H \leftarrow \circ(\neg p_1\,\mathcal{U}\,p_2) \wedge B)\}\end{array}$$

Fig. 4. The Context-Free Rules $(\mathcal{U}\,H_{-})$, $(\mathcal{U}\,B_{+})$ and $(\mathcal{U}\,B_{-})$

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 *jumping to the next state* and for *subsumption*. We explain these four kinds of rules in the following four subsections.

4.1. The Resolution Rule

The TeDiLog's resolution rule (Res) is a natural generalization of the classical rule for binary resolution. It is depicted in Figure 2 in the usual format of premises and resolvent separated by an horizontal line. The rule (Res) applies to two temporal clauses such that one of the atoms in the head of one clause is in the body of the other clause. The premises can be headed or not by an always connective. By means of the product $b \times b'$ in the superscript of the resolvent, the resolvent is an always-clause if and only if both premises are always-clauses. 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.

4.2. The Temporal Rules

The temporal rules serve to transform the set of clauses according to the inductive definitions of temporal atoms. We write them as transformation rules $\Phi \mapsto \Psi$ where Φ and Ψ are sets of clauses, respectivelly called the antecedent and the consequent. Temporal rules are grouped into two classes. On the one hand, the *context-free rules* are based on the usual inductive definitions of the temporal connectives. The antecedent and consequent of any context-free rule are logically equivalent. On the other hand, the *context-dependent* rules come up from a more complex inductive definition of the connective $\mathcal{U}$, and their antecedent and consequent are equisatisfiable.

$$
\begin{array}{ll}
(\mathcal{R}\,H_+) & \Box^b((p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B) \\
& \longmapsto \{\Box^b(p_2 \vee H \leftarrow B),\ \ \Box^b(p_1 \vee \circ(p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B)\} \\[1em]
(\mathcal{R}\,H_-) & \Box^b((\neg p_1\,\mathcal{R}\,p_2) \vee H \leftarrow B) \\
& \longmapsto \{\Box^b(p_2 \vee H \leftarrow B),\ \ \Box^b(\circ(\neg p_1\,\mathcal{R}\,p_2) \vee H \leftarrow p_1 \wedge B)\} \\[1em]
(\mathcal{R}\,B_+) & \Box^b(H \leftarrow (p_1\,\mathcal{R}\,p_2) \wedge B) \\
& \longmapsto \{\Box^b(H \leftarrow p_2 \wedge p_1 \wedge B),\ \ \Box^b(H \leftarrow p_2 \wedge \circ(p_1\,\mathcal{R}\,p_2) \wedge B)\} \\[1em]
(\mathcal{R}\,B_-) & \Box^b(H \leftarrow (\neg p_1\,\mathcal{R}\,p_2) \wedge B) \\
& \longmapsto \{\Box^b(p_1 \vee H \leftarrow p_2 \wedge B),\ \ \Box^b(H \leftarrow p_2 \wedge \circ(\neg p_1\,\mathcal{R}\,p_2) \wedge B)\}
\end{array}
$$

Fig. 5. The Context-Free Rules $(\mathcal{R}\,H_+)$, $(\mathcal{R}\,H_-)$, $(\mathcal{R}\,B_+)$ and $(\mathcal{R}\,B_-)$

$$
\begin{array}{l}
\mathsf{def}(a, L, \emptyset) = \{\Box\,(\bot \leftarrow a)\} \\
\mathsf{def}(a, p, \Delta) = \{\Box\,(p \leftarrow a)\} \cup \{\Box\,(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset \\
\mathsf{def}(a, \neg p, \Delta) = \{\Box\,(\bot \leftarrow p \wedge a)\} \cup \{\Box\,(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\Delta\} \text{ if } \Delta \neq \emptyset
\end{array}
$$

Fig. 6. The set of clauses $\mathsf{def}(a, L, \Delta)$

$$
\begin{array}{ll}
(\mathcal{U}\,C_+) & \Omega \cup \{\Box^{b_i}((p_1\,\mathcal{U}\,p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto\ \Omega\ \cup\ \{p_2 \vee p_1 \vee H_i \leftarrow B_i,\ p_2 \vee \circ(a\,\mathcal{U}\,p_2) \vee H_i \leftarrow B_i\ \mid\ 1 \leq i \leq n\} \\
& \qquad\quad \cup\ \mathsf{def}(a, p_1, \mathsf{now}(\Omega)) \\
& \qquad\quad \cup\ \{\Box^{b_i}(\circ(p_1\,\mathcal{U}\,p_2) \vee \circ H_i \leftarrow \circ B_i)\ \mid\ b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}
$$

where $\mathsf{def}(a, p_1, \mathsf{now}(\Omega))$ is defined in Figure 6.

Fig. 7. The Context-Dependent Rule $(\mathcal{U}\,C_+)$

4.2.1. Context-Free Rules. In the context-free rules, the antecedent Φ is a singleton and we write directly its unique clause. The context-free rule $(\mathcal{U}\,H_+)$ –depicted in Figure 3– deals with an atom of the form $p_1\,\mathcal{U}\,p_2$ that appears in the head of a clause. This rule replaces a clause of the form $\Box^b((p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B)$ with a logically equivalent set of (two) clauses according to the well-known inductive definition $p_1\,\mathcal{U}\,p_2 \equiv p_2 \vee (p_1 \wedge \circ(p_1\,\mathcal{U}\,p_2))$, from which the distribution law guarantees the equivalence

$$p_1\,\mathcal{U}\,p_2 \equiv (p_2 \vee p_1) \wedge (p_2 \vee \circ(p_1\,\mathcal{U}\,p_2)) \qquad (9)$$

which justifies that the antecedent $(p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B$ of the rule $(\mathcal{U}\,H_+)$ is logically equivalent to the conjunction of the two clauses in its consequent: $p_2 \vee p_1 \vee H \leftarrow B$ and $p_2 \vee \circ(p_1\,\mathcal{U}\,p_2) \vee H \leftarrow B$. Our system also includes (see Figure 4) the rules $(\mathcal{U}\,H_-)$, $(\mathcal{U}\,B_+)$ and $(\mathcal{U}\,B_-)$ for the respective occurrences of $\neg p_1\,\mathcal{U}\,p_2$ in the clause head and $p_1\,\mathcal{U}\,p_2$ and $\neg p_1\,\mathcal{U}\,p_2$ in the clause body. The rules $(\mathcal{U}\,H_-)$, $(\mathcal{U}\,B_+)$ and $(\mathcal{U}\,B_-)$ are respectively obtained by using the inductive definition $L\,\mathcal{U}\,p \equiv p \vee (L \wedge \circ(L\,\mathcal{U}\,p))$ for $\neg p_1\,\mathcal{U}\,p_2$ in the clause head, and $p_1\,\mathcal{U}\,p_2$ and $\neg p_1\,\mathcal{U}\,p_2$ in the clause body. Additionally, the rules $(\mathcal{R}\,H_+)$, $(\mathcal{R}\,H_-)$, $(\mathcal{R}\,B_+)$ and $(\mathcal{R}\,B_-)$ in Figure 5 are obtained from the inductive definition $L\,\mathcal{R}\,p \equiv p \wedge (L \vee \circ(L\,\mathcal{R}\,p))$ by considering the same four kinds of occurrences of the release connective $\mathcal{R}$ in a clause.

4.2.2. Context-Dependent Rules. The context-dependent rules are based on an inductive definition of $\mathcal{U}$ that takes into account, not only the clauses where the temporal atom occurs, but also the remaining now-clauses in the antecedent of the rule. The rule $(\mathcal{U} C_+)$ in Figure 7 is the context-dependent rule that deals with atoms of the form $p_1 \mathcal{U} p_2$ in clause heads. The antecedent of $(\mathcal{U} C_+)$ must be interpreted as a partition of the whole set of clauses (on which we are applying temporal resolution) into two sets. The second set $\{\Box^{b_i}((p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}$ in the antecedent is a non-empty set of clauses that contain the same temporal atom $p_1 \mathcal{U} p_2$ in the head. The first set, Ω, is formed by all the remaining clauses and it is called the *context*. The crucial idea behind the context-dependent rule $(\mathcal{U} C_+)$ (and, hence, behind the resolution mechanism of TeDiLog) is based on the following equisatisfiability result that relates two eventualities.

PROPOSITION 4.1. *Let Δ be a set of formulas, $\Delta_1 = \Delta \cup \{p_2 \vee p_1 \vee \beta, p_2 \vee \circ(p_1 \mathcal{U} p_2) \vee \beta\}$ and $\Delta_2 = \Delta \cup \{p_2 \vee p_1 \vee \beta, p_2 \vee \circ((p_1 \wedge \neg\Delta) \mathcal{U} p_2) \vee \beta\}$. Then Δ_1 and Δ_2 are equisatisfiable.*

PROOF. Suppose that Δ_1 has a model $\mathcal{M}$. If $\langle \mathcal{M}, s_0\rangle \models \Delta \cup \{p_2\}$ or $\langle \mathcal{M}, s_0\rangle \models \Delta \cup \{\beta\}$, then $\mathcal{M}$ is also a model of Δ_2. Otherwise, $\langle \mathcal{M}, s_0\rangle \models \{p_1, \circ(p_1 \mathcal{U} p_2)\}$ and p_2 should be satisfied in some state s_j with $j \geq 1$ and p_1 is true in all the states s_h such that $0 \leq h < j$. Let k be the greatest index in $\{0, \ldots, j-1\}$ such that $\langle \mathcal{M}, s_k\rangle \models \Delta$ and Δ is not satisfied in the states $s_{k+1}, \ldots, s_{j-1}$ of $\mathcal{M}$. Then, we can construct a model $\mathcal{M}'$ of Δ by simply deleting the states $s_0, \ldots, s_{k-1}$ in $\mathcal{M}$. As a consequence of the choice of k, the PLTL-structure $\mathcal{M}'$ is also a model of $\{p_1, \circ((p_1 \wedge \neg\Delta) \mathcal{U} p_2)\}$. Hence, $\mathcal{M}' \models \Delta_2$. Conversely, any model of Δ_2 is a model of Δ_1. □

Now, we will transform the antecedent of $(\mathcal{U} C_+)$ into its consequent, while preserving equisatisfiability (indeed, logical equivalence is preserved at most steps).
The first transformation step is based on the equivalence $\Box\psi \equiv \psi \wedge \Box\circ\psi$. Consequently, each clause $\Box^{b_i}((p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i)$ such that $b_i = 1$ is split (while clauses with $b_i = 0$ remain unchanged). So that, the set in the antecedent of $(\mathcal{U} C_+)$:

$$\Psi_0 = \Omega \ \cup \ \{\Box^{b_i}((p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}$$

is equivalent to

$$\begin{aligned}\Psi_1 = \Omega \ &\cup \{(p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\} \\ &\cup \{\Box^{b_i}(\circ(p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

Then, by the classical inductive definition of $\mathcal{U}$ (see previous (9)), Ψ_1 is equivalent to

$$\begin{aligned}\Psi_2 = \Omega \ &\cup \{p_2 \vee p_1 \vee H_i \leftarrow B_i, \mid \ 1 \leq i \leq n\} \\ &\cup \underline{\{p_2 \vee \circ(p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i \ \mid \ 1 \leq i \leq n\}} \\ &\cup \{\Box^{b_i}(\circ(p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

Let Υ be the last set in the description of Ψ_2, that is

$$\Upsilon = \{\Box^{b_i}(\circ(p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\},$$

we replace the above underlined set (inside Ψ_2) with the following set

$$\{p_2 \vee \circ((p_1 \wedge \neg(\Omega \cup \Upsilon)) \mathcal{U} p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\} \tag{10}$$

Hence, we obtain

$$\begin{aligned}\Psi_3 = \Omega \ &\cup \{p_2 \vee p_1 \vee H_i \leftarrow B_i, \mid \ 1 \leq i \leq n\} \\ &\cup \{p_2 \vee \circ((p_1 \wedge \neg(\Omega \cup \Upsilon)) \mathcal{U} p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\} \\ &\cup \{\Box^{b_i}(\circ(p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

By Proposition 4.1, the sets Ψ_2 and Ψ_3 are equisatisfiable. Additionally, any set of the form

$$\{\Box\chi_1, \Box\chi_2, \ldots, \Box\chi_m, \circ((\varphi \wedge (\gamma \vee \neg\Box\chi_1 \vee \neg\Box\chi_2 \vee \ldots \vee \neg\Box\chi_m)) \mathcal{U} \psi)\}$$

is equivalent to the set

$$\{\Box\chi_1, \Box\chi_2, \ldots, \Box\chi_m, \circ((\varphi \wedge \gamma)\,\mathcal{U}\,\psi)\}$$

because if the formulas $\chi_1, \chi_2, \ldots, \chi_m$ are true from now forever, then the truth of the formula

$$\circ((\varphi \wedge (\gamma \vee \neg\Box\chi_1 \vee \neg\Box\chi_2 \vee \ldots \vee \neg\Box\chi_m))\,\mathcal{U}\,\psi)$$

does not depend on the truth of the disjunction $\neg\Box\chi_1 \vee \neg\Box\chi_2 \vee \ldots \vee \neg\Box\chi_m$ which should be false. Consequently, it is not necessary to consider the clauses that belong to $\mathsf{alw}(\Omega) \cup \Upsilon$ (see Definition 3.1) in the subset of Ψ_3 considered in (10) because only clauses in $\mathsf{now}(\Omega)$ are needed. Therefore, we replace the subformula $\neg(\Omega \cup \Upsilon)$ with $\neg\mathsf{now}(\Omega)$ in Ψ_3 and we obtain the following (logically equivalent) set

$$\begin{aligned}\Psi_4 = \Omega\ &\cup \{p_2 \vee p_1 \vee H_i \leftarrow B_i,\ |\ 1 \leq i \leq n\}\\ &\cup \{p_2 \vee \circ((p_1 \wedge \neg\mathsf{now}(\Omega))\,\mathcal{U}\,p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\}\\ &\cup \{\Box^{b_i}(\circ(p_1\,\mathcal{U}\,p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

Now, since the above formula $\circ((p_1 \wedge \neg\mathsf{now}(\Omega))\,\mathcal{U}\,p_2)$ is not (in general) an atom, we should transform Ψ_4 into clausal form. For that, we substitute the subformula $p_1 \wedge \neg\mathsf{now}(\Omega)$ by the fresh variable a and we add the clauses that define the meaning of a. This gives the following set Ψ_5 where $\mathsf{def}(a, p_1, \mathsf{now}(\Omega))$ is the result of transforming the formula $\Box((p_1 \wedge \neg\mathsf{now}(\Omega)) \leftarrow a)$ to a set of clauses (whose definition is given in Figure 6):

$$\begin{aligned}\Psi_5 = \Omega\ &\cup \{p_2 \vee p_1 \vee H_i \leftarrow B_i,\ |\ 1 \leq i \leq n\}\\ &\cup \{p_2 \vee \circ(a\,\mathcal{U}\,p_2) \vee H_i \leftarrow B_i \mid 1 \leq i \leq n\}\\ &\cup \mathsf{def}(a, p_1, \mathsf{now}(\Omega))\\ &\cup \{\Box^{b_i}(\circ(p_1\,\mathcal{U}\,p_2) \vee \circ H_i \leftarrow \circ B_i) \mid b_i = 1, 1 \leq i \leq n\}\end{aligned}$$

Finally, let us check that the sets Ψ_4 and Ψ_5 are equisatisfiable. On the one hand, since a does not appear in Ψ_4, a model $\mathcal{M}'$ of Ψ_5 can be built from a model $\mathcal{M}$ of Ψ_4 by just defining $V_{\mathcal{M}'}(s'_j)$ as $V_{\mathcal{M}}(s_j) \cup \{a\}$ if $p_1 \wedge \neg\mathsf{now}(\Omega)$ is true in the state s_j of $\mathcal{M}$ and by defining $V_{\mathcal{M}'}(s'_j)$ as $V_{\mathcal{M}}(s_j) \setminus \{a\}$ if $p_1 \wedge \neg\mathsf{now}(\Omega)$ is false in the state s_j of $\mathcal{M}$. On the other hand, since every model of Ψ_5 satisfies the formula $\Box((p_1 \wedge \neg\mathsf{now}(\Omega)) \leftarrow a)$, we can ensure that $p_1 \wedge \neg\mathsf{now}(\Omega)$ is true in any state s of a model of Ψ_5 whenever a is true in s. Consequently, $\circ((p_1 \wedge \neg\mathsf{now}(\Omega))\,\mathcal{U}\,p_2)$ is true in any state s of a model of Ψ_5 whenever $\circ(a\,\mathcal{U}\,p_2)$ is true in s. Therefore, every model of Ψ_5 is also a model of Ψ_4. As can be seen in the above reasoning, the clauses that correspond to the formula $\Box((p_1 \wedge \neg\mathsf{now}(\Omega)) \rightarrow a)$ are not needed for equisatisfiability.
To summarize, the initial set Ψ_0 –which is the antecedent of the rule $(\mathcal{U}\,C_+)$– and the last set Ψ_5 –which is the consequent of the rule $(\mathcal{U}\,C_+)$– are equisatisfiable. Our system also includes a similar context-dependent rule $(\mathcal{U}\,C_-)$ for $\neg p_1\,\mathcal{U}\,p_2$ in the head, which is depicted in Figure 8.
The context-dependent rules $(\mathcal{R}\,C_+)$ and $(\mathcal{R}\,C_-)$ in Figure 8 are due to the fact that a release atom appearing in the body of a clause C is an eventuality literal of C (see Definition 3.2). The rules for $\mathcal{R}$ are explained by its duality with $\mathcal{U}$. Additionally, by using the definitions $\Diamond\varphi \equiv \neg\varphi\,\mathcal{U}\,\varphi$ and $\Box\varphi \equiv \neg\varphi\,\mathcal{R}\,\varphi$, the context-free rules $(\Diamond H_+)$, $(\Diamond B_+)$, $(\Box H_+)$ and $(\Box B_+)$ and the context-dependent rules $(\Diamond C_+)$ and $(\Box C_+)$ are derived. These derived rules are depicted in Figure 10.

4.3. The Rule for Jumping to the Next State

The *unnext rule* in Figure 11 applies to a pair formed by a program and a goal, giving a new pair of program and goal. The expression $\mathsf{unnext}(\Psi)$ stands for the set of all clauses that should be satisfied at the next state of a state that satisfies the set of clauses Ψ. Note that the definition of the function unnext implicitly uses the equivalence $\Box\varphi \equiv \varphi \wedge \Box\circ\varphi$ and also that the unnext target of a program (resp. goal) is also a program (resp. goal). It is worth remembering that $\top$ and $\bot$ respectively represent the empty body and the empty head, and it holds that every atom in $\top$ and $\bot$ is of the form $\circ A$. For example, $\mathsf{unnext}(\{\Box(\circ r \leftarrow \top), \Box(q \leftarrow \top)\})$ is the set $\{\Box(\circ r \leftarrow \top), \Box(q \leftarrow \top), r \leftarrow \top\}$.

$$
\begin{array}{ll}
(\mathcal{U}C_-) & \Omega \cup \{\Box^{b_i}((\neg p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Omega \;\cup\; \{p_2 \vee H_i \leftarrow p_1 \wedge B_i,\; p_2 \vee \circ(a \mathcal{U} p_2) \vee H_i \leftarrow B_i \;\mid\; 1 \leq i \leq n\} \\
& \qquad \cup\; \mathsf{def}(a, \neg p_1, \mathsf{now}(\Omega)) \\
& \qquad \cup\; \{\Box^{b_i}(\circ(\neg p_1 \mathcal{U} p_2) \vee \circ H_i \leftarrow \circ B_i) \;\mid\; b_i = 1 \text{ and } 1 \leq i \leq n\} \\
\\
(\mathcal{R}C_+) & \Omega \cup \{\Box^{b_i}(H_i \leftarrow (p_1 \mathcal{R} p_2) \wedge B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Omega \;\cup\; \{H_i \leftarrow p_2 \wedge p_1 \wedge B_i,\; H_i \leftarrow p_2 \wedge \circ(\neg a \mathcal{R} p_2) \wedge B_i \mid 1 \leq i \leq n\} \\
& \qquad \cup\; \mathsf{def}(a, \neg p_1, \mathsf{now}(\Omega)) \\
& \qquad \cup\; \{\Box^{b_i}(\circ H_i \leftarrow \circ(p_1 \mathcal{R} p_2) \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\} \\
\\
(\mathcal{R}C_-) & \Omega \cup \{\Box^{b_i}(H_i \leftarrow (\neg p_1 \mathcal{R} p_2) \wedge B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Omega \;\cup\; \{p_1 \vee H_i \leftarrow p_2 \wedge B_i,\; H_i \leftarrow p_2 \wedge \circ(\neg a \mathcal{R} p_2) \wedge B_i \mid 1 \leq i \leq n\} \\
& \qquad \cup\; \mathsf{def}(a, p_1, \mathsf{now}(\Omega)) \\
& \qquad \cup\; \{\Box^{b_i}(\circ H_i \leftarrow \circ(\neg p_1 \mathcal{R} p_2) \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}
$$

where $n \geq 1$, $a \in \mathsf{Prop}$ is fresh and def is in Figure 6.

Fig. 8. The Context-Dependent Rules $(\mathcal{U}C_-)$, $(\mathcal{R}C_+)$ and $(\mathcal{R}C_-)$

$$
\mathsf{def}(a, \mathsf{now}(\Omega)) = \begin{cases} \{\Box(\bot \leftarrow a)\} & \text{if } \mathsf{now}(\Omega) = \emptyset \\ \{\Box(H \leftarrow B \wedge a) \mid H \leftarrow B \in \neg\mathsf{now}(\Omega)\} & \text{otherwise} \end{cases}
$$

Fig. 9. The set of clauses $\mathsf{def}(a, \mathsf{now}(\Omega))$

$$
\begin{array}{ll}
(\Diamond H_+) & \Box^{b}(\Diamond p \vee H \leftarrow B) \longmapsto \{\Box^{b}(p \vee \circ\Diamond p \vee H \leftarrow B)\} \\
(\Diamond B_+) & \Box^{b}(H \leftarrow \Diamond p \wedge B) \longmapsto \{\Box^{b}(H \leftarrow p \wedge B),\; \Box^{b}(H \leftarrow \circ\Diamond p \wedge B)\} \\
(\Box H_+) & \Box^{b}(\Box p \vee H \leftarrow B) \longmapsto \{\Box^{b}(p \vee H \leftarrow B),\; \Box^{b}(\circ\Box p \vee H \leftarrow B)\} \\
(\Box B_+) & \Box^{b}(H \leftarrow \Box p \wedge B) \longmapsto \{\Box^{b}(H \leftarrow p \wedge \circ\Box p \wedge B)\} \\
\\
(\Diamond C_+) & \Omega \cup \{\Box^{b_i}(\Diamond p \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Omega \;\cup\; \{p \vee \circ(a \mathcal{U} p) \vee H_i \leftarrow B_i \;\mid\; 1 \leq i \leq n\} \\
& \qquad \cup\; \mathsf{def}(a, \mathsf{now}(\Omega)) \\
& \qquad \cup\; \{\Box^{b_i}(\circ\Diamond p \vee \circ H_i \leftarrow \circ B_i) \;\mid\; b_i = 1 \text{ and } 1 \leq i \leq n\} \\
\\
(\Box C_+) & \Omega \cup \{\Box^{b_i}(H_i \leftarrow \Box p \wedge B_i) \mid 1 \leq i \leq n\} \\
& \longmapsto \Omega \;\cup\; \{H_i \leftarrow p \wedge \circ(\neg a \mathcal{R} p) \wedge B_i \mid 1 \leq i \leq n\} \\
& \qquad \cup\; \mathsf{def}(a, \mathsf{now}(\Omega)) \\
& \qquad \cup\; \{\Box^{b_i}(\circ H_i \leftarrow \circ\Box p \wedge \circ B_i) \mid b_i = 1 \text{ and } 1 \leq i \leq n\}
\end{array}
$$

where $n \geq 1$, $a \in \mathsf{Prop}$ is fresh and $\mathsf{def}(a, \mathsf{now}(\Omega))$ is in Figure 9

Fig. 10. Derived Rules for $\Diamond$ and $\Box$

$$(Unx)\quad (\Pi,\Gamma) \longmapsto (\mathsf{unnext}(\Pi), \mathsf{unnext}(\Gamma))$$
$$\text{where } \mathsf{unnext}(\Psi) = \mathsf{alw}(\Psi)\ \cup\ \{H \leftarrow B \mid \Box^{b}(\circ H \leftarrow \circ B) \in \Psi\}$$

Fig. 11. The Rule (Unx)

$$(Sbm)\quad \{\Box^{b}(H \leftarrow B), \Box^{b}(H' \leftarrow B')\} \longmapsto \{\Box^{b}(H' \leftarrow B')\}$$
$$\text{where } H' \subseteq H \text{ and } B' \subseteq B.$$

Fig. 12. The Rule (Sbm)

4.4. The Subsumption Rule

The rule (Sbm) is formulated in Figure 12. Regarding the clauses in the antecedent, it is said that the clause $\Box^{b}(H \leftarrow B)$ is subsumed by the clause $\Box^{b}(H' \leftarrow B')$.
Our resolution mechanism requires the subsumption rule (Sbm) for completeness (see Lemma 41 in [Gaintzarain et al. 2013]).

5. TEDILOG SEMANTICS

In this section we summarize our results on TeDiLog semantics. The first subsection is devoted to the operational semantics that is formalized by means of the notion of IFT-derivation in Definition 5.8. The second subsection shows three sample derivations. In the third subsection we define the logical semantics. In the fourth subsection we prove the equivalence between the operational and the logical semantics. Finally, in the last subsection we define the fixpoint semantics and we prove the equivalence between the logical and the fixpoint semantics.

5.1. Operational Semantics

In this subsection we define the operational semantics of TeDiLog. Since TeDiLog's language is a syntactically simpler variant of the clausal language in [Gaintzarain et al. 2013], TeDiLog's operational semantics is based on a restriction of the TRS-resolution procedure, i.e. the complete resolution method for PLTL that was introduced in [Gaintzarain et al. 2013].

For TeDiLog completeness, it is not needed to exhaustively apply the rule (Res) for producing all possible resolvents. Indeed, the use of the rule (Res) can be limited as follows.

Definition 5.1. An application of the rule (Res) is a TeDiLog-*resolution* whenever at least one of the input clauses is of the form $\Box^{b}(\circ H \leftarrow B)$. In particular, it is called a *goal-resolution* if H is empty (i.e. it is a goal clause). Otherwise, it is called a *next-resolution* because both inputs are program clauses and at least one of them is of the form $\Box^{b}(\circ H \leftarrow B)$.

In the rest of this section, we define the temporal resolution procedure underlying TeDiLog, that we call IFT-resolution (for *Invariant-Free Temporal* resolution). Every step of an IFT-derivation consists in applying one of the rules presented in Section 4, with the above restriction for the rule (Res) and also a restricted use of the context-dependent rules. We first introduce the notion of local derivation and show that our restricted resolution works to achieve completeness of local derivation.

Definition 5.2. Let Π and Γ respectively be a program and a goal, a *local derivation* is a finite sequence $(\Pi_0,\Gamma_0) \mapsto (\Pi_1,\Gamma_1) \mapsto \ldots \mapsto (\Pi_n,\Gamma_n)$ such that $(\Pi_0,\Gamma_0) = (\Pi,\Gamma)$ and for all $j \in \{1,\ldots,n\}$ the pair (Π_j,Γ_j) is obtained from (Π_{j-1},Γ_{j-1}) by exactly one of the following

(a) the application of a temporal rule
(b) the application of the subsumption rule

(c) a goal-resolution (to a clause in Π_{j-1} and a clause in Γ_{j-1}) whose resolvent is neither in $\Pi_{j-1} \cup \Gamma_{j-1}$ nor subsumed by a clause in $\Pi_{j-1} \cup \Gamma_{j-1}$

(d) a next-resolution to two clauses in Π_{j-1} whose resolvent is neither in Π_{j-1} nor subsumed by a clause in Π_{j-1}.

Definition 5.3. A local derivation $(\Pi_0, \Gamma_0) \mapsto \ldots \mapsto (\Pi_n, \Gamma_n)$ is called a *local refutation* if $\Box^b(\bot \leftarrow \top) \in \Gamma_n$. Given a program Π and a goal Γ, the pair (Π, Γ) is *locally inconsistent* iff there exists a local refutation for (Π, Γ). Otherwise it is *locally consistent*.

Goal-resolution and next-resolution are two particular forms of the well-known set-of-support restriction of resolution[6] (see e.g. Section 2.6 in [Schöning 1989] and [Dixon and Fisher 1998]). In particular, they can be seen as N-restrictions of resolution. The completeness of these N-restrictions, along with the completeness of TRS-resolution (cf. [Gaintzarain et al. 2013]), ensure the completeness result for local derivations we are going to prove next.

PROPOSITION 5.4. *Let Δ be a set of clauses that does not contain temporal atoms (i.e. clauses are exclusively formed by atoms in* Prop *and atoms of the form $\circ A$). If Δ is locally inconsistent with respect to* TRS*-resolution, then* IFT*-resolution produces a local refutation by using only goal-resolution.*

PROOF. In local derivations, TeDiLog clauses can be seen as classical propositional clauses (This is a direct consequence of Proposition 25 in [Gaintzarain et al. 2013]). So that, by completeness of the N-restriction of resolution in classical propositional logic (see e.g. Section 2.6 in [Schöning 1989]) goal-resolution is enough to obtain a local refutation whenever there exists a local refutation. □

PROPOSITION 5.5. *Let Δ be a set of clauses that does not contain temporal atoms (i.e. clauses are exclusively formed by atoms in* Prop *and atoms of the form $\circ A$). If Δ is locally consistent with respect to* TRS*-resolution, then every clause of the form $\Box^b(\circ H \leftarrow \circ B)$ that is produced by* TRS*-resolution (from Δ) is also produced by* IFT*-resolution (that is, using only next-resolution and goal-resolution).*

PROOF. First, using goal-resolution we can generate all the minimal clauses of the form $\Box^b(H \leftarrow B)$, where H and B may contain propositional atoms. This is a direct consequence of the completeness of N-restriction of resolution (see e.g. [Schöning 1989]), along with the fact that temporal clauses (in local derivations) can be seen as classical propositional clauses (by Proposition 25 in [Gaintzarain et al. 2013]). Then, using next-resolution we can obtain all the clauses of the form $\Box^b(\circ H \leftarrow \circ B)$, by completeness of N-restriction of resolution and considering that a clause is negative whenever the head does not contain propositional atoms (in particular a clause of the form $\Box^b(\circ H \leftarrow \circ B)$ is negative). □

In order to formulate the local completeness result, we first define the notion of locally closed.

Definition 5.6. We say that a pair (Π, Γ) is *locally closed* if and only if it satisfies the following three conditions:

(a) The clauses in $\Pi \cup \Gamma$ are exclusively formed by atoms in Prop and atoms of the form $\circ A$.

(b) The subsumption rule (Sbm) cannot be applied to (Π, Γ).

(c) Every clause that can be obtained by applying TeDiLog-resolution (that is, goal- or next-resolution) is already in (Π, Γ) or it is subsumed by some clause in (Π, Γ).

THEOREM 5.7. *For any pair (Π_0, Γ_0), there exists a local derivation $(\Pi_0, \Gamma_0) \mapsto \ldots \mapsto (\Pi_n, \Gamma_n)$ such that either $\Box^b(\bot \leftarrow \top) \in \Gamma_n$ or (Π_n, Γ_n) is locally closed.*

PROOF. By Propositions 5.4 and 5.5. □

[6] Also known as *set-of-support strategy for resolution.*

Note that the above Theorem 5.7 shows that IFT-resolution (which is a restriction of the (general) temporal resolution method in ([Gaintzarain et al. 2013])) is complete for local derivation from TeDiLog clauses. In Example 5.10, we illustrate why next-resolution is necessary for completeness.

Finally, we define the notion of IFT-*derivation* which is based on the notion of local derivation along with a strategy to apply the context-based temporal rules to one selected eventuality. In the sequel, we denote by (Π^*, Γ^*) the last pair of a local derivation starting by (Π, Γ) and such that either $\square^b(\bot \leftarrow \top) \in \Gamma^*$ or (Π^*, Γ^*) is locally closed.

Definition 5.8. Let Π and Γ respectively be a program and a goal, an IFT-*derivation* for Π and Γ, denoted by $\mathcal{D}(\Pi, \Gamma)$, consists of a (possibly infinite) sequence

$$(S_0, \mathsf{sel}_0, \mathsf{sel}_0^*) \Mapsto (S_1, \mathsf{sel}_1, \mathsf{sel}_1^*) \Mapsto (S_2, \mathsf{sel}_2, \mathsf{sel}_2^*) \Mapsto \dots$$

where, for all $j \geq 0$, S_j is a local derivation $(\Pi_j, \Gamma_j) \mapsto \dots \mapsto (\Pi_j^*, \Gamma_j^*)$[7] and $\mathsf{sel}_j \subseteq \mathsf{EventLits}(\Pi_j \cup \Gamma_j)$, such that

(a) $(\Pi_0, \Gamma_0) = (\Pi, \Gamma)$,
(b) if $\mathsf{EventLits}(\Pi_0 \cup \Gamma_0) = \emptyset$ then $\mathsf{sel}_0 = \emptyset$, and otherwise sel_0 contains exactly one eventuality that is fairly selected from $\mathsf{EventLits}(\Pi_0 \cup \Gamma_0)$, and
(c) for all $j \geq 0$ the following four conditions hold:
  (c1) $\Pi_{j+1} = \mathsf{unnext}(\Pi_j^*)$ and $\Gamma_{j+1} = \mathsf{unnext}(\Gamma_j^*)$.
  (c2) If $\mathsf{sel}_j = \emptyset$ then no context-dependent rule is applied in the local derivation S_j.
  (c3) If $\mathsf{sel}_j \neq \emptyset$ then the corresponding context-dependent rule is applied to $\Pi_j \cup \Gamma_j$ and to the eventuality designated by sel_j. Additionally, sel_j^* is the singleton formed by the new eventuality literal that appears in the consequent of the applied context-dependent rule (which involves a fresh variable). Besides, no other context-dependent rule is applied along S_j.
  (c4) If $\mathsf{sel}_j^* \cap \mathsf{EventLits}(\Pi_{j+1} \cup \Gamma_{j+1}) = \emptyset$ then sel_{j+1} is fairly selected from $(\Pi_{j+1}, \Gamma_{j+1})$ else sel_{j+1} takes the value of sel_j^*.

An IFT-derivation which contains a local refutation is called an IFT-refutation.

By the above condition *(c3)*, for example, if $\mathsf{sel}_j = \{p_1 \mathcal{U} p_2\}$ and $\Pi_j \cup \Gamma_j$ contains a (non-empty) set of clauses of the form $\{\square^{b_i}((p_1 \mathcal{U} p_2) \vee H_i \leftarrow B_i) \mid 1 \leq i \leq n\}$, then the context-dependent rule $(\mathcal{U} C_+)$ in Figure 7 is applied to $\Pi_j \cup \Gamma_j$ and $\mathsf{sel}_j^* = \{a \mathcal{U} p_2\}$. Similarly for the rules $(\mathcal{U} C_-)$, $(\mathcal{R} C_+)$, $(\mathcal{R} C_-)$, and the derived context-dependent rules (see Figures 8 and 10).

The fairness in the selection means that for every literal T such that $T \in \mathsf{EventLits}(\Pi_i \cup \Gamma_i)$ for some $i \geq 0$, either $\mathsf{sel}_j = \{T\}$ for some $j \geq i$ or $T \notin \mathsf{EventLits}(\Pi_k \cup \Gamma_k)$ for some $k > i$.

5.2. Examples

In this subsection we present three detailed examples that illustrate the IFT-resolution procedure. In Example 5.9 we simply show how IFT-resolution deals with eventualities. The Example 5.10 illustrates the need of next-resolution (Definition 5.1). Finally, one could think that the selection of an eventuality that cannot be satisfied requires to backtrack for the re-selection of a new eventuality. However, IFT-resolution works without backtracking, as Example 5.11 shows. The three sample derivations are showed in the respective figures, where we indicate which rule is applied and we underline the clauses designated by the rule application, except for the rule (Unx).

Example 5.9. We consider the program $\Pi = \{q \mathcal{U} r \leftarrow \top\}$ and the goal $\Gamma = \{\square(\bot \leftarrow r)\}$. The goal clause is equivalent to the formula $\square \neg r$ and $\Pi \cup \Gamma$ is unsatisfiable. In Figure 13 we show an IFT-refutation for (Π, Γ). First, Π_0^0 and Γ_0^0 are respectively initialized as Π and Γ. Since $q \mathcal{U} r$ is the only eventuality literal in a clause that belongs to $\Pi \cup \Gamma$, it is selected. Therefore $\mathsf{sel}_0 = \{q \mathcal{U} r\}$.

[7]Note that we use two different symbols ($\mapsto$ and $\Mapsto$) to highlight the difference between applying the rule (Unx) and any other rule.

$\Pi_0^0 = \{\underline{q\mathcal{U}\,r \leftarrow \top}\}$	$\Gamma_0^0 = \{\Box\,(\bot \leftarrow r)\}$	$(\mathcal{U}\,C_+)$	$\mathsf{sel}_0 = \{q\mathcal{U}\,r\}$
$\Pi_0^1 = \{\underline{r \vee q \leftarrow \top},$ $r \vee \circ(a\mathcal{U}\,r) \leftarrow \top\}$	$\Gamma_0^1 = \{\underline{\Box\,(\bot \leftarrow r)},$ $\Box\,(\bot \leftarrow a)\}$	(Res)	$\mathsf{sel}_0^* = \{a\mathcal{U}\,r\}$
$\Pi_0^2 = \{\underline{r \vee q \leftarrow \top},$ $r \vee \circ(a\mathcal{U}\,r) \leftarrow \top,$ $\underline{q \leftarrow \top}\}$	$\Gamma_0^2 = \{\Box\,(\bot \leftarrow r),$ $\Box\,(\bot \leftarrow a)\}$	(Sbm)	
$\Pi_0^3 = \{\underline{r \vee \circ(a\mathcal{U}\,r) \leftarrow \top},$ $q \leftarrow \top\}$	$\Gamma_0^3 = \{\underline{\Box\,(\bot \leftarrow r)},$ $\Box\,(\bot \leftarrow a)\}$	(Res)	
$\Pi_0^4 = \{\underline{r \vee \circ(a\mathcal{U}\,r) \leftarrow \top},$ $q \leftarrow \top,$ $\underline{\circ(a\mathcal{U}\,r) \leftarrow \top}\}$	$\Gamma_0^4 = \{\Box\,(\bot \leftarrow r),$ $\Box\,(\bot \leftarrow a)\}$	(Sbm)	
$\Pi_0^5 = \{q \leftarrow \top,$ $\circ(a\mathcal{U}\,r) \leftarrow \top\}$	$\Gamma_0^5 = \{\Box\,(\bot \leftarrow r),$ $\Box\,(\bot \leftarrow a)\}$	(Unx)	
$\Pi_1^0 = \{\underline{a\mathcal{U}\,r \leftarrow \top}\}$	$\Gamma_1^0 = \{\Box\,(\bot \leftarrow r),$ $\Box\,(\bot \leftarrow a)\}$	$(\mathcal{U}\,C_+)$	$\mathsf{sel}_1 = \{a\mathcal{U}\,r\}$
$\Pi_1^1 = \{\underline{r \vee a \leftarrow \top},$ $r \vee \circ(b\mathcal{U}\,r) \leftarrow \top\}$	$\Gamma_1^1 = \{\underline{\Box\,(\bot \leftarrow r)},$ $\Box\,(\bot \leftarrow a),$ $\Box\,(\bot \leftarrow b)\}$	(Res)	$\mathsf{sel}_1^* = \{b\mathcal{U}\,r\}$
$\Pi_1^2 = \{r \vee a \leftarrow \top,$ $r \vee \circ(b\mathcal{U}\,r) \leftarrow \top,$ $\underline{a \leftarrow \top}\}$	$\Gamma_1^2 = \{\Box\,(\bot \leftarrow r),$ $\underline{\Box\,(\bot \leftarrow a)},$ $\underline{\Box\,(\bot \leftarrow b)}\}$	(Res)	
$\Pi_1^3 = \{r \vee a \leftarrow \top,$ $r \vee \circ(b\mathcal{U}\,r) \leftarrow \top,$ $a \leftarrow \top\}$	$\Gamma_1^3 = \{\Box\,(\bot \leftarrow r),$ $\Box\,(\bot \leftarrow a),$ $\Box\,(\bot \leftarrow b),$ $\bot \leftarrow \top\}$		

Fig. 13. IFT-Refutation for $\Pi = \{q\mathcal{U}\,r \leftarrow \top\}$ and $\Gamma = \{\Box\,(\bot \leftarrow r)\}$

We apply the rule $(\mathcal{U}\,C_+)$ to $\Pi_0^0 \cup \Gamma_0^0$ with selected literal $q\mathcal{U}\,r$ and empty context. Hence, we obtain the new program clauses $r \vee q \leftarrow \top$ and $r \vee \circ(a\mathcal{U}\,r) \leftarrow \top$ and the goal clause $\Box\,(\bot \leftarrow a)$, where a is a fresh variable. Since the context is empty (its negation is $\bot$), the goal clause $\Box\,(\bot \leftarrow a)$ gives meaning to a. The new atom $a\mathcal{U}\,r$ is the new selected literal, i.e, $\mathsf{sel}_0^* = \{a\mathcal{U}\,r\}$. Then the resolution rule and the subsumption rule are applied twice, obtaining Π_0^5 and Γ_0^5. Since a refutation cannot be obtained in this state, the application of the rule (Unx) serves to jump to the next state, generating Π_1^0 and Γ_1^0. Since the atom $a\mathcal{U}\,r$ appears as eventuality literal in a clause that belongs to $\Pi_1^0 \cup \Gamma_1^0$, it is kept as selected literal, i.e., $\mathsf{sel}_1 = \{a\mathcal{U}\,r\}$. Now the context-dependent rule $(\mathcal{U}\,C_+)$ is applied to the set $\Pi_1^0 \cup \Gamma_1^0$ with selected literal $a\mathcal{U}\,r$ and empty context. Then, we obtain two new program clauses, $r \vee a \leftarrow \top$ and $r \vee \circ(b\mathcal{U}\,r) \leftarrow \top$, and the goal clause $\Box\,(\bot \leftarrow b)$, where b is a fresh variable. Now $\mathsf{sel}_1^* = \{b\mathcal{U}\,r\}$. Two additional applications of the rule (Res) yield the empty clause $\bot \leftarrow \top$. ■

$\Pi_0^0 = \{q \leftarrow \top,$ $\Box(\circ q \leftarrow q)\}$	$\Gamma_0^0 = \{\underline{\bot \leftarrow \Box q}\}$	$(\Box C_+)$	$\mathsf{sel}_0 = \{\neg\Box q\}$
$\Pi_0^1 = \{\underline{q \leftarrow \top},$ $\Box(\circ q \leftarrow q)\}$	$\Gamma_0^1 = \{\underline{\bot \leftarrow q \wedge \circ(\neg a \mathcal{R} q)},$ $\Box(\bot \leftarrow q \wedge a)\}$	(Res)	$\mathsf{sel}_0^* = \{\neg(\neg a \mathcal{R} q)\}$
$\Pi_0^2 = \{q \leftarrow \top,$ $\Box(\circ q \leftarrow q)\}$	$\Gamma_0^2 = \{\underline{\bot \leftarrow q \wedge \circ(\neg a \mathcal{R} q)},$ $\Box(\bot \leftarrow q \wedge a),$ $\underline{\bot \leftarrow \circ(\neg a \mathcal{R} q)}\}$	(Sbm)	
$\Pi_0^3 = \{\underline{q \leftarrow \top},$ $\Box(\circ q \leftarrow q)\}$	$\Gamma_0^3 = \{\underline{\Box(\bot \leftarrow q \wedge a)},$ $\bot \leftarrow \circ(\neg a \mathcal{R} q)\}$	(Res)	
$\Pi_0^4 = \{\underline{q \leftarrow \top},$ $\underline{\Box(\circ q \leftarrow q)}\}$	$\Gamma_0^4 = \{\Box(\bot \leftarrow q \wedge a),$ $\bot \leftarrow \circ(\neg a \mathcal{R} q),$ $\bot \leftarrow a\}$	(Res)	
$\Pi_0^5 = \{q \leftarrow \top,$ $\Box(\circ q \leftarrow q),$ $\circ q \leftarrow \top\}$	$\Gamma_0^5 = \{\Box(\bot \leftarrow q \wedge a),$ $\bot \leftarrow \circ(\neg a \mathcal{R} q),$ $\bot \leftarrow a\}$	(Unx)	
$\Pi_1^0 = \{\Box(\circ q \leftarrow q),$ $q \leftarrow \top\}$	$\Gamma_1^0 = \{\Box(\bot \leftarrow q \wedge a),$ $\underline{\bot \leftarrow \neg a \mathcal{R} q}\}$	$(\mathcal{R} C_-)$	$\mathsf{sel}_1 = \{\neg(\neg a \mathcal{R} q)\}$
$\Pi_1^1 = \{\Box(\circ q \leftarrow q),$ $\underline{q \leftarrow \top},$ $a \leftarrow q,$ $\Box(a \leftarrow b)\}$	$\Gamma_1^1 = \{\underline{\Box(\bot \leftarrow q \wedge a)},$ $\bot \leftarrow q \wedge \circ(\neg b \mathcal{R} q),$ $\Box(\bot \leftarrow q \wedge b)\}$	(Res)	$\mathsf{sel}_1^* = \{\neg(\neg b \mathcal{R} q)\}$
$\Pi_1^2 = \{\Box(\circ q \leftarrow q),$ $q \leftarrow \top,$ $\underline{a \leftarrow q},$ $\underline{\Box(a \leftarrow} b)\}$	$\Gamma_1^2 = \{\Box(\bot \leftarrow q \wedge a),$ $\bot \leftarrow q \wedge \circ(\neg b \mathcal{R} q),$ $\Box(\bot \leftarrow q \wedge b),$ $\underline{\bot \leftarrow a}\}$	(Res)	
$\Pi_1^3 = \{\Box(\circ q \leftarrow q),$ $\underline{q \leftarrow \top},$ $a \leftarrow q,$ $\Box(a \leftarrow b)\}$	$\Gamma_1^3 = \{\Box(\bot \leftarrow q \wedge a),$ $\bot \leftarrow q \wedge \circ(\neg b \mathcal{R} q),$ $\Box(\bot \leftarrow q \wedge b),$ $\bot \leftarrow a,$ $\underline{\bot \leftarrow q}\}$	(Res)	
$\Pi_1^4 = \{\Box(\circ q \leftarrow q),$ $q \leftarrow \top,$ $a \leftarrow q,$ $\Box(a \leftarrow b)\}$	$\Gamma_1^4 = \{\Box(\bot \leftarrow q \wedge a),$ $\bot \leftarrow q \wedge \circ(\neg b \mathcal{R} q),$ $\Box(\bot \leftarrow q \wedge b),$ $\bot \leftarrow a,$ $\bot \leftarrow q,$ $\bot \leftarrow \top\}$		

Fig. 14. IFT-refutation for $\Pi = \{q \leftarrow \top, \Box(\circ q \leftarrow q)\}$ and $\Gamma = \{\bot \leftarrow \Box q\}$

In the next example we illustrate why *next-resolution* (Definition 5.1) is necessary for completeness.

Example 5.10. Let us consider the program $\Pi = \{q \leftarrow \top, \Box(\circ q \leftarrow q)\}$ and the goal $\Gamma = \{\bot \leftarrow \Box q\}$. The set of clauses $\Pi \cup \Gamma$ is unsatisfiable. The IFT-refutation for (Π, Γ) is shown in Figure 14. The goal clause $\bot \leftarrow \Box q$ contains the only eventuality literal, $\neg\Box q$, in (Π, Γ). Hence $\mathsf{sel}_0 =$

$\{\neg\Box\, q\}$ and the application of the rule $(\Box\, C_+)$ with context $\{q \leftarrow \top\}$ generates the goal clauses $\bot \leftarrow q \wedge \circ(\neg a\,\mathcal{R}\, q)$ and $\Box\,(\bot \leftarrow q \wedge a)$, where a is a new propositional variable. Additionally, we have that $\mathsf{sel}_0^* = \{\neg(\neg a\,\mathcal{R}\, q)\}$. By applying the resolution rule to the program clause $q \leftarrow \top$ and the goal clause $\bot \leftarrow q \wedge \circ(\neg a\,\mathcal{R}\, q)$, the goal clause $\bot \leftarrow \circ(\neg a\,\mathcal{R}\, q)$ is obtained as resolvent. Then, by (Sbm), the goal clause $\bot \leftarrow q \wedge \circ(\neg a\,\mathcal{R}\, q)$ is subsumed by $\bot \leftarrow \circ(\neg a\,\mathcal{R}\, q)$. The second application of (Res), this time with the program clause $q \leftarrow \top$ and the goal clause $\Box\,(\bot \leftarrow q \wedge a)$ as premises, yields the goal clause $\bot \leftarrow a$. Then the rule (Res) is applied to the two program clauses $q \leftarrow \top$ and $\Box\,(\circ q \leftarrow q)$ and the program clause $\circ q \leftarrow \top$ is obtained as resolvent before jumping to the next state by applying the rule (Unx) to (Π_0^5, Γ_0^5).

Remark. Note that (Π_0^5, Γ_0^5) is obtained by next-resolution from (Π_0^4, Γ_0^4). Let us explain that this step is essential. In (Π_0^4, Γ_0^4) goal-resolution is not applicable. If instead of applying next-resolution to the clauses $q \leftarrow \top$ and $\Box\,(\circ q \leftarrow q)$ in Π_0^4, we applied the rule (Unx) to the pair (Π_0^4, Γ_0^4), then we would obtain the program $\Pi' = \{\Box\,(\circ q \leftarrow q)\}$ and the goal $\Gamma' = \{\bot \leftarrow \neg a\,\mathcal{R}\, q, \Box\,(\bot \leftarrow a)\}$. Since $\Pi' \cup \Gamma'$ is satisfiable, the refutation of $\Pi \cup \Gamma$ would never be found.

By applying the rule (Unx) to (Π_0^5, Γ_0^5), we obtain the pair (Π_1^0, Γ_1^0). Then, we apply the context-dependent rule $(\mathcal{R}\, C_-)$ with respect to the selected eventuality literal $\neg(\neg a\,\mathcal{R}\, q)$ and the clause $q \leftarrow \top$ as context. The pair (Π_1^1, Γ_1^1) is obtained by replacing the goal clause $\bot \leftarrow \neg a\,\mathcal{R}\, q$ in Γ_1^0 with the program clauses $a \leftarrow q$ and $\Box\,(a \leftarrow b)$ and the goal clauses $\bot \leftarrow q \wedge \circ(\neg b\,\mathcal{R}\, q)$ and $\Box\,(\bot \leftarrow q \wedge b)$, where b is a fresh propositional variable. The value of sel_1^* is $\{\neg(\neg b\,\mathcal{R}\, q)\}$. In (Π_1^1, Γ_1^1) the resolution rule is applied with the program clause $q \leftarrow \top$ and the goal clause $\Box\,(\bot \leftarrow q \wedge a)$ as premises, obtaining the goal clause $\bot \leftarrow a$ as resolvent. In (Π_1^2, Γ_1^2) the resolution between the program clause $a \leftarrow q$ and the goal clause $\bot \leftarrow a$ yields the goal clause $\bot \leftarrow q$. Finally, since Π_1^3 contains the program clause $q \leftarrow \top$ and Γ_1^3 contains the goal clause $\bot \leftarrow q$, the empty clause is obtained by applying the resolution rule (Res) to these two clauses. ∎

One could think that if there are more than one eventuality literal that can be (fairly) selected to apply the corresponding context-dependent rule, then it could be that not all of the eventualities were right choices (e.g. because the program prevents the satisfaction of some of them). This view leads to the idea that wrong choices will have to be repaired by backtracking to the choice point and changing the selection. Moreover, sometimes one eventuality φ must be necessarily fulfilled before another eventuality ψ. In those cases, one could think that selecting ψ before selecting φ could end up requiring backtracking. In the next example we illustrate that IFT-resolution does not need backtracking (independently of the selection strategy).

Example 5.11. We consider the program $\Pi = \{\Diamond\, q \leftarrow \top, \Diamond\, r \leftarrow \top\}$ and the goal $\Gamma = \{\Box\,(\bot \leftarrow q \wedge \Diamond\, r)\}$. It is easy to see that $\Pi \cup \Gamma$ is satisfiable. There are two eventualities, $\Diamond\, q$ and $\Diamond\, r$, that must be fulfilled, but the goal clause states that once the eventuality $\Diamond\, q$ is fulfilled, the eventuality $\Diamond\, r$ cannot be fulfilled. However, if we first select $\Diamond\, q$, it does not mean that $\Diamond\, q$ is fulfilled before $\Diamond\, r$ is fulfilled. Actually, since $\Diamond\, r$ must be fulfilled before $\Diamond\, q$, that is what happens. The corresponding infinite IFT-derivation is shown in detail in Figures 15, 16 and 17 (it is split due to space reasons).

After the first selection, $\mathsf{sel}_0 = \{\Diamond\, q\}$. Then the application of the rule $(\Diamond\, C_+)$ with context $\{\Diamond\, r \leftarrow \top\}$ generates the program clause $q \vee \circ(a\,\mathcal{U}\, q) \leftarrow \top$ and the goal clause $\Box\,(\bot \leftarrow a \wedge \Diamond\, r)$ where a is a fresh propositional variable. Additionally the value of sel_0^* is set to $\{a\,\mathcal{U}\, q\}$. Then, several rule applications yield the locally closed pair $(\Pi_0^{12}, \Gamma_0^{12})$ (Definition 5.6). Next, by rule (Unx), the pair (Π_1^0, Γ_1^0) is generated. Since the atom $a\,\mathcal{U}\, q$ belongs to $\mathsf{EventLits}(\Pi_1^0 \cup \Gamma_1^0)$, it remains as the selected literal and, consequently, the rule $(\mathcal{U}\, C_+)$ is applied to $(\Pi_1^0 \cup \Gamma_1^0)$ with $a\,\mathcal{U}\, q$ as selected literal (i.e., $\mathsf{sel}_1 = \{a\,\mathcal{U}\, q\}$) and with empty context, obtaining the pair (Π_1^1, Γ_1^1) and setting sel_1^* to $\{b\,\mathcal{U}\, q\}$, where b is a fresh propositional variable. Then the locally closed pair (Π_1^7, Γ_1^7) is obtained from (Π_1^1, Γ_1^1) by means of several applications of the rule (Res) and the rule (Sbm). The pair (Π_2^0, Γ_2^0) is obtained from (Π_1^7, Γ_1^7) by using the rule (Unx). Since the set $\mathsf{EventLits}(\Pi_2^0 \cup \Gamma_2^0)$ is empty, the value of sel_2 as well as the value of sel_2^* is the empty

$\Pi_0^0 = \{\underline{\Diamond q \leftarrow \top}, \Diamond r \leftarrow \top\}$	$\Gamma_0^0 = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$	$(\Diamond C_+)$	$\mathsf{sel}_0 = \{\Diamond q\}$
$\Pi_0^1 = \{\underline{\Diamond r \leftarrow \top}, q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top\}$	$\Gamma_0^1 = \{\Box(\bot \leftarrow q \wedge \Diamond r), \Box(\bot \leftarrow a \wedge \Diamond r)\}$	$(\Diamond H_+)$	$\mathsf{sel}_0^* = \{a\,\mathcal{U}\,q\}$
$\Pi_0^2 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, r \vee \circ\Diamond r \leftarrow \top\}$	$\Gamma_0^2 = \{\underline{\Box(\bot \leftarrow q \wedge \Diamond r)}, \underline{\Box(\bot \leftarrow a \wedge \Diamond r)}\}$	$(\Diamond B_+)$	
$\Pi_0^3 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, r \vee \circ\Diamond r \leftarrow \top\}$	$\Gamma_0^3 = \{\underline{\Box(\bot \leftarrow a \wedge \Diamond r)}, \Box(\bot \leftarrow q \wedge r), \Box(\bot \leftarrow q \wedge \circ\Diamond r)\}$	$(\Diamond B_+)$	
$\Pi_0^4 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, \underline{r \vee \circ\Diamond r \leftarrow \top}\}$	$\Gamma_0^4 = \{\Box(\bot \leftarrow q \wedge r), \underline{\Box(\bot \leftarrow q \wedge \circ\Diamond r)}, \Box(\bot \leftarrow a \wedge r), \Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$	(Res)	
$\Pi_0^5 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, r \vee \circ\Diamond r \leftarrow \top, \underline{r \leftarrow q}\}$	$\Gamma_0^5 = \{\underline{\Box(\bot \leftarrow q \wedge r)}, \Box(\bot \leftarrow q \wedge \circ\Diamond r), \Box(\bot \leftarrow a \wedge r), \Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$	(Res)	
$\Pi_0^6 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, \underline{r \vee \circ\Diamond r \leftarrow \top}, r \leftarrow q\}$	$\Gamma_0^6 = \{\Box(\bot \leftarrow q \wedge r), \Box(\bot \leftarrow q \wedge \circ\Diamond r), \Box(\bot \leftarrow a \wedge r), \underline{\Box(\bot \leftarrow a \wedge \circ\Diamond r)}, \bot \leftarrow q\}$	(Res)	
$\Pi_0^7 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, r \vee \circ\Diamond r \leftarrow \top, r \leftarrow q, \underline{r \leftarrow a}\}$	$\Gamma_0^7 = \{\Box(\bot \leftarrow q \wedge r), \Box(\bot \leftarrow q \wedge \circ\Diamond r), \underline{\Box(\bot \leftarrow a \wedge r)}, \Box(\bot \leftarrow a \wedge \circ\Diamond r), \bot \leftarrow q\}$	(Res)	
$\Pi_0^8 = \{\underline{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top}, r \vee \circ\Diamond r \leftarrow \top, r \leftarrow q, r \leftarrow a\}$	$\Gamma_0^8 = \{\Box(\bot \leftarrow q \wedge r), \Box(\bot \leftarrow q \wedge \circ\Diamond r), \Box(\bot \leftarrow a \wedge r), \Box(\bot \leftarrow a \wedge \circ\Diamond r), \underline{\bot \leftarrow q}, \bot \leftarrow a\}$	(Res)	
$\Pi_0^9 = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top, r \vee \circ\Diamond r \leftarrow \top, \underline{r \leftarrow q}, r \leftarrow a, \circ(a\,\mathcal{U}\,q) \leftarrow \top\}$	$\Gamma_0^9 = \{\Box(\bot \leftarrow q \wedge r), \Box(\bot \leftarrow q \wedge \circ\Diamond r), \Box(\bot \leftarrow a \wedge r), \Box(\bot \leftarrow a \wedge \circ\Diamond r), \underline{\bot \leftarrow q}, \bot \leftarrow a\}$	(Sbm)	

Fig. 15. IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ (Part 1 of 3)

$\Pi_0^{10} = \{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top,\ r \vee \circ\Diamond r \leftarrow \top,\ \underline{r \leftarrow a},\ \circ(a\,\mathcal{U}\,q) \leftarrow \top\}$	$\Gamma_0^{10} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \bot \leftarrow q,\ \underline{\bot \leftarrow a}\}$	(Sbm)	
$\Pi_0^{11} = \{\underline{q \vee \circ(a\,\mathcal{U}\,q) \leftarrow \top},\ r \vee \circ\Diamond r \leftarrow \top,\ \underline{\circ(a\,\mathcal{U}\,q) \leftarrow \top}\}$	$\Gamma_0^{11} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \bot \leftarrow q,\ \bot \leftarrow a\}$	(Sbm)	
$\Pi_0^{12} = \{r \vee \circ\Diamond r \leftarrow \top,\ \circ(a\,\mathcal{U}\,q) \leftarrow \top\}$	$\Gamma_0^{12} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \bot \leftarrow q,\ \bot \leftarrow a\}$	(Unx)	
$\Pi_1^{0} = \{\underline{a\,\mathcal{U}\,q \leftarrow \top}\}$	$\Gamma_1^{0} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r)\}$	$(\mathcal{U}C_+)$	$\mathsf{sel}_1 = \{a\,\mathcal{U}\,q\}$
$\Pi_1^{1} = \{\underline{q \vee a \leftarrow \top},\ q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top\}$	$\Gamma_1^{1} = \{\underline{\Box(\bot \leftarrow q \wedge r)},\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \Box(\bot \leftarrow b)\}$	(Res)	$\mathsf{sel}_1^* = \{b\,\mathcal{U}\,q\}$
$\Pi_1^{2} = \{q \vee a \leftarrow \top,\ q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top,\ \underline{a \leftarrow r}\}$	$\Gamma_1^{2} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \underline{\Box(\bot \leftarrow a \wedge r)},\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \Box(\bot \leftarrow b)\}$	(Res)	
$\Pi_1^{3} = \{\underline{q \vee a \leftarrow \top},\ \underline{q \vee \circ(b\,\mathcal{U}\,q)} \leftarrow \top,\ a \leftarrow r\}$	$\Gamma_1^{3} = \{\Box(\bot \leftarrow q \wedge r),\ \underline{\Box(\bot \leftarrow q \wedge \circ\Diamond r)},\ \Box(\bot \leftarrow a \wedge r),\ \Box(\bot \leftarrow a \wedge \circ\Diamond r),\ \Box(\bot \leftarrow b),\ \bot \leftarrow r\}$	(Res)	
$\Pi_1^{4} = \{q \vee a \leftarrow \top,\ q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top,\ a \leftarrow r,\ \underline{a \leftarrow \circ\Diamond r}\}$	$\Gamma_1^{4} = \{\Box(\bot \leftarrow q \wedge r),\ \Box(\bot \leftarrow q \wedge \circ\Diamond r),\ \Box(\bot \leftarrow a \wedge r),\ \underline{\Box(\bot \leftarrow a \wedge \circ\Diamond r)},\ \Box(\bot \leftarrow b),\ \bot \leftarrow r\}$	(Res)	

Fig. 16. IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ (Part 2 of 3)

$$\begin{array}{llll}
\Pi_1^5 = \{q \vee a \leftarrow \top, & \Gamma_1^5 = \{\Box(\bot \leftarrow q \wedge r), & (Sbm) & \\
\quad q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top, & \quad \Box(\bot \leftarrow q \wedge \circ\Diamond r), & & \\
\quad \underline{a \leftarrow r}, & \quad \Box(\bot \leftarrow a \wedge r), & & \\
\quad a \leftarrow \circ\Diamond r\} & \quad \Box(\bot \leftarrow a \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow b), & & \\
 & \quad \underline{\bot \leftarrow r}, & & \\
 & \quad \bot \leftarrow \circ\Diamond r\} & &
\end{array}$$

$$\begin{array}{llll}
\Pi_1^6 = \{q \vee a \leftarrow \top, & \Gamma_1^6 = \{\Box(\bot \leftarrow q \wedge r), & (Sbm) & \\
\quad q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top, & \quad \Box(\bot \leftarrow q \wedge \circ\Diamond r), & & \\
\quad \underline{a \leftarrow \circ\Diamond r}\} & \quad \Box(\bot \leftarrow a \wedge r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow b), & & \\
 & \quad \bot \leftarrow r, & & \\
 & \quad \underline{\bot \leftarrow \circ\Diamond r}\} & &
\end{array}$$

$$\begin{array}{llll}
\Pi_1^7 = \{q \vee a \leftarrow \top, & \Gamma_1^7 = \{\Box(\bot \leftarrow q \wedge r), & (Unx) & \\
\quad q \vee \circ(b\,\mathcal{U}\,q) \leftarrow \top\} & \quad \Box(\bot \leftarrow q \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow b), & & \\
 & \quad \bot \leftarrow r, & & \\
 & \quad \bot \leftarrow \circ\Diamond r\} & &
\end{array}$$

$$\begin{array}{llll}
\Pi_2^0 = \emptyset & \Gamma_2^0 = \{\Box(\bot \leftarrow q \wedge r), & (\Diamond B_+) & \mathsf{sel}_2 = \emptyset \\
 & \quad \Box(\bot \leftarrow q \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow b), & & \\
 & \quad \underline{\bot \leftarrow \Diamond r}\} & &
\end{array}$$

$$\begin{array}{llll}
\Pi_2^1 = \emptyset & \Gamma_2^1 = \{\Box(\bot \leftarrow q \wedge r), & (Unx) & \mathsf{sel}_2^* = \emptyset \\
 & \quad \Box(\bot \leftarrow q \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge r), & & \\
 & \quad \Box(\bot \leftarrow a \wedge \circ\Diamond r), & & \\
 & \quad \Box(\bot \leftarrow b), & & \\
 & \quad \bot \leftarrow r, & & \\
 & \quad \bot \leftarrow \circ\Diamond r\} & &
\end{array}$$

$$\begin{array}{ll}
\vdots & \vdots
\end{array}$$

$\Pi_2^0 = \Pi_j^0, \Gamma_2^0 = \Gamma_j^0, \Pi_2^1 = \Pi_j^1, \Gamma_2^1 = \Gamma_j^1$ and

$\mathsf{sel}_j = \mathsf{sel}_j^* = \emptyset$ for every $j \geq 3$

Fig. 17. IFT-derivation for $\Pi = \{\Diamond q \leftarrow \top, \Diamond r \leftarrow \top\}$ and $\Gamma = \{\Box(\bot \leftarrow q \wedge \Diamond r)\}$ (Part 3 of 3)

set. Therefore no context-dependent rule is applied to (Π_2^0, Γ_2^0) and we get the locally closed pair (Π_2^1, Γ_2^1) by applying the context-free rule $(\Diamond B_+)$. From that point onwards the derivation is a repetition where $\Pi_j^0 = \Pi_2^0$, $\Gamma_j^0 = \Gamma_2^0$, $\Pi_2^1 = \Pi_j^1$, $\Gamma_2^1 = \Gamma_j^1$ and $\mathsf{sel}_j = \mathsf{sel}_j^* = \emptyset$ for every $j \geq 3$. From the pairs $(\Pi_0^{12}, \Gamma_0^{12}), (\Pi_1^7, \Gamma_1^7), (\Pi_2^1, \Gamma_2^1), (\Pi_3^1, \Gamma_3^1), \dots$ we can deduce that the literals that are forced by this infinite derivation at states $s_0, s_1, s_2, s_3, \dots$ are, respectively, $\{\neg q, r, \neg a\}, \{q, \neg r, \neg b\}, \{\neg r, \neg b\}, \{\neg r, \neg b\}, \dots$. Hence, a model $\mathcal{M}$ can be built from this infinite derivation by making $V_{\mathcal{M}}(s_0) = \{r\}$, $V_{\mathcal{M}}(s_1) = \{q\}$ and $V_{\mathcal{M}}(s_j) = \emptyset$ for every $j \geq 2$. ■

In Example 5.11 one can see that the strategy for selecting eventualities does not compromise the completeness of IFT-resolution. However it can affect efficiency. In particular, if we had selected the eventuality $\Diamond r$ instead of the eventuality $\Diamond q$, the derivation would have been considerably longer.

5.3. Logical Semantics

In this subsection we define the logical characterization of the declarative meaning of TeDiLog programs.

In classical LP (see e.g. [Lloyd 1984]), and also in some extensions like Templog ([Baudinet 1989b]) and Chronolog ([Wadge 1988; Orgun et al. 1993; Orgun 1995]), the declarative meaning of a program is formalized in three equivalent ways:

(1) Logically, as the set of bodies that are logical consequences of the program.
(2) Model-theoretically, by means of the minimal model of the program.
(3) By fixpoint characterization, based on the immediate consequence operator.

These three formalizations are equivalent in the sense that, on one hand, the bodies that are logical consequences of the program are just the bodies that are satisfied by the minimal model of the program and, on the other hand, the minimal model of the program is the fixpoint of the immediate consequence operator.

In DLP ([Lobo et al. 1992]), and existing temporal extensions of DLP ([Gergatsoulis et al. 2000]), where a goal is of the form $\{\bot \leftarrow B_1, \dots, \bot \leftarrow B_n\}$, the logical characterization of the declarative meaning of a program is provided by the set of formulas of the form $B_1 \vee \dots \vee B_n$ (i.e. disjunctions of bodies) that are logical consequences of the program. The model-theoretic characterization is provided by means of all the minimal models (in general there is no only one minimal model). The fixpoint characterization can also be extended to the disjunctive paradigm –as shown in [Lobo et al. 1992; Gergatsoulis et al. 2000]– yielding the set of disjunctions of atoms that are true in all the minimal models.

In TeDiLog a goal $\Gamma = \{\Box^{b_1}(\bot \leftarrow B_1), \dots, \Box^{b_n}(\bot \leftarrow B_n)\}$ is understood as the conjunction of the goal clauses in Γ. Since a goal clause $\Box^{b}(\bot \leftarrow B)$ represents the formula $\neg\Diamond^{b} B$, the set Γ is logically equivalent to the formula $\neg\Diamond^{b_1} B_1 \wedge \dots \wedge \neg\Diamond^{b_n} B_n$ or equivalently to $\neg(\Diamond^{b_1} B_1 \vee \dots \vee \Diamond^{b_n} B_n)$.

Definition 5.12. The declarative semantics of a program Π is logically characterized as the set of all the formulas of the form $\Diamond^{b_1} B_1 \vee \dots \vee \Diamond^{b_n} B_n$ that are logical consequences of Π.

5.4. Equivalence between operational and logical semantics

In this subsection we address the soundness and completeness of IFT-resolution with respect to the logical semantics of TeDiLog.

Soundness is a consequence of the fact that each rule preserves satisfiability (indeed, some of them preserve logical equivalence).

THEOREM 5.13. *If there exists an* IFT-*refutation from* Π *with top-goal* Γ, *then* $\Pi \cup \Gamma$ *is unsatisfiable.*

PROOF. If $\Box^b(\bot \leftarrow \top) \in \Gamma'$ for some (Π', Γ') in an IFT-derivation from (Π, Γ), then $\Pi' \cup \Gamma'$ is unsatisfiable. Therefore, since the rule (Unx) preserves satisfiability and the initial set and the target set of every application of the remaining rules are equisatisfiable, $\Pi \cup \Gamma$ is also unsatisfiable. $\square$

For more details about the proof of the above theorem see Section 6 in [Gaintzarain et al. 2013].
TeDiLog's completeness means that whenever a set of clauses $\Pi \cup \Gamma$ is unsatisfiable, the IFT-resolution gives a refutation for (Π, Γ).

PROPOSITION 5.14. *If $\Pi \cup \Gamma$ is unsatisfiable then* IFT*-resolution produces a refutation.*

PROOF. Since the resolution system TRS is complete, the systematic algorithm $\mathcal{SR}$ for TRS-resolution ([Gaintzarain et al. 2013]) produces a refutation for $\Pi \cup \Gamma$. By Propositions 5.4 and 5.5, goal- and next-resolution are enough to obtain all the clauses of the form $\Box^b(\circ H \leftarrow \circ B)$. Additionally, the clauses obtained by TRS-resolution steps that are not goal- or next-resolution, do not belong to the context in the further applications of the context-dependent rules. Consequently, any refutation produced by the systematic algorithm $\mathcal{SR}$ can be transformed into an IFT-refutation by removing, on one hand, the applications of the rule (Res) that are not goal- or next-resolutions and, on the other hand, the applications of the rule (Sbm) that involve clauses generated by the applications of the rule (Res) that are not goal- or next-resolutions. $\square$

5.5. Fixpoint semantics

In this subsection we provide a fixpoint semantics for TeDiLog. The operational semantics of TeDiLog is based on the set of support strategy and N-resolution (see Subsection 5.1), whereas the fixpoint semantics introduced in this subsection is based on the set of support strategy and P-resolution. Indeed, the fixpoint semantics is a bottom-up approach that is obtained as the reverse of the resolution procedure underlying the operational semantics, which is a top-down approach.

Definition 5.15. An application of the rule (Res) to two program clauses is called a *reverse* TeDiLog*-resolution* whenever at least one of the input clauses is of the form $\Box^b(H \leftarrow \circ B)$. In particular, it is called a *reverse goal-resolution* if B is empty. Otherwise, it is called a *reverse next-resolution*.

It is easy to see that the above applications of (Res) are reverse of the applications in Definition 5.1.

In the rest of this subsection, we define the notion of *reverse* IFT*-derivation*. For that, we first introduce local reverse derivations and prove some useful properties of them.

Definition 5.16. A *local reverse derivation* for a program Π is a finite sequence $\Pi_0, \mapsto \Pi_1 \mapsto \ldots \mapsto \Pi_n$ such that $\Pi_0 = \Pi$ and for all $j \in \{1, \ldots, n\}$ the set Π_j is obtained from Π_{j-1} either

(a) by the application of a context-free temporal rule, or
(b) by a reverse TeDiLog-resolution to two clauses in Π_{j-1} whose resolvent is not in Π_{j-1}.

PROPOSITION 5.17. *A local reverse derivation for a program never contains a goal clause.*

PROOF. It is obvious that context-free temporal rules do not produce goal clauses from program clauses. Additionally, reverse TeDiLog-resolution cannot generate a goal clause from program clauses because two non-empty heads produce a non-empty head. $\square$

In order to formulate a completeness result for local reverse derivations, we first define the notion of reversely closed program.

Definition 5.18. We say that a program Π is *reversely closed* if and only if it satisfies the following two conditions:

(a) The clauses in Π are exclusively formed by atoms in Prop and atoms of the form $\circ A$.

$$\begin{array}{rl}
\Pi_0 = & \{\Box\,(\circ p \leftarrow p), p \leftarrow \top, z \vee \circ(r\,\mathcal{U}\,q) \leftarrow \circ^5\Box\,p, (\neg s)\,\mathcal{R}\,v \leftarrow \circ\Diamond\,q\} \\
\vdots & \\
\Pi_0^* = & \{\Box\,(\circ p \leftarrow p), p \leftarrow \top, z \vee \circ(r\,\mathcal{U}\,q) \leftarrow \circ^5\Box\,p, v \leftarrow \circ\Diamond\,q, \circ((\neg s)\,\mathcal{R}\,v) \leftarrow s \wedge \circ\Diamond\,q, \\
& \circ p \leftarrow \top\} \\
\Pi_1 = & \{\Box\,(\circ p \leftarrow p), p \leftarrow \top\} \\
\vdots & \\
\Pi_1^* = & \{\Box\,(\circ p \leftarrow p), p \leftarrow \top, \circ p \leftarrow \top\}
\end{array}$$

Fig. 18. Reverse IFT-derivation for $\Pi = \{\Box\,(\circ p \leftarrow p), p \leftarrow \top, z \vee \circ(r\,\mathcal{U}\,q) \leftarrow \circ^5\Box\,p, (\neg s)\,\mathcal{R}\,v \leftarrow \circ\Diamond\,q\}$

(b) Every clause that can be obtained by applying reverse TeDiLog-resolution (that is, reverse goal- or reverse next-resolution) is already in Π.

It is worth noting that the subsumption rule is not involved in the previous definition.

As well as goal- and next-resolution, their reverse versions are also particular forms of the set-of-support restriction of resolution. Indeed, they can be seen as P-restrictions of resolution.

PROPOSITION 5.19. *For any program* Π_0*, there exists a local reverse derivation* $\Pi_0 \mapsto \ldots \mapsto \Pi_n$ *such that* Π_n *is reversely closed.*

PROOF. By Proposition 5.17, the empty clause cannot be generated. Hence, it suffices to prove that reverse IFT-resolution allows us to produce all the clauses of the form $\Box^b(H \leftarrow \top)$ and $\Box^b(\circ H \leftarrow \circ B)$ that can be generated from Π_0 by using the context-free temporal rules and the rule (Res) (without any restriction). First, we can obtain, by reverse goal-resolution, the minimal clauses of the form $\Box^b(H \leftarrow B)$, where H and B may contain propositional atoms. This is a direct consequence of the completeness of P-restriction of resolution (see e.g. Section 2.6 in [Schöning 1989]), along with the fact that temporal clauses (in local reverse derivations) can be seen as classical propositional clauses (see Definition 24 and Proposition 25 in [Gaintzarain et al. 2013]). Second, using reverse next-resolution we can obtain all the clauses of the form $\Box^b(\circ H \leftarrow \circ B)$. Here we use the completeness of P-restriction of resolution, by considering that a clause is positive whenever the body does not contain propositional atoms (in particular a clause of the form $\Box^b(\circ H \leftarrow \circ B)$ is positive). □

Finally, we define the notion of *reverse* IFT*-derivation* which is based on the notion of local reverse derivation along with a condition to stop when a particular kind of cycle is detected. In the sequel, we denote by Π^* the last set of a local reverse derivation starting by Π and such that Π^* is reversely closed.

Definition 5.20. Let Π be a program, a *reverse* IFT*-derivation* for Π consists of a finite sequence

$$S_0 \Mapsto S_1 \Mapsto \ldots \Mapsto S_j \Mapsto \ldots \Mapsto S_k$$

where, for all $i \in \{0, \ldots, k\}$, S_i is a local reverse derivation $\Pi_i \mapsto \ldots \mapsto \Pi_i^*$ such that

(a) $\Pi_0 = \Pi$,
(b) for all $i \in \{0, \ldots, k-1\}$, $\Pi_{i+1} = \mathsf{unnext}(\Pi_i^*)$
(c) $\mathsf{unnext}(\Pi_k^*) = \Pi_j$ for some $j \in \{0, \ldots, k\}$.

Example 5.21. Let Π be the program

$$\{\Box\,(\circ p \leftarrow p), p \leftarrow \top, z \vee \circ(r\,\mathcal{U}\,q) \leftarrow \circ^5\Box\,p, (\neg s)\,\mathcal{R}\,v \leftarrow \circ\Diamond\,q\}$$

In Figure 18 we depict the first and last program of each local reverse derivation in the reverse IFT-derivation for Π. Note that $\mathsf{unnext}(\Pi_1^*) = \Pi_1$. ■

PROPOSITION 5.22. *For any program* Π*, there exists a reverse* IFT*-derivation for* Π.

PROOF. On one hand, Proposition 5.19 ensures the existence of the successive local reverse derivations S_0, S_1, etc. On the other hand, the existence of a cycle is ensured because the set of different clauses that can be produced is finite. In particular, the number of different clauses is bounded by 2^n where n is linear on the number of atoms (including sub-atoms) that occur in Π. $\square$

In order to define a fixpoint semantics for any program Π, we are going to introduce an operator T_Π based on the notion of reverse IFT-derivation for Π. Since the logical semantics of TeDiLog relies on formulas of the form $\diamond(B_1 \vee \ldots \vee B_n) \vee B_{n+1} \vee \ldots \vee B_m$, the operator T_Π will allow us to obtain their conjunctive normal forms. These are formulas of the form $\diamond(H_1 \wedge \ldots \wedge H_n) \vee H$, which in particular could be H and $\diamond(H_1 \wedge \ldots \wedge H_n)$. Next, we consequently define the temporal disjunctive Herbrand base for a program.

Definition 5.23. Let Π be a program. The temporal Herbrand base THB_Π is the set of all the atoms whose propositional sub-atoms appear in Π. The temporal disjunctive Herbrand base TDHB_Π is the set of all the formulas of the form H, $\diamond(H_1 \wedge \ldots \wedge H_n) \vee H$ and $\diamond(H_1 \wedge \ldots \wedge H_n)$ where $n \geq 1$ and $H_1, \ldots, H_n, H$ are non-empty heads whose atoms belong to THB_Π.

The operator T_Π is based on the following notion of basic logical consequences of a head H, denoted by $\mathsf{basic}(H)$, that are obtained by direct substitutions of atoms. For that, we use the auxiliary notion of basic logical consequence of an atom A, denoted by $\mathsf{basic}(A)$.

Definition 5.24. Let $A \in \mathrm{THB}_\Pi$, $\mathsf{basic}(A)$ is defined as follows:

(1) $\mathsf{basic}(\circ^g \square p) = \{\circ^g p, \circ^{g+1}\square p\} \cup \{\circ^g(L\,\mathcal{R}\,p) \mid \circ^g(L\,\mathcal{R}\,p) \in \mathrm{THB}_\Pi\}$
(2) $\mathsf{basic}(\circ^g p) = \{\circ^g(L\,\mathcal{U}\,p) \mid \circ^g(L\,\mathcal{U}\,p) \in \mathrm{THB}_\Pi\}$
(3) $\mathsf{basic}(\circ^g \diamond p) = \{\circ^{g-1}\diamond p \mid g-1 \geq 0\} \cup \{\circ^g p \vee \circ^{g+1}\diamond p\}$
(4) $\mathsf{basic}(\circ^g(L\,\mathcal{U}\,p)) = \{\circ^g \diamond p, \circ^g(p \vee L), \circ^g(p \vee \circ(L\,\mathcal{U}\,p))\}$
(5) $\mathsf{basic}(\circ^g(L\,\mathcal{R}\,p)) = \{\circ^g p, \circ^g(L \vee \circ(L\,\mathcal{R}\,p))\}$

where, in (4) and (5), if L is a negative literal of the form $\neg q$ then $\circ^g(p \vee L)$ is the program clause $\circ^g(p \leftarrow q)$ and $\circ^g(L \vee \circ(L\,\mathcal{R}\,p))$ is the program clause $\circ^g(\circ(\neg q\,\mathcal{R}\,p) \leftarrow q)$.

Let $H \in \mathrm{TDHB}_\Pi$, $\mathsf{basic}(H)$ is the smallest set such that $H \in \mathsf{basic}(H)$ and, if a head $A' \vee H'$ belongs to $\mathsf{basic}(H)$ then $H'' \vee H' \in \mathsf{basic}(H)$ for every $H'' \in \mathsf{basic}(A')$.
Let Λ be a set of heads, $\mathsf{basic}(\Lambda) = \bigcup_{H \in \Lambda} \mathsf{basic}(H)$.

It is worth noting that the above definition explicitly mentions heads, because $\mathsf{basic}(H)$ can also contain program clauses with non-empty bodies (which are not heads). Indeed, for the sake of simplicity, in the rest of this subsection we consider sets containing program clauses and heads. However, the program clause $H \leftarrow \top$ and the head H are syntactically distinct (although semantically equivalent).

Next, we introduce an operator τ_Π that uses basic for producing heads that are logical consequences of a collection of heads and program clauses.

Definition 5.25. Let Π be a program and Σ a set of heads and program clauses, the operator τ_Π is defined as follows

$$\begin{aligned}\tau_\Pi(\Sigma) = \mathsf{basic}(\{\, H \vee H_1 \vee \ldots \vee H_n \mid\ & \square^b(H \leftarrow A_1 \wedge \ldots \wedge A_n) \in \Sigma,\\ & \{A_1 \vee H_1, \ldots, A_n \vee H_n\} \subseteq \Sigma \cap \mathrm{TDHB}_\Pi\})\end{aligned}$$

Besides the operator τ_Π, the following operator σ_Π is also used to define the operator T_Π. The operator σ_Π is applied to cyclic sequences of the form $\Sigma_0, \ldots, \Sigma_j, \ldots, \Sigma_k$ where each Σ_i is a set that contains heads and program clauses.

Definition 5.26. Let Π be a program. We define $\sigma_\Pi(\Sigma_0, \dots, \Sigma_j, \dots, \Sigma_k)$ to be the sequence $\Sigma'_0, \dots, \Sigma'_j, \dots, \Sigma'_k$ such that each Σ'_i is the least superset of Σ_i that satisfies the following conditions:

(1) If $p \in \Sigma'_i$ for every $i \in \{j, \dots, k\}$ then $\circ\Box p \in \Sigma'_j$
(2) If $\{H \vee \circ^g\Diamond p, H \vee \circ^g\Box q\} \subseteq \Sigma'_i$ then $H \vee \circ^g(q\,\mathcal{U}\,p) \in \Sigma'_i$
(3) If $\circ^g\psi \in \Sigma'_i$ and $i \geq 1$ then $\circ^{g+1}\psi \in \Sigma'_{i-1}$
(4) If $\circ^g\psi \in \Sigma'_j$ then $\circ^{g+1}\psi \in \Sigma'_k$
(5) If $\circ^g\psi \in \Sigma'_i$, $g \geq 1$ and $i \leq k-1$ then $(\circ^{g-1}\psi)^* \subseteq \Sigma'_{i+1}$ [8]
(6) If $\circ^g\psi \in \Sigma'_k$, $g \geq 1$ then $(\circ^{g-1}\psi)^* \subseteq \Sigma'_j$
(7) If $\circ^g\psi \in \Sigma'_i$, $g \geq 1$ and $i \in \{j, \dots, k\}$ then $\circ^{g+(k-j)+1}\psi \in \Sigma'_i$.

Definition 5.27. Let Σ be a set of heads and program clauses, $\mathsf{fold}(\Sigma)$ denotes the smallest superset of Σ that satisfies the following conditions:

(1) If $\{H \vee \circ^g q \vee \circ^g p, H \vee \circ^g q \vee \circ^{g+1}(p\,\mathcal{U}\,q)\} \subseteq \Sigma$ then $H \vee \circ^g(p\,\mathcal{U}\,q) \in \mathsf{fold}(\Sigma)$
(2) If $\{H \leftarrow B \wedge \circ^g q, H \leftarrow B \wedge \circ^g p \wedge \circ^{g+1}(p\,\mathcal{U}\,q)\} \subseteq \Sigma$ then $H \leftarrow B \wedge \circ^g(p\,\mathcal{U}\,q) \in \mathsf{fold}(\Sigma)$
(3) If $\{H \vee \circ^g q, H \vee \circ^g p \vee \circ^{g+1}(p\,\mathcal{R}\,q)\} \subseteq \Sigma$ then $H \vee \circ^g(p\,\mathcal{R}\,q) \in \mathsf{fold}(\Sigma)$
(4) If $\{H \leftarrow B \wedge \circ^g q \wedge \circ^g p, H \leftarrow B \wedge \circ^g q \wedge \circ^{g+1}(p\,\mathcal{R}\,q)\} \subseteq \Sigma$ then $H \leftarrow B \wedge \circ^g(p\,\mathcal{R}\,q) \in \mathsf{fold}(\Sigma)$
(5) If $H \vee \circ^g p \vee \circ^{g+1}\Diamond p \in \Sigma$ then $H \vee \circ^g\Diamond p \in \mathsf{fold}(\Sigma)$
(6) If $\{H \leftarrow B \wedge \circ^g p, H \leftarrow B \wedge \circ^{g+1}\Diamond p\} \subseteq \Sigma$ then $H \leftarrow B \wedge \circ^g\Diamond p \in \mathsf{fold}(\Sigma)$
(7) If $H \leftarrow B \wedge \circ^g p \wedge \circ^{g+1}\Box p \in \Sigma$ then $H \leftarrow B \wedge \circ^g\Box p \in \mathsf{fold}(\Sigma)$
(8) If $\{H \vee \circ^g p, H \vee \circ^{g+1}\Box p\} \subseteq \Sigma$ then $H \vee \circ^g\Box p \in \mathsf{fold}(\Sigma)$

It is worth noting that fold is a generalization of the reverse process of the temporal decomposition carried out by the context-free temporal rules. As a consequence, $\mathsf{fold}(\Sigma)$ is logically equivalent to Σ.

Definition 5.28. Let Π be a program, the operator

$$T_\Pi(\Sigma_0, \dots, \Sigma_j, \dots, \Sigma_k) = \sigma_\Pi(\tau_\Pi(\mathsf{fold}(\Sigma_0)), \dots, \tau_\Pi(\mathsf{fold}(\Sigma_j)), \dots, \tau_\Pi(\mathsf{fold}(\Sigma_k)))$$

The first iteration $T_\Pi \uparrow 0$ gives any reverse IFT-derivation for Π $\Pi^*_0, \dots, \Pi^*_j, \dots, \Pi^*_k$. Then, $T_\Pi \uparrow n = T_\Pi(T_\Pi \uparrow (n-1))$ for $n \geq 1$.

Example 5.29. In the case of the reverse IFT-derivation in Figure 18, $T_\Pi \uparrow 0$ would be the sequence Π^*_0, Π^*_1. Then $T_\Pi \uparrow 1$ would extend both Π^*_0 and Π^*_1 by means of τ_Π and the head p and its logical consequences produced by $\mathsf{basic}(p)$ would be introduced. Then, the operator σ_Π would extend Π^*_1 with $\circ\Box p$ and an infinite number of heads and clauses. Also Π^*_0 would be extended with an infinite number of heads and clauses. $T_\Pi \uparrow 2$ would contain the head $z \vee \circ(r\,\mathcal{U}\,q)$ and its logical consequence $z \vee \circ\circ\Diamond q$. Since also the clause $(\neg s)\,\mathcal{R}\,v \leftarrow \circ\circ\Diamond q$ is available because of the operator fold, $T_\Pi \uparrow 3$ would include the head $z \vee (\neg s)\,\mathcal{R}\,v$. $T_\Pi \uparrow 3$ would be the least fixpoint. ■

Next we show that the operator T_Π is monotonic and continuous and consequently it has the least fixpoint.

LEMMA 5.30. *The operator T_Π is monotonic and continuous for any program Π.*

PROOF. The monotonicity of T_Π is based on the monotonicity of the operators τ_Π, σ_Π and fold. Regarding the monotonicity of τ_Π, the only non-trivial cases are the ones that introduce heads

[8] In (5) and (6), when $g > 1$, $(\circ^{g-1}\psi)^* = \{\circ^{g-1}\psi\}$. However, for $g = 1$, $(\circ^{g-1}\psi)^*$ is the local closure of ψ (see Definition 5.6) which in general is a set of clauses that contain clauses with non-empty body even when ψ is a head or a clause with empty body.

of the forms $\circ^g(\neg p \mathcal{U}\, q) \vee H$ and $\circ^g(\neg p\, \mathcal{R}\, q) \vee H$ because they contain occurrences of negative literals. However these kinds of heads are not introduced by using negative or absent information (see Definitions 5.24 and 5.25). Hence, if $\Sigma \subseteq \Sigma'$ then $\tau_\Pi(\Sigma) \subseteq \tau_\Pi(\Sigma')$ for any pair of sets Σ and Σ'. The operators σ_Π and fold are trivially monotone. Therefore T_Π is monotone. Now, in order to show that T_Π is continuous, it suffices to show that T_Π is compact. The operator τ_Π is compact because –by using Definition 5.25– we can ensure that for every set Σ and every $\psi \in \tau_\Pi(\Sigma)$, there exists a finite $\Sigma' \subseteq \Sigma$ such that $\psi \in \tau_\Pi(\Sigma')$. On the other hand, regarding the compactness of the operator σ_Π, given a sequence $\Sigma_0, \ldots, \Sigma_j, \ldots, \Sigma_k$, the crucial point is the introduction of an atom of the form $\circ\Box\, p$ in the set Σ'_j when $p \in \Sigma_i$ for every $i \in \{j, \ldots, k\}$, that is, when $\circ^{i+\ell} p$ is a logical consequence of the program for every $\ell \geq j$. Let us consider the finite sequence made up of finite sets $S = \Sigma'_0, \ldots, \Sigma'_j, \ldots, \Sigma'_k$ where $\Sigma'_i = \emptyset$ for every $i \in \{0, \ldots, j-1\}$ and $\Sigma'_i = \{p\}$ for every $i \in \{j, \ldots, k\}$. If $\sigma_\Pi(S) = \Sigma''_0, \ldots, \Sigma''_j, \ldots, \Sigma''_k$, then, by Definition 5.26(1), $\circ\Box\, p \in \Sigma''_j$. Consequently, we can ensure that σ_Π is compact. The operator fold is trivially compact. Hence the operator T_Π is compact and also continuous. □

As a consequence of the well-known fixpoint theorem by Knaster and Tarski ([Knaster 1928; Tarski 1955]), we can provide the next result.

COROLLARY 5.31. *The least fixpoint* $T_\Pi \uparrow \omega = \bigcup_{n=0}^{\omega} T_\Pi \uparrow n$ *for any program* Π.

The least fixpoint $T_\Pi \uparrow \omega = \Sigma_0, \ldots, \Sigma_j, \ldots, \Sigma_k$ represents an infinite ultimately periodic structure of states $\Sigma_0, \ldots, \Sigma_{j-1}, \langle \Sigma_j, \ldots, \Sigma_k \rangle^\omega$ that asserts which heads and clauses are true at each state. By construction of $T_\Pi \uparrow \omega$, in particular by items (2), (3) and (6) in Definition 5.26, all such information is actually in Σ_0. Consequently, instead of dealing with $T_\Pi \uparrow \omega$ as a cyclic sequence, from now on we consider that $T_\Pi \uparrow \omega$ is just the set Σ_0. Next we show that for every program Π, there exists a model for this set $T_\Pi \uparrow \omega$.

LEMMA 5.32. *For any program* Π*, there exists a model of* $T_\Pi \uparrow \omega$.

PROOF. Since every atom in $T_\Pi \uparrow \omega$ can be satisfied by using only positive propositional literals, we consider the PLTL-structure $\mathcal{M} = (S_\mathcal{M}, V_\mathcal{M})$ such that $s_i = \{\psi \in \mathsf{TDHB}_\Pi \mid \circ^i\psi \in T_\Pi \uparrow \omega\}$ and $V(s_i) = \{\psi \in \mathsf{Prop} \mid \circ^i\psi \in \mathsf{THB}_\Pi\}$ for every $i \geq 0$. $\mathcal{M}$ is trivially a model of $T_\Pi \uparrow \omega$. □

COROLLARY 5.33. *Any program* Π *has a model.*

Finally, we are going to prove that $T_\Pi \uparrow \omega$ is formed by all the heads that are needed to obtain (by a kind of closure, below called expansion) all the formulas in the temporal disjunctive Herbrand base that are logical consequences of Π. Next we define the expansion of $T_\Pi \uparrow \omega$ which generalizes the usual disjunctive expansion in DLP (see e.g [Lobo et al. 1992; Gergatsoulis et al. 2000]).

Definition 5.34. Let Π be a program and $T_\Pi \uparrow \omega$ its least fixpoint. The *expansion* of $T_\Pi \uparrow \omega$, denoted by $\exp(T_\Pi \uparrow \omega)$, is the smallest superset of $T_\Pi \uparrow \omega$ that satisfies the following conditions:

(1) If $\psi \in \mathsf{TDHB}_\Pi \cap \exp(T_\Pi \uparrow \omega)$ and H is a head in TDHB_Π, then $\psi \vee H \in \exp(T_\Pi \uparrow \omega)$
(2) If H is a head in $\exp(T_\Pi \uparrow \omega)$ and $\psi \in \mathsf{TDHB}_\Pi \cap T_\Pi \uparrow \omega$, then $\psi \vee H \in \exp(T_\Pi \uparrow \omega)$
(3) If $\circ^i H_1, \ldots, \circ^i H_n$, with $n \geq 1$ and $i \geq 0$, are heads that belong to $\exp(T_\Pi \uparrow \omega)$, then $\Diamond(H_1 \vee \ldots \vee H_n) \in \exp(T_\Pi \uparrow \omega)$.

We now proof that $\exp(T_\Pi \uparrow \omega)$ exactly contains the formulas in TDHB_Π that are logical consequences of Π.

THEOREM 5.35. *Let* Π *be a program and* $\psi \in$ *TDHB*$_\Pi$. *If* $\psi \in \mathsf{exp}(T_\Pi \uparrow \omega)$ *then* ψ *is a logical consequence of* Π.

PROOF. On one hand, if $\psi \in T_\Pi \uparrow \omega$, then $\psi \in T_\Pi \uparrow n$ for some $n \geq 0$. Next we show, by induction on n, that ψ is a logical consequence of Π. Since $T_\Pi \uparrow 0 \cap \mathsf{TDHB}_\Pi$ is empty, the case $\psi \in T_\Pi \uparrow 0$ is trivial. Now let us consider the case $\psi \in T_\Pi \uparrow n$ for $n \geq 1$. By Definitions

5.25, 5.26, 5.27 and 5.28, ψ is obtained by using a finite set of heads in $T_\Pi \uparrow (n-1)$ –that, by induction hypothesis, are logical consequences of Π– and the applications of the operators fold, τ_Π and σ_Π. By the semantics of the temporal connectives, fold introduces formulas that are logically equivalent to some formula in the input set. The operator τ_Π and σ_Π produce formulas that are logical consequences of some set of formulas that is a subset of the input set. Therefore, by transitivity of the logical consequence relation, ψ is a logical consequence of Π. On the other hand, if $\psi \in \exp(T_\Pi \uparrow \omega)$ but $\psi \notin T_\Pi \uparrow \omega$, then, by the semantics of the temporal connectives and the classical disjunction, ψ is a logical consequence of some finite subset of $T_\Pi \uparrow \omega$. □

THEOREM 5.36. *Let* Π *be a program and* $\psi \in TDHB_\Pi$. *If* ψ *is a logical consequence of* Π *then* $\psi \in \mathsf{exp}(T_\Pi \uparrow \omega)$.

PROOF. If $\psi \notin \mathsf{exp}(T_\Pi \uparrow \omega)$, then, by Definitions 5.28 and 5.34 and by considering that the atoms can only have a finite number of syntactical forms (see Definition 5.24), the generation of ψ requires an infinite set, but this contradicts the compactness result obtained in Lemma 5.30. □

6. RELATED WORK

In Section 1, we have already surveyed the main features of the works that are more close to our proposal. In this section we add more details.

6.1. Templog [Abadi and Manna 1987; 1989; Baudinet 1989b]

The only temporal connectives allowed in the TLP language Templog introduced in [Abadi and Manna 1987; 1989; Baudinet 1989b] are $\Box$, $\Diamond$ and $\circ$. An atom is of the form $\circ^i A$ where A is a classical atom. A body B is recursively defined as a conjunction $B_1 \wedge \ldots \wedge B_n$ with $n \geq 0$ and where each B_i is a classical atom A, a formula of the form $\circ B'$, i.e., a body preceded by the connective $\circ$, or a formula of the form $\Diamond B'$, i.e., a body preceded bay the connective $\Diamond$. Program clauses are of the form $\Box^b((\Box^{b'} \circ^i A) \leftarrow B)$, with $b, b' \in \{0,1\}$, and goal clauses are of the form $\bot \leftarrow B$. Templog does not deal with eventualities because the connective $\Diamond$ appears only in clause bodies. As can be appreciated in the recursive definition of bodies, the nesting of connectives in Templog clauses is not as restricted as in TeDiLog. Therefore, the structure of clauses is considerably more complex in Templog than in TeDiLog. For example, we do not allow the connective $\Diamond$ to prefix a conjunction of atoms. Since this normal form of Templog clauses is not well suited for resolution, the notion of canonical body is additionally considered in Templog. A canonical body is a body in which occurrences of the connectives $\wedge$ and $\Diamond$ cannot appear in the scope of the connective $\circ$ and every atom of the form $\circ^i A$ is in the scope of the least possible numbers of $\Diamond$. The equivalences $\circ(\varphi \wedge \psi) \equiv \circ\varphi \wedge \circ\psi$, $\circ\Diamond\psi \equiv \Diamond\circ\psi$ and $\Diamond(\Diamond\Diamond\varphi \wedge \Diamond\psi) \equiv \Diamond(\Diamond\varphi \wedge \psi)$ are used to obtain the canonical form of bodies. However, although the bodies of the premises are in canonical form, the resolvent obtained by a resolution application may yield a clause whose body is not in canonical form, hence a transformation to obtain the canonical form may be required after each resolution application. In TeDiLog, translation into clausal normal form is required only when the context-dependent rules are applied. Such translation obtains a set of clauses that defines the meaning of the fresh variable relating it to the negation of the context. The resolution procedure TSLD ([Baudinet 1989b]) consists of eight rules obtained by considering all the possible cases in which temporal atoms of a program clause and a goal clause can be resolved. For instance, we depict here one of the rules

$$\frac{\Box(\circ^j A \leftarrow B_0) \qquad\qquad \bot \leftarrow B_1 \wedge \Diamond(B_2 \wedge \circ^i A \wedge B_3) \wedge B_4}{\bot \leftarrow B_1 \wedge \Diamond(\circ^{j-i} B_2 \wedge B_0 \wedge \circ^{j-i} B_3) \wedge B_4} \text{ where } j \geq i$$

This resolution rule states that a program clause of the form $\Box(\circ^j A \leftarrow B_0)$ is resolved with a goal clause of the form $\bot \leftarrow B_1 \wedge \Diamond(B_2 \wedge \circ^i A \wedge B_3) \wedge B_4$ and the resolvent $\bot \leftarrow B_1 \wedge \Diamond(\circ^{j-i} B_2 \wedge B_0 \wedge \circ^{j-i} B_3) \wedge B_4$ is obtained, whenever $j \geq i$. Note that A is a classical atom.

The Templog resolution procedure does not follow the state by state forward reasoning approach and, consequently, it does not use any rule similar to our rule (Unx). As already mentioned in Section 1, the satisfiability of a Templog program can be reduced to the satisfiability of a (possibly infinite) classical logic program. This is easily made by considering, for instance, that a clause of the form $\circ^i A \leftarrow \diamond B$ can be expressed by means of the infinite set of clauses $\{\circ^i A \leftarrow \circ^j B \mid j \geq 0\}$ and, in the same way, a clause of the form $(\square \circ^i A) \leftarrow B$ can be expressed by means of the infinite set of clauses $\{(\circ^{j+i} A) \leftarrow B \mid j \geq 0\}$. This approach is possible neither when the connectives $\diamond$ and $\mathcal{U}$ appear in the head of a clause nor when the connectives $\square$ and $\mathcal{R}$ appear in the body. For instance, note that a clause of the form $\diamond A \leftarrow B$ should be replaced with a unique clause $\circ^k A \leftarrow B$ but the value of such k is unknown. As a consequence, the minimal model characterization of Templog (see [Baudinet 1989b]) is a straightforward adaptation of the classical case. Unlike Templog, TeDiLog does not have the classical Minimal Model Property (MMP in short). The presence of the connectives $\diamond$ and $\mathcal{U}$ in clause heads and $\square$ and $\mathcal{R}$ in clause bodies (see [Orgun and Wadge 1992]) as well as the use of disjunction in clause heads (see e.g. [Lobo et al. 1992]) prevent from having such property. The compensation for the loss of the MMP is that TeDiLog is much more expressive than the propositional fragment of Templog.

In order to study Templog's expressiveness, Baudinet considers in [Baudinet 1989b; 1995] the propositional fragment TL1 where the connective $\diamond$ is not allowed at all and $\square$ is not allowed in clause heads. Consequently, TL1 program clauses are of the form $\square^b(\circ^i A_0 \leftarrow \circ^{j_1} A_1 \wedge \ldots \wedge \circ^{j_n} A_n)$ where $b \in \{0, 1\}$ and $n \geq 0$ and goal clauses are of the form $\bot \leftarrow \circ^{j_1} A_1 \wedge \ldots \wedge \circ^{j_n} A_n$ where $n \geq 0$. Baudinet shows that the expressiveness of TL1 and propositional Templog is the same. On one hand, Templog clauses of the form $\square \circ^i p \leftarrow B$ can be expressed without using the connective $\square$ by introducing a fresh propositional variable. So that, the above program clause can be expressed by means of the program clauses $\{q \leftarrow B, \square(\circ^i p \leftarrow q), \square(\circ q \leftarrow q)\}$ where q is fresh. On the other hand, each element of the form $\diamond \circ^i p$ in a body of a clause, can be substituted by a fresh propositional symbol q and then the clauses that define the meaning of q would be added: $\{\square(q \leftarrow \circ^i p), \square(q \leftarrow \circ q)\}$. Moreover, Baudinet shows that, for instance, it is possible to define, in TL1, a predicate that holds exactly when $p \mathcal{U} q$ holds, whereas the connective $\mathcal{U}$ is not expressible in temporal logic with only $\circ$, $\square$ and $\diamond$ (see [Kamp 1968]). So that, there are predicates that can be defined by using TL1 but are inexpressible in temporal logic. Baudinet also shows that, for instance, the connective $\square$ is not expressible in Templog, in the sense that it is not possible to prove $\square q$ or to write a Templog program defining a predicate that would hold exactly when $\square q$ holds. This last result proves that TeDiLog is more expressive than (propositional) Templog, because in TeDiLog $\square q$ can be proved, as has been shown in Example 5.10 (Figure 14).

6.2. Chronolog [Wadge 1988; Orgun 1991; 1995; Gergatsoulis et al. 2000]

In Chronolog ([Wadge 1988; Orgun 1991; 1995]) the only temporal operators are the unary connectives first and next. The connective first serves to refer to the initial state s_0. Therefore the connective $\square$ is not needed to differentiate between always- and now-clauses. The TeDiLog now-clauses $p \leftarrow \circ q$, $\square p \leftarrow \circ q$ and $p \leftarrow \circ\diamond q \wedge r$ can be expressed in Chronolog as $\mathsf{first}\, p \leftarrow \mathsf{first\ next}\, q$, $p \leftarrow \mathsf{first\ next}\, q$ and $\mathsf{first}\, p \leftarrow \mathsf{next}\, q \wedge \mathsf{first}\, r$, respectively. The TeDiLog always-clause $\square(p \leftarrow \circ q)$ can be expressed in Chronolog as $p \leftarrow \mathsf{next}\, q$. Note that in the Chronolog clauses above, there is a hidden temporal information not made explicit by means of temporal connectives. Regarding always-clauses of the form $\square(\square p \leftarrow \circ q)$ and $\square(s \leftarrow \diamond r)$, the translations pointed out to obtain TL1 clauses in the previous subsection must be considered for $\square p$ and $\diamond r$. Consequently, intricate sets of Chronolog clauses are needed for expressing interesting properties. In TeDiLog, the explicit use of temporal connectives, together with the fact that such connectives are more expressive, facilitates readability and understanding of program and goal clauses. In [Baudinet 1995], Baudinet shows –by means of TL1– that Templog and Chronolog have the same expressive power. Hence Chronolog can be considered as a syntactical variant of Templog. In fact, Templog and Chronolog also coincide in the metalogical properties of minimal model existence and fixpoint characterization. The resolution procedure TiSLD that defines the operational semantics of Chronolog, applies the resolution rule to *rigid instances* of program clauses and goal clauses, which are formed by atoms of the form $\mathsf{first}\ \mathsf{next}^n\ p$ with $n \geq 0$. In [Orgun 1995], the inclusion of the temporal connectives $\diamond$ and $\square$ is discussed. However, by taking into account the results presented in [Orgun and Wadge 1992], and in order to keep the metalogical properties of Chronolog, only the use of $\diamond$ in clause bodies and $\square$ in clause heads is proposed. This extension would yield a language that would be (syntactically) very similar to Templog. However, the expressive power would remain unchanged. The disjunctive extension presented in [Gergatsoulis et al. 2000] combines Chronolog with the Disjunctive LP paradigm. Therefore, only the temporal connectives first and next are used and the results obtained in the Disjunctive Logic Programming paradigm are extended to Disjunctive Chronolog in the same way that the results obtained in classical Logic Programming are extended to Chronolog.

6.3. Temporal Prolog [Gabbay 1987b]

Gabbay's Temporal Prolog allows eventuality literals in clause heads but not in clause bodies. In particular, $\diamond$ is allowed in clause heads but $\square$ is not allowed in clause bodies. A program clause is either a now-clause $H \leftarrow B$ or an always clause $\square\,(H \leftarrow B)$. The head H is either a classical atom A or a formula of the form $\circ\diamond\, C$ where C is a conjunction of now-clauses. The body B is a classical atom A, a conjunction of bodies or a formula of the form $\circ\diamond\, B'$ where B' is a body. A goal clause is of the form $\bot \leftarrow B$ where B is a body. Additionally, a connective to express "sometime in the past" is also used. So that, the clausal form of Gabbay's Temporal Prolog is more complex than ours. In particular, the nesting of connectives is not so restricted as in TeDiLog. Although eventuality literals are allowed in clause heads, the way of dealing with them is very different from our method. For instance, given a goal of the form $\bot \leftarrow \circ\diamond\, p$ the resolution procedure tries to find a program clause whose head is either p or $\circ\diamond\, p$. If such clause is found, a forward jump is produced. The resolution procedure of TeDiLog is based on a state by state forward reasoning and eventualities are dealt with by means of the context-dependent rules which do not allow to indefinitely postpone the fulfillment of such eventualities. As mentioned above, unlike in TeDiLog, the connective $\square$ is not allowed in clause bodies, hence TeDiLog is more expressive. For Gabbay's Temporal Prolog the MMP does not hold because of the use of eventualities in clauses heads. Additionally, the completeness proof of the resolution procedure is not provided. The IFT-resolution procedure for TeDiLog is complete.

6.4. MetateM [Barringer et al. 1989]

MetateM programs are sets of clauses in the Separated Normal Form (SNF), where clauses are of the form $\varphi \rightarrow \psi$ such that φ is a conjunction of propositional literals and ψ is either of the form $\diamond\,\chi$ –where χ is a propositional literal– or a disjunction of propositional literals prefixed by the connective $\circ$. Every PLTL formula can be translated into SNF by introducing, in general, fresh variables. Although the connective $\mathcal{U}$ is not expressible by using $\diamond$ and $\circ$ in PLTL ([Kamp 1968]), SNF eliminates the occurrences of $\mathcal{U}$ by taking into account that the formulas $p \leftrightarrow \psi_1\,\mathcal{U}\,\psi_2$ and $p \leftrightarrow (\psi_2 \vee (\psi_1 \wedge \circ p)) \wedge \diamond\,\psi_2$ are equivalent.[9] Consequently, a formula and the corresponding SNF formula are equisatisfiable but, in general, they are not logically equivalent. MetateM is as expressive as TeDiLog and complete for full PLTL. However, MetateM is based on the imperative future approach and is not based on resolution. Regarding execution, at each step the MetateM execution procedure must build the next state by choosing to make true one proposition from the ψ part of each clause for which the φ part is true in the current state. In this way, a sequence of states is produced with the aim of building a model for the program. Choices that lead to inconsistency must be repaired by means of backtracking, which serves to choose another disjunct from the corresponding ψ part. Additionally, the finite-model property is used to calculate an upper bound of forward chaining steps and, in this way, to detect model construction processes where the fulfillment of an eventuality is being indefinitely delayed. Such upper bound, in the worst case, is $2^{5|\Pi|}$ where $|\Pi|$ is the size

[9] $\leftrightarrow$ is the classical connective for logical equivalence, i.e., $\chi_1 \leftrightarrow \chi_2 \equiv (\chi_1 \rightarrow \chi_2) \wedge (\chi_1 \leftarrow \chi_2)$.

of the initial program Π (see [Barringer et al. 1989] and Subsection 6.2.4 in [Fisher 2011]). The IFT-resolution procedure underlying TeDiLog does need neither backtracking nor the calculation of upper bounds to stop derivations. As in TeDiLog, the execution mechanism of MetateM must make sure that the satisfaction of an eventuality is not continually postponed. For a clause $\varphi \rightarrow \circ\diamond p$, it is possible to make true p or to make true $\diamond p$ in the next state. If there are two clauses of the form $\varphi \rightarrow \circ\diamond p$ and $\varphi' \rightarrow \circ\diamond\neg p$ such that φ and φ' are satisfied in every state, it is necessary to satisfy p and $\neg p$ in an interleaved way. Therefore, fairness is required when deciding which eventuality to satisfy. This is handled by keeping an ordered list of eventualities (see Subsection 6.2.7 in [Fisher 2011]).

6.5. Clausal Temporal Resolution for PLTL [Fisher 1991]

The clausal temporal resolution method introduced in [Fisher 1991] (see also [Fisher et al. 2001]) is complete for full PLTL. Our clausal normal form is different from the Separated Normal Form used in that method but the crucial difference of our approach with respect to that method is that TeDiLog's resolution mechanism is powerful enough to deal with eventualities without requiring invariant generation.

7. CONCLUDING REMARKS

TeDiLog is a very expressive resolution-based propositional TLP language with a purely declarative nature and mathematically defined semantics. In particular, the language TeDiLog is strictly more expressive than the propositional fragments of the main declarative TLP languages in the literature ([Abadi and Manna 1987; Wadge 1988; Gabbay 1987b; Gergatsoulis et al. 2000]). The most significant imperative TLP language MetateM ([Barringer et al. 1989]) is as expressive as TeDiLog. However, MetateM is a very different approach that is not based on resolution and uses backtracking.

TeDiLog's resolution mechanism 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 [Fisher 1991] (see also [Fisher et al. 2001]) which needs to generate invariant formulas for solving eventualities.

We see TeDiLog as the propositional kernel of a new generation of TLP languages based on the so-called invariant-free temporal resolution. In this sense we hope that TeDiLog could influence the design of future TLP languages to incorporate more expressive temporal features and new resolution procedures for temporal reasoning.

The implementation of TeDiLog remains as future work. At the moment we are adapting a previous prototype that implements the TRS-resolution method for general clauses (cf. [Gaintzarain et al. 2013]) to the more restricted IFT-resolution for TeDiLog programs and goals. However, much experimentation is needed for optimization and improvement.

The worst case complexity for TeDiLog (regarding the generation of a refutation proof) is doubly exponential (see [Gaintzarain et al. 2013]) but it has been shown (see e.g. [Goranko et al. 2010; Hustadt and Schmidt 2002]) that in some cases the average performance of a doubly exponential algorithm can be better than the average performance of an exponential one. The reason is that, in the former the cases with high complexity rarely occur, while in the latter the cases with exponential complexity occur very often. The results obtained in the analysis carried out in [Hustadt and Schmidt 1999] give hints about improvements to be considered for a practical implementation. On the other hand, the possibility of searching for tractable fragments must be considered, as it is done in [Dixon et al. 2006; Dixon et al. 2000].

It is well known (see [Baudinet 1989a; 1989b; 1992; 1995]) that, although logic programs are formulas of a given logic, a logic programming language may be in some respects more expressive than its underlying logic. Intuitively, a logic formula characterizes just the collection of its models whereas a logic program characterizes the collection of facts that can be inferred from it. The notion of deduction intervenes and adds the ability to express properties that are not expressible in the underlying logic. In this sense it would be interesting to compare the expressiveness of TeDiLog to other formalisms such as PLTL, automata-theoretic formalisms, quantified PLTL (i.e. QPTL), μTL, etc.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their comments that were very helpful for shaping the manuscript.

REFERENCES

Martín Abadi and Zohar Manna. 1987. Temporal Logic Programming. In *Proceedings of the International Symposium on Logic Programming*. IEEE Computer Society Press, 4–16.

Martín Abadi and Zohar Manna. 1989. Temporal logic programming. *Journal of Symbolic Computation* 8, 3 (September 1989), 277–295.

Felicidad Aguado, Pedro Cabalar, Gilberto Pérez, and Concepción Vidal. 2008. Strongly Equivalent Temporal Logic Programs. In *Proceedings of the 11th European Conference on Logics in Artificial Intelligence (JELIA)*. LNCS, Vol. 5293. Springer, 8–20.

Felicidad Aguado, Pedro Cabalar, Gilberto Pérez, and Concepción Vidal. 2011. Loop formulas for splitable temporal logic programs. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. LNCS, Vol. 6645. Springer, 80–92.

Philippe Balbiani, Luis Fariñas del Cerro, and Andreas Herzig. 1988. Declarative Semantics for Modal Logic Programs. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988 (FGCS)*. OHMSHA Ltd. Tokyo and Springer-Verlag, 507–514.

Howard Barringer, Michael Fisher, Dov M. Gabbay, Graham Gough, and Richard Owens. 1989. METATEM: A Framework for Programming in Temporal Logic. In *Proceedings of the REX (Research and Education in Concurrent Systems) Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*. LNCS, Vol. 430. Springer, 94–129.

Marianne Baudinet. 1988. *On the Semantics of Temporal Logic Programming*. Technical Report CS-TR-88-1203. Department of Computer Science, Stanford University, California, USA. ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/88/1203/CS-TR-88-1203.pdf.

Marianne Baudinet. 1989a. *Logic Programming Semantics: Techniques and Applications*. Ph.D. Dissertation. Department of Computer Science, Stanford University, California, USA.

Marianne Baudinet. 1989b. Temporal Logic Programming is Complete and Expressive. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL)*. ACM Press, 267–280.

Marianne Baudinet. 1992. A Simple Proof of the Completeness of Temporal Logic Programming. In *Intensional Logics for Programming*. Oxford University Press, 51–83.

Marianne Baudinet. 1995. On the Expressiveness of Temporal Logic Programming. *Information and Computation* 117, 2 (1995), 157–180.

Kai Brünnler and Martin Lange. 2008. Cut-Free Sequent Systems for Temporal Logic. *Journal of Logic and Algebraic Programming* 76, 2 (July-August 2008), 216–225.

Christoph Brzoska. 1991. Temporal Logic Programming and its Relation to Constraint Logic Programming. In *Proceedings of the International Symposium on Logic Programming (ISLP)*. MIT Press, 661–677.

Christoph Brzoska. 1993. Temporal Logic Programming with Bounded Universal Modality Goals. In *Proceedings of the 10th International Conference on Logic Programming (ICLP)*. MIT Press, 239–256.

Christoph Brzoska. 1995a. Temporal Logic Programming in Dense Time. In *Proceedings of the International Logic Programming Symposium (ILPS)*. MIT Press, 303–317.

Christoph Brzoska. 1995b. Temporal Logic Programming with Metric and Past Operators. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*. LNCS, Vol. 897. Springer, 21–39.

Christoph Brzoska. 1998. Programming in Metric Temporal Logic. *Theoretical Computer Science* 202, 1-2 (July 1998), 55–125.

Christoph Brzoska and Karl Schäfer. 1995. Temporal Logic Programming Applied to Image Sequence Evaluation. In *Logic Programming: Formal Methods and Practical Applications*. Elsevier Science B.V./North-Holland, 381–395.

Antonio Cau, Hussein Zedan, Nick Coleman, and Ben C. Moszkowski. 1996. Using ITL and Tempura for Large-Scale Specification and Simulation. In *Proceedings of the 4th Euromicro Workshop on Parallel and Distributed Processing (PDP)*. IEEE Computer Society Press, 493–500.

Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. 1986. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems* 8, 2 (April 1986), 244–263.

Edmund M. Clarke, Orna Grumberg, and Doron Peled. 2001. *Model checking*. MIT Press.

Clare Dixon and Michael Fisher. 1998. The Set of Support Strategy in Temporal Resolution. In *Proceedings of the 5th International Workshop on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society Press, 113–120.

Clare Dixon, Michael Fisher, and Boris Konev. 2006. Is There a Future for Deductive Temporal Verification?. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE Computer Society Press, 11–18.

Clare Dixon, Michael Fisher, and Mark Reynolds. 2000. Execution and proof in a Horn-clause temporal logic. In *Advances in Temporal Logic*. Kluwer Academic Publishers, 413–433.

Zhenhua Duan, Xiaoxiao Yang, and Maciej Koutny. 2005. Semantics of Framed Temporal Logic Programs. In *Proceedings of the 21st International Conference on Logic Programming (ICLP)*. LNCS, Vol. 3668. Springer, 356–370.

Zhenhua Duan, Xiaoxiao Yang, and Maciej Koutny. 2008. Framed temporal logic programming. *Science of Computer Programming* 70, 1 (January 2008), 31–61.

Luis Fariñas del Cerro. 1986. MOLOG: A System that Extends PROLOG with modal logic. *New Generation Computing* 4 (1986), 35–50.

Michael Fisher. 1991. A Resolution Method for Temporal Logic. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 99–104.

Michael Fisher. 1992. A Normal Form for First-Order Temporal Formulae. In *Proceedings of the 11th International Conference on Automated Deduction (CADE)*. LNCS, Vol. 607. Springer, 370–384.

Michael Fisher. 1993. Concurrent MetateM – A Language for Modeling Reactive Systems. In *Proceedings of the Conference on Parallel Architectures and Languages, Europe (PARLE)*. LNCS, Vol. 694. Springer, 185–196.

Michael Fisher. 1997. Implementing BDI-like Systems by Direct Execution. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 1. Morgan Kaufmann, 316–321.

Michael Fisher. 2011. *An Introduction to Practical Formal Methods Using Temporal Logic*. John Wiley & Sons, Ltd.

Michael Fisher, Clare Dixon, and Martin Peim. 2001. Clausal temporal resolution. *ACM Transactions on Computational Logic* 2, 1 (January 2001), 12–56.

Michael Fisher and Chiara Ghidini. 2010. Executable specifications of resource-bounded agents. *Autonomous Agents and Multi-Agent Systems* 21, 3 (November 2010), 368–396.

Thom W. Frühwirth. 1994. Annotated Constraint Logic Programming Applied to Temporal Reasoning. In *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming (PLILP)*. LNCS, Vol. 844. Springer, 230–243.

Thom W. Frühwirth. 1995. Temporal Logic and Annotated Constraint Logic Programming. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*. LNCS, Vol. 897. Springer, 58–68.

Thom W. Frühwirth. 1996. Temporal Annotated Constraint Logic Programming. *Journal of Symbolic Computation* 22, 5/6 (November/December 1996), 555–583.

Masahiro Fujita, Shinji Kono, Hidehiko Tanaka, and Tohru Moto-Oka. 1986. Tokio: Logic Programming Language Based on Temporal Logic and its Compilation to Prolog. In *Proceedings of the 3rd International Conference on Logic Programming (ICLP)*. LNCS, Vol. 225. Springer, 695–709.

Dov M. Gabbay. 1987a. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In *Proceedings of the Colloquium on Temporal Logic in Specification*. LNCS, Vol. 398. Springer, 409–448.

Dov M. Gabbay. 1987b. Modal And Temporal Logic Programming. In *Temporal Logics And Their Application*. Academic Press, 197–237.

Jose Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. 2009. Dual Systems of Tableaux and Sequents for PLTL. *The Journal of Logic and Algebraic Programming* 78, 8 (November 2009), 701–722.

Jose Gaintzarain, Montserrat Hermo, Paqui Lucio, Marisa Navarro, and Fernando Orejas. 2013. Invariant-Free Clausal Temporal Resolution. *Journal of Automated Reasoning* 50, 1 (January 2013), 1–49.

Jose Gaintzarain and Paqui Lucio. 2009. A New Approach to Temporal Logic Programming. In *Proceedings of the 9th Spanish Conference on Programming and Languages (PROLE)*. 341–350.
http://www.sistedes.es/ficheros/actas-conferencias/PROLE/2009.pdf.

Manolis Gergatsoulis. 2001. Temporal and Modal Logic Programming Languages. In *Encyclopedia of Microcomputers*, Vol. 27. CRC Press, 393–408.

Manolis Gergatsoulis, Panos Rondogiannis, and Themis Panayiotopoulos. 2000. Temporal Disjunctive Logic Programming. *New Generation Computing* 19, 1 (December 2000), 87–102.

Valentin Goranko, Angelo Kyrilov, and Dmitry Shkatov. 2010. Tableau Tool for Testing Satisfiability in LTL: Implementation and Experimental Analysis. In *Proceedings of the 6th Workshop on Methods for Modalities*. Electronic Notes in Theoretical Computer Science, Vol. 262. Elsevier, 113–125.

Tomas Hrycej. 1988. Temporal Prolog. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI)*. Pitmann Publishing, 296–301.

Tomas Hrycej. 1993. A Temporal Extension of Prolog. *Journal of Logic Programming* 15, 1 & 2 (January 1993), 113–145.

Ullrich Hustadt and Renate A. Schmidt. 1999. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics* 9, 4 (1999), 479–522.

Ullrich Hustadt and Renate A. Schmidt. 2002. Scientific Benchmarking with Temporal Logic Decision Procedures. In *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR)*. Morgan Kaufmann, 533–544.

Johan Anthony Willem Kamp. 1968. *Tense Logic and the Theory of Linear Order*. Ph.D. Dissertation. Department of Computer Science, University of California at Los Angeles, California, USA.

Bronislaw Knaster. 1928. Un théorèm sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathematique* 6 (1928), 133–134.

Shinji Kono. 1995. A Combination of Clausal and Non Clausal Temporal Logic Programs. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*. LNCS, Vol. 897. Springer, 40–57.

Shinji Kono, T. Aoyagi, Masahiro Fujita, and Hidehiko Tanaka. 1985. Implementation of Temporal Logic Programming Language Tokio. In *Proceedings of the 4th Conference on Logic Programming (LP)*. LNCS, Vol. 221. Springer, 138–147.

John W. Lloyd. 1984. *Foundations of Logic Programming, 1st Edition*. Springer.

Jorge Lobo, Jack Minker, and Arcot Rajasekar. 1992. *Foundations of disjunctive logic programming*. MIT Press.

Stephan Merz. 1992. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-Classical Logics* 2, 2 (1992), 139–156.

Stephan Merz. 1995. Efficiently Executable Temporal Logic Programs. In *Proceedings of the IJCAI'93 Workshop on Executable Modal and Temporal Logics*. LNCS, Vol. 897. Springer, 69–85.

Ben C. Moszkowski. 1986. *Executing temporal logic programs*. Cambridge University Press.

Ben C. Moszkowski. 1998. Compositional Reasoning Using Interval Temporal Logic and Tempura. In *Compositionality: The Significant Difference. International Symposium, COMPOS'97. Revised Lectures*. LNCS, Vol. 1536. Springer, 439–464.

Hiroshi Nakamura, Masaya Nakai, Shinji Kono, Masahiro Fujita, and Hidehiko Tanaka. 1989. Logic Design Assistance Using Temporal Logic Based Language Tokio. In *Proceedings of the 8th Conference on Logic Programming (LP)*. LNCS, Vol. 485. Springer, 174–183.

Linh Anh Nguyen. 2000. Constructing the least models for positive modal logic programs. *Fundamenta Informaticae* 42, 1 (March 2000), 29–60.

Linh Anh Nguyen. 2003. A fixpoint semantics and an SLD-resolution calculus for modal logic programs. *Fundamenta Informaticae* 55, 1 (2003), 63–100.

Linh Anh Nguyen. 2006. Multimodal logic programming. *Theoretical Computer Science* 360, 1-3 (August 2006), 247–288.

Linh Anh Nguyen. 2009. Modal logic programming revisited. *Journal of Applied Non-Classical Logics* 19, 2 (2009), 167–181.

Mehmet A. Orgun. 1991. *Intensional Logic Programming*. Ph.D. Dissertation. Department of Computer Science, University of Victoria, British Columbia, Canada.

Mehmet A. Orgun. 1994. Temporal and Modal Logic Programming: An Annotated Bibliography. *SIGART Bulletin* 5, 3 (July 1994), 52–59.

Mehmet A. Orgun. 1995. Foundations of linear-time logic programming. *International Journal of Computer Mathematics* 58, 3-4 (1995), 199–219.

Mehmet A. Orgun and Wanli Ma. 1994. An Overview of Temporal and Modal Logic Programming. In *Proceedings of the 1st International Conference on Temporal Logic (ICTL)*. LNCS, Vol. 827. Springer, 445–479.

Mehmet A. Orgun and William W. Wadge. 1992. Towards a Unified Theory of Intensional Logic Programming. *Journal of Logic Programming* 13, 4 (August 1992), 413–440.

Mehmet A. Orgun and William W. Wadge. 1994. Extending Temporal Logic Programming with Choice Predicates Non-Determinism. *Journal of Logic and Computation* 4, 6 (December 1994), 877–903.

Mehmet A. Orgun, William W. Wadge, and Weichang Du. 1993. Chronolog (Z): Linear-Time Logic Programming. In *Proceedings of the 5th International Conference on Computing and Information (ICCI)*. IEEE Computer Society Press, 545–549.

Barbara Paech. 1988. Gentzen-Systems for Propositional Temporal Logics. In *Proceedings of the 2nd Workshop on Computer Science Logic (CSL)*. LNCS, Vol. 385. Springer, 240–253.

Regimantas Pliuskevicius. 1991. Investigation of Finitary Calculus for a Discrete Linear Time Logic by means of Infinitary Calculus. In *Baltic Computer Science, selected papers*. LNCS, Vol. 502. Springer, 504–528.

Regimantas Pliuskevicius. 1992. Logical Foundation for Logic Programming Based on First Order Linear Temporal Logic. In *Proceedings of the First (1990) and Second (1991) Russian Conference on Logic Programming (RCLP)*. LNCS, Vol. 592. Springer, 391–406.

Alessandra Raffaetà and Thom W. Frühwirth. 1999. Two semantics for temporal annotated constraint logic programming. In *Proceedings of the 12th International Symposium on Languages for Intensional Programming (ISLIP)*. World Scientific Press, 126–140.

Han Reichgelt. 1987. Semantics for reified temporal logic. In *Advances in Artificial Intelligence*. John Wiley & Sons, Ltd., 49–61.

Panos Rondogiannis, Manolis Gergatsoulis, and Themis Panayiotopoulos. 1997. Cactus: A Branching-Time Logic Programming Language. In *Proceedings of the 1st International Joint Conference on Qualitative and Quantitative Practical Reasoning (ECSQARU-FAPR)*. LNCS, Vol. 1244. Springer, 511–524.

Panos Rondogiannis, Manolis Gergatsoulis, and Themis Panayiotopoulos. 1998. Branching-Time Logic Programming: The Language Cactus and its Applications. *Computer Languages* 24, 3 (1998), 155–178.

Takashi Sakuragawa. 1986. *Temporal Prolog*. Technical Report. Kyoto University. http://repository.kulib.kyoto-u.ac.jp/dspace/bitstream/2433/99379/1/0586-16.pdf.

Uwe Schöning. 1989. *Logic for Computer Scientists*. Birkhäuser.

Yoav Shoham. 1986. Reified Temporal Logics: Semantical and Ontological Considerations. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI)*. North-Holland, 183–190.

A. Prasad Sistla and Edmund M. Clarke. 1985. The Complexity of Propositional Linear Temporal Logics. *J. ACM* 32, 3 (July 1985), 733–749.

Andrzej Szalas. 1995. Temporal Logic of Programs: A Standard Approach. In *Time and Logic. A Computational Approach*. UCL Press Ltd., 1–50.

Andrzej Szalas and Leszek Holenderski. 1988. Incompleteness of First-Order Temporal Logic with Until. *Theoretical Computer Science* 57 (1988), 317–325.

Chih-Sung Tang. 1983. Toward a Unified Logical Basis for Programming Languages. In *Proceedings of the 9th World Computer Congress on Information Processing (IFIP–International Federation for Information Processing)*. North-Holland/IFIP, 425–429.

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its application. *Pacific J. Math.* 5 (1955), 285–309.

William W. Wadge. 1988. Tense logic programming: a respectable alternative. In *Proceedings of the International Symposium on Lucid and Intensional Programming*. 26–32.

Xiaoxiao Yang and Zhenhua Duan. 2008. Operational Semantics of Framed Tempura. *The Journal of Logic and Algebraic Programming* 78, 1 (November 2008), 22–51.

Xiaoxiao Yang, Zhenhua Duan, and Qian Ma. 2010. Axiomatic semantics of projection temporal logic programs. *Mathematical Structures in Computer Science* 20, 5 (October 2010), 865–914.

Received January 2012; revised Month 2012; accepted Month 2012