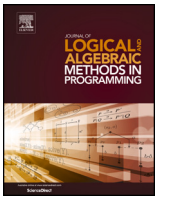
J. Log. Algebraic Methods Program. 148 (2026) 101082

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



Counterpart-based Quantified Temporal Logics

Fabio Gadducci [a,1], Andrea Laretto [b,*], Davide Trotta [a,1]

[a] Department of Computer Science, University of Pisa, Pisa, Italy
[b] Department of Software Science, Tallinn University of Technology, Tallinn, Estonia

ABSTRACT

The aim of this work is to present counterpart-based quantified temporal logics from several points of view. We start by introducing a set-based semantics for a (first-order) linear temporal logic based on the counterpart paradigm, along with results on its positive normal form both in the case of partial functions and of (possibly duplicating) relations. Then, a categorical semantics of the logic is introduced by means of relational presheaves. Both the presentations of the logic via the positive normal form and its categorical semantics are formalised using the proof assistant Agda, and we highlight the crucial aspects of our implementation and the practical use of (quantified) temporal logics in a constructive proof assistant.

Contents

* Corresponding author.
E-mail addresses: fabio.gadducci@unipi.it (F. Gadducci), andrea.laretto@taltech.ee (A. Laretto), trottadavide92@gmail.com (D. Trotta).

[1] Research partially supported by the University of Pisa project PRA_2022_99 “FM4HD” and by the Italian MUR project PRIN 20228KXFN2 “STENDHAL”.

https://doi.org/10.1016/j.jlamp.2025.101082
Received 11 May 2023; Received in revised form 26 June 2025; Accepted 4 August 2025

Available online 13 August 2025
2352-2208/© 2025 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

1. Introduction

During the design of hardware and software for complex systems, increasingly more time and effort is spent on the verification of the desired mechanisms rather than in their actual construction. Formal methods provide an effective framework to verify the correctness of these computational devices and ensure that they satisfy a set of desired specifications. Among the many tools, temporal logics have proven to be one of the most effective techniques for the verification of both large-scale and stand-alone programs, see e.g. the standard textbook [1] and the examples and references therein.

After the foundational work by Pnueli [2], the research on temporal logics focused on both algorithmic procedures for the verification of properties as well as on finding sufficiently expressive fragments of these logics suitable for the specification of complex multi-component systems.

Several models for temporal logics have been developed, with the leading example being the notion of transition systems, also known as Kripke frames: a set of states, each one representing a configuration of the system, and a relation among them, each one identifying a possible state evolution. Often one is interested in enriching both states and transitions with more structure, for example by taking states as algebras and transitions as algebra homomorphisms. A prominent use case of these models is the one exploiting graph logics [3,4], where states are specialised as graphs and transitions are families of (partial) graph morphisms. These logics may combine temporal and spatial reasoning and allow to express the possible transformations of the topology of a graph over time, see [5,6] for two early entries.

1.1. Quantified temporal logics

In classical temporal logics, such as LTL and CTL [7], the states of the model are taken as atomic. Instead, one of the defining characteristics of graph logics is that they permit reasoning and expressing properties on the individual elements of the graph or the algebraic structure being considered. Despite their undecidability [8,9], quantified temporal logics have been advocated in this setting due to their expressiveness and the possibility for quantification to range over the elements in the states of the model.

Unfortunately, the semantical models of these logics are not clearly cut. Consider for example a simple model with two states s_0, s_1, two transitions $s_0 \to s_1$ and $s_1 \to s_0$, and an item i that appears only in s_0. Is the item i being destroyed and recreated again and again, or is it just an identifier being reused multiple times? This issue is denoted in the literature as the *trans-world identity problem* [10,11]. The typical solution provided by the "Kripke semantics" consists in fixing a single set of universal items, which gives identity to each individual appearing in the states of the model. Since each item i belongs to this universal domain, it is exactly the same individual after every temporal evolution in s_1. However, this means that transitions basically behave as injections among the items of the states, and this view is conceptually difficult to reconcile with the simple model sketched above where we describe the destruction and recreation of a given item. Similarly, the possibility of cloning items is then ruled out, since it is impossible to accomodate it with the idea of evolution steps as injections.

1.2. Counterpart semantics

A solution to this problem was proposed by Lewis [12] with the *counterpart paradigm*: instead of a universal set of items, each state identifies a local set of elements, and (possibly partial) morphisms connect them by carrying elements from one state to the other. This allows to speak formally about entities that are destroyed, duplicated, (re)created, or merged, and to adequately deal with the identity problem of individuals between worlds.

In [13], the idea of a counterpart-based semantics is used to introduce a set-theoretical presentation of a μ-calculus with second-order quantifiers. This modal logic provides a sufficiently expressive and general presentation that enriches states with algebras and transitions with partial homomorphisms, thus also subsuming the case of graph logics.

These semantics and models are generalised to a categorical setting in [14] by means of relational presheaves, building on the ideas presented in [15,16]. The models are represented with categories and (families of) relational presheaves, which are used to give a categorical representation for the states-as-algebras approach with partial homomorphisms. The notion of temporal advancement of a system is captured by equipping categories with a set of *one-step* arrows of the model called *temporal structure*, and the categorical framework is used to introduce a second-order linear temporal logic QLTL.

1.3. Contributions

A first contribution of this work is to provide a comprehensive presentation and introduction to the setting of counterpart models and quantified temporal logics, as they are presented in [13,14].

Classical semantics and positive normal form. We start in Section 2 by introducing the semantics of our main temporal logic QLTL with a standard set-based perspective, with satisfiability being defined inductively as a logical predicate. Unlike [13,14], where the models and semantics are defined using *partial functions*, we generalise our case to the setting of *relations*, thus modelling the duplication of elements and allowing for an element to have multiple counterparts in the next world. We conclude the chapter by giving some results and equivalences on the positive normal form presentation of this logic, considering both the cases where the models use partial morphisms and relations and highlighting their differences. Positive normal forms (i.e., where negation is defined only for atomic formulae) are a standard tool of temporal logics, since they simplify its theoretical treatment as well as its model checking algorithms [17,18]. The use of relations instead of (possibly partial) functions weakens the expressiveness of such normal forms, and requires the introduction of additional operators for the logics. However, the duplication of individuals is a central feature of graph transformation formalisms such as Sequi-Pushout [19], and thus worthy of investigation. Both the classical semantics and the positive normal form results have been formalised using the dependently typed proof assistant Agda [20].

Categorical semantics. In Section 3 we introduce the categorical semantics for QLTL. We first motivate the use of the categorical formalism and of relational presheaves with a standard non-algebraic case. We then present the semantics of the logic using the notion of classical attribute [15,16], following the intuition that the meaning of a formula is identified with the set of individuals satisfying it in each world. Finally, we discuss the mechanisms required to extend the categorical presentation and its semantics to the general case of states as algebras and relational homomorphisms between them.

Agda formalisation. An additional contribution presented in this work is a computer-assisted formalisation in Agda of the categorical notions and constructions just presented. We give an overview of the general aspects of the formalisation in Section 4, and highlight the key definitions of the work in Section 5. Providing a mechanised presentation of these constructions has several advantages:

- Formalising the paper further solidifies the correctness and coherence of the mathematical ideas presented in the work, as they can be independently inspected and verified concretely by means of a software tool.
- Given the constructive interpretation of the formalisation, by following the work we essentially codified a procedure to convert classical set-theoretical notions into categorical ones, providing concrete witnesses of how the constructions work for any given setting.
- A formal presentation of modal and temporal logics effectively provides a playground in which the mechanisms and the validity of these logics can be expressed, tested, and experimented with.

To the best of our knowledge, few and sparse formalizations of temporal logics have been provided with a proof assistant, and a systematic study of formally-presented temporal logics and their mechanisation aspects is absent in the literature. This work constitutes a step towards the machine-verified use of temporal logics by embedding in an interactive proof assistant a relatively complex quantified extension of LTL that can reason on the individual elements of states. This formalisation work employs the library `agda-categories` [21], a proof-relevant category theory library for Agda, as a practical foundation to formalise the results in [14] on temporal logics and their models. Aside from the theoretical results and constructions provided by the library, our work witnesses the usefulness and flexibility of `agda-categories` from the point of view of practical applications.

1.4. Comparison with previous works

Following the advice of the second reviewer, we highlighted in Section 1.4 the comparison between the material present in this work and the previous ones on the topic of counterpart-based logics, where the main innovations are indeed, as mentioned, the

formalisation work of the categorical semantics in Agda, the generalisation of the categorical semantics to include the relational one, and a comprehensive overview and comparison of the different semantic perspectives on this logic via a more consistent and unified presentation.

The paper draws from [14] and [22]. More precisely, [14] introduces a categorical presentation of counterpart semantics based on partial functions. In this work, we drop the second-order aspects of the logic discussed there, and focus on extending the categorical semantics from morphisms to relations. These results are in Section 2 and Section 3. The work in [22] presents the Agda implementation of the set-theoretical semantics for a logic based on the two-sorted signature of graphs and graph morphisms. In this work we generalize the formalization in Section 4 and Section 5 to multi-sorted ones on arbitrary signatures, and by adding a formalisation of the categorical semantics using the `agda-categories` library.

2. Quantified Temporal Logics

In this chapter we introduce the counterpart paradigm and define the class of models later used for our logic. We then provide the syntax and semantics of our main first-order linear temporal logic QLTL by adopting a standard set-based presentation, and conclude by introducing a positive normal form of this logic along with equiexpressivity results. Both the theoretical constructions and the positive normal form results have been defined and checked in Agda, and we provide pointers to the formalisation files in each definition. The formalisation of the set-based QTL semantics and of the positive normal form results is available at [23].

2.1. Counterpart semantics

We start by recalling the notion of *Kripke frame* as widely known in modal logic [11,24] and extend it for the case of counterpart semantics.

Definition 2.1. A **Kripke frame** is a 3-tuple $\langle W, U, R, D\rangle$ defined as

- W is a non-empty set;
- U is a set of elements, called *global domain*;
- R is a binary relation on W;
- D is a function assigning to any $\omega \in W$ a set $D(\omega) \subseteq U$ called *domain*.

The set W is interpreted as the set of all *possible worlds*, whereas the binary relation R represents an *accessibility relation* among worlds, connecting them whenever a transition from a world to another is possible. A *domain* $D(\omega)$ identifies the individuals that exist locally in the world ω: individuals in different worlds are identified together by the fact that they all belong to U.

A crucial development in the presentation of Kripke models was introduced by Lewis [12] with the notion of *counterpart relations* and the subsequent introduction of counterpart theory. The idea is to tackle the trans-world identity problem by rejecting strict identity of individuals belonging to a global domain, and instead employing the notion of *counterpart relation* between worlds to connect the individuals that are preserved from one world to the next one. Inspired by Lewis's approach, a more general notion of counterpart model is considered in [13], where worlds are related through *multiple* accessibility relations, and each instance of the accessibility relation is equipped with a counterpart relation.

Definition 2.2. A **counterpart model** is a 3-tuple $\langle W, D, C\rangle$ such that

- W and D are defined as for Kripke frames;
- C is a function assigning to every 2-tuple $\langle \omega, \omega'\rangle$ a set of relations $C\langle \omega, \omega'\rangle \in \mathcal{P}(\mathcal{P}(D(\omega) \times D(\omega')))$, where $\mathcal{P}$ denotes the powerset, and every element $C \in C\langle \omega, \omega'\rangle$ is a relation $C \subseteq D(\omega) \times D(\omega')$. We call these partial functions **atomic (or one-step) counterpart relations**.

Given two worlds ω and ω', the set $C\langle \omega, \omega'\rangle$ is the collection of atomic transitions from ω to ω', defining the possible ways we can access worlds with a *one-step transition* in the system. When the set $C\langle \omega, \omega'\rangle$ is empty, there are no atomic transitions from ω to ω'.

Each atomic counterpart relation $C \in C\langle \omega, \omega'\rangle$ connects the individuals between two given worlds ω and ω', intuitively identifying them as the same element after a single evolution of the model. In particular, if we consider two elements $s \in D(\omega)$ and $s' \in D(\omega')$ and a relation $C \in C\langle \omega, \omega'\rangle$, if $\langle s, s'\rangle \in C$ then s' represents a future development of s via C.

The use of relations allows us to model the notion of removal of an element, which is represented by having no counterpart in the next state. For example, if there is no element $s' \in D(\omega')$ such that $\langle s, s'\rangle \in C$, then we can conclude that the element s has been deallocated by C. Similarly, the duplication of an element can be represented by connecting it with two instances of the counterpart relation, for example by having two elements $s'_1, s'_2 \in D(\omega')$ such that $\langle s, s'_1\rangle \in C$ and $\langle s, s'_2\rangle \in C$.

Now we formally introduce counterpart relations, fixing notation for the rest of the work. We indicate composition of relations in diagrammatic order: as an example, given $C_1 \subseteq A \times B$ and $C_2 \subseteq B \times C$, the composite relation is denoted with $C_1; C_2 = \{(a, c) \mid \exists b.\ \langle a, b\rangle \in C_1 \wedge \langle b, c\rangle \in C_2\} \subseteq A \times C$.

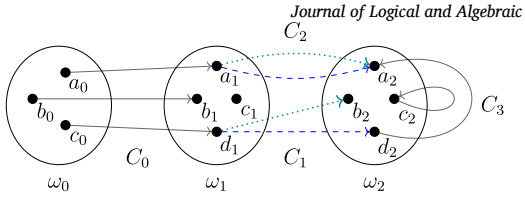


Fig. 1. An example of counterpart model. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Definition 2.3. A relation $C \subseteq D(\omega) \times D(\omega')$ is a **counterpart relation** if one of the following three cases holds

- C is the identity relation;
- $C \in \mathcal{C}\langle \omega, \omega' \rangle$ is a one-step counterpart relation given by the model;
- C is the composite relation of a sequence $C_0; C_1; \cdots; C_n$ for $C_i \in \mathcal{C}\langle \omega_i, \omega_{i+1} \rangle$.

We remark here that the resulting composition $C_1; C_2 \subseteq D(\omega_1) \times D(\omega_3)$ of two atomic counterpart relations $C_1 \in \mathcal{C}\langle \omega_1, \omega_2 \rangle$ and $C_2 \in \mathcal{C}\langle \omega_2, \omega_3 \rangle$ might not necessarily be an atomic counterpart relation, and the model only identifies atomic transitions. This intuitively represents the fact that transitioning through an intermediate state and transitioning directly between worlds can be regarded as two different possibilities, and the direct transition is not necessarily the composition of the two counterpart relations. Moreover, the former requires one evolution step, the latter two.

Definition 2.4. We say that an element $s' \in D(\omega')$ **is the counterpart of** $s \in D(\omega)$ through a counterpart relation C whenever $\langle s, s' \rangle \in C$.

Finally, observe that when each set $\mathcal{C}\langle \omega, \omega' \rangle$ has at most one element, the notion of counterpart model presented in Definition 2.2 becomes a particular case of Lewis's original notion of counterpart frame.

Example 2.1 *(Counterpart model).* In Fig. 1 we provide a graphical presentation of a counterpart model defined by the set of worlds $W := \{\omega_1, \omega_2, \omega_3\}$, where for example $D(\omega_0) = \{a_0, b_0, c_0\}$, $D(\omega_1) = \{a_1, b_1, c_1, d_1\}$, and $D(\omega_2) = \{a_2, b_2, c_2, d_2\}$. The worlds are connected by the following relations: $\mathcal{C}\langle \omega_0, \omega_1 \rangle := \{C_0\}$ is a single counterpart relation C_0 between ω_0 and ω_1, $\mathcal{C}\langle \omega_1, \omega_2 \rangle := \{C_1, C_2\}$ has two possible counterpart relations between ω_1, ω_2, and $\mathcal{C}\langle \omega_2, \omega_2 \rangle = \{C_3\}$ is a looping counterpart relation. We use blue dashed and green dotted lines to distinguish C_1 and C_2, respectively.

2.1.1. Temporal structures

As is the case of LTL, where we can identify traces connecting linearly evolving states (see e.g. [1, Definition 5.7]), we can consider linear sequences of counterpart relations providing a list of sequentially accessible worlds.

Definition 2.5. A **trace** σ on a counterpart model $\langle W, D, \mathcal{C} \rangle$ is an infinite sequence of one-step counterpart relations $(C_0, C_1, \dots)$ such that $C_i \in \mathcal{C}\langle \omega_i, \omega_{i+1} \rangle$ for any $i \geq 0$.

Given a trace $\sigma = (C_0, C_1, \dots)$, we use i as subscript $\sigma_i := (C_i, C_{i+1}, \dots)$ to denote the trace obtained by excluding the first i counterpart relations. We use $\omega_0, \omega_1, \dots$ and ω_i to indicate the worlds provided by the trace σ whenever it is clear from the context.

Since a trace $\sigma = (C_0, C_1, \dots)$ provides a sequence of counterpart relations step-by-step connected through a world, we denote with $C_{\leq i}$ the composite relation $C_0; \cdots; C_{i-1}$ from the first world ω_0 up to the i-th world ω_i through the relations given by the trace σ. In the edge case $i = 0$, the relation $C_{\leq 0}$ is defined to be the identity relation on ω_0.

2.2. Quantified linear temporal logic

In this section we present the syntax and semantics of our quantified linear temporal logic QLTL. We will assume a fixed counterpart model $\langle W, D, \mathcal{C} \rangle$, with definitions referring to the data provided by the underlying model.

2.2.1. Syntax and semantics of QLTL

To have a simpler presentation, it is customary to exclude the elementary constructs that can be expressed in terms of other operators, such as conjunction and universal quantification. Thus, we initially present QLTL with a minimal set of standard operators and derive other ones with negation.

Definition 2.6 *(QLTL).* Let $\mathcal{X}$ be a set of variables with $x, y \in \mathcal{X}$ and $\mathcal{P}$ a set of (unary) predicates with $P \in \mathcal{P}$. The set $\mathcal{F}^{\mathsf{QLTL}}$ of QLTL formulae is generated by the following rules

$$\psi ::= \mathsf{true} \mid x = y \mid P(x)$$

$$\phi ::= \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists x.\phi \mid \mathsf{O}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{W} \phi_2$$

The *next* operator $\mathsf{O}\phi$ expresses the fact that a certain property ϕ has to be true at the next state. The *until* operator $\phi_1 \mathsf{U} \phi_2$ indicates that the property ϕ_1 has to hold at least until the property ϕ_2 becomes true, which must hold at the present or future time. Finally, the *weak until* operator $\phi_1 \mathsf{W} \phi_2$ is similar to the $\phi_1 \mathsf{U} \phi_2$ operator, but allows for counterparts to always exist while satisfying ϕ_1 without ever reaching a point where ϕ_2 holds.

We use the letter ψ to indicate the case of elementary predicates and we refer to these formulae as *atomic formulae*. Given the variables $x, y \in \mathcal{X}$ denoting two individuals, the formula $x = y$ indicates that the two individuals coincide in the current world. Finally, our logic is extended with unary predicate symbols $P(x)$ that will be used in the running example in Fig. 2. The usual dual operators can be syntactically expressed by taking $\mathsf{false} := \neg\mathsf{true}$, $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$, and $\forall x.\phi := \neg\exists x.\neg\phi$. Note that, differently from classical LTL, the until and the weak until operators are not self-dual: this fact will be discussed and made explicit in Remark 2.4.

Example 2.2 *(Deallocation).* As we anticipated in Section 2.1, one of the main advantages of a counterpart semantics is the possibility to reason about existence, deallocation, duplication and merging elements of a system. For example, we can capture a notion of existence of an element at the current moment with the shorthand

$$\mathbf{present}(x) := \exists y.x = y$$

We combine this predicate with the *next* operator to talk about elements that are present in the current world and that will still be present at the next step, for example with the formula

$$\mathbf{nextStepPreserved}(x) := \mathbf{present}(x) \wedge \mathsf{O}\mathbf{present}(x)$$

Similarly, we can refer to elements that are now present but that will be deallocated at the next step by considering

$$\mathbf{nextStepDeallocated}(x) := \mathbf{present}(x) \wedge \neg\mathsf{O}\mathbf{present}(x)$$

2.2.2. Contexts and assignments

Since free variables referring to individuals can now appear inside formulae, we recall the usual presentation of context and formulae-in-context as similarly defined in [13].

Definition 2.7 *(Context).* A **context** Γ over a set of variables $\mathcal{X}$ is a finite subset of $\mathcal{X}$. We use the notation Γ, x to indicate the augmented context $\Gamma \cup \{x\}$, with the empty context being indicated as $\emptyset$.

Definition 2.8 *(Formulae-in-context).* A **formula-in-context** is a formula ϕ along with an associated context Γ that contains all the free variables of the formula ϕ (and possibly more), and we indicate this decoration with $[\Gamma]\phi$.

We omit the bracketed context whenever it is unnecessary to specify it.

To properly present the notion of satisfiability of a formula-in-context with respect to a given counterpart model, we need to first introduce the definition of *assignment* in a given world.

Definition 2.9 *(Assignment).* An **assignment** in the world $\omega \in W$ for the context Γ is a function $\mu : \Gamma \to D(\omega)$. We use the notation $\mathcal{A}^{\Gamma}_{\omega}$ to indicate the set of assignments μ defined in ω for the context Γ.

Moreover, we denote by $\mu[x \mapsto s] : \Gamma, x \to D(\omega)$ the assignment obtained by extending the domain of μ with $s \in D(\omega)$ at the variable $x \notin \Gamma$.

We now define the lifting of counterpart relations to assignments. The intuition is that we want to transfer all elements of an assignment to the next world using the counterpart relation individual-by-individual.

Definition 2.10 *(Counterpart relations on assignments).* Given a counterpart relation $C \subseteq D(\omega_1) \times D(\omega_2)$ and two assignments $\mu_1 : \Gamma \to D(\omega_1)$ and $\mu_2 : \Gamma \to D(\omega_2)$ defined on the same context Γ, we say that the assignments μ_1 and μ_2 are **counterpart related** whenever $\langle \mu_1(x), \mu_2(x) \rangle \in C$ for all variables $x \in \Gamma$, and we indicate this simply with the notation $\langle \mu_1, \mu_2 \rangle \in C$.

2.2.3. Satisfiability

We now introduce the notion of satisfiability of a formula with respect to a given trace and assignment.

Definition 2.11 *(QLTL satisfiability).* Given a QLTL formula-in-context $[\Gamma]\phi$, a trace $\sigma = (C_0, C_1, \dots)$, an interpretation $P(\omega_i) \subseteq D(\omega_i)$ for all the predicates $P \in \mathcal{P}$ and the worlds in σ, and an assignment $\mu : \Gamma \to D(\omega_0)$ for the first world of σ, we inductively define the *satisfiability relation* as follows

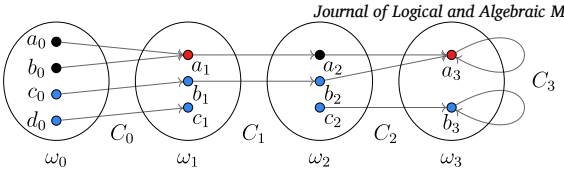


Fig. 2. An example with four worlds $\omega_0, \omega_1, \omega_2, \omega_3$.

- $\sigma, \mu \vDash \mathsf{true}$;
- $\sigma, \mu \vDash x = y$ if $\mu(x) = \mu(y)$;
- $\sigma, \mu \vDash P(x)$ if $\mu(x) \in P(\omega_0)$;
- $\sigma, \mu \vDash \neg\phi$ if $\sigma, \mu \nvDash \phi$;
- $\sigma, \mu \vDash \phi_1 \vee \phi_2$ if $\sigma, \mu \vDash \phi_1$ or $\sigma, \mu \vDash \phi_2$;
- $\sigma, \mu \vDash \exists x.\phi$ if there is an individual $s \in D(\omega_0)$ such that $\sigma, \mu[x \mapsto s] \vDash \phi$;
- $\sigma, \mu \vDash \mathsf{O}\phi$ if there is $\mu_1 \in \mathcal{A}^{\Gamma}_{\omega_1}$ such that $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$;
- $\sigma, \mu \vDash \phi_1 \mathsf{U} \phi_2$ if there is $\bar{n} \geq 0$ such that

  1. for any $i < \bar{n}$, there is $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi_1$;
  2. there is $\mu_{\bar{n}} \in \mathcal{A}^{\Gamma}_{\omega_{\bar{n}}}$ such that $\langle \mu, \mu_{\bar{n}} \rangle \in C_{\leq \bar{n}}$ and $\sigma_{\bar{n}}, \mu_{\bar{n}} \vDash \phi_2$;

- $\sigma, \mu \vDash \phi_1 \mathsf{W} \phi_2$ if one of the following holds

  – the same conditions for $\phi_1 \mathsf{U} \phi_2$ apply;
  – for any i there is $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi_1$.

Example 2.3. We present a running example in Fig. 2 to better describe the expressiveness of QLTL and to illustrate the mechanisms of working in a counterpart-based semantics. We consider a fixed trace $\sigma = (C_0, C_1, C_2, C_3, C_3, \dots)$ and we indicate with $\mathsf{B}(x)$ and $\mathsf{R}(x)$ the unary predicates that hold for any individual coloured in blue and red, respectively. As a concrete scenario for the temporal operators $\mathsf{O}\phi$ and $\phi_1 \mathsf{U} \phi_2$ we presented in Definition 2.11, we have for example that $\sigma, \{x \mapsto a_0\} \vDash \mathsf{O}(\mathsf{R}(x))$ and $\sigma, \{x \mapsto c_0\} \vDash \mathsf{B}(x) \mathsf{U} \mathsf{R}(x)$. Also, we have that a_0 is preserved at the next step with $\sigma, \{x \mapsto a_0\} \vDash \mathbf{nextStepPreserved}(x)$, whereas c_1 is removed and indeed $\sigma_1, \{x \mapsto c_1\} \vDash \mathbf{nextStepDeallocated}(x)$.

Remark 2.1 *(*Eventually *and* always *operators).* As in LTL, we can define the additional *eventually* $\Diamond\phi$ and *always* $\Box\phi$ operators as $\Diamond\phi := \mathsf{true} \mathsf{U} \phi$ and $\Box\phi := \phi \mathsf{W} \mathsf{false}$, respectively. Their semantics can be presented directly as

- $\sigma, \mu \vDash \Diamond\phi$ if there is $i \geq 0$ and $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi$;
- $\sigma, \mu \vDash \Box\phi$ if for any $i \geq 0$ there is $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi$.

In our example in Fig. 2, we have for instance that $\sigma, \{x \mapsto c_0\} \vDash \Diamond \mathsf{R}(x)$ but $\sigma, \{x \mapsto d_0\} \nvDash \Diamond \mathsf{R}(x)$ and similarly $\sigma_2, \{x \mapsto c_2\} \nvDash \Diamond \mathsf{R}(x)$. Moreover, we have that $\sigma, \{x \mapsto c_0\} \vDash \Diamond\Box \mathsf{R}(x)$ and $\sigma_2, \{x \mapsto c_2\} \vDash \Box \mathsf{B}(x)$. However, $\sigma, \{x \mapsto d_0\} \nvDash \Box \mathsf{B}(x)$ since a counterpart is always required to exist.

Example 2.4 *(Merging).* In QLTL we can express the merging of two individuals at some point in the future with the predicate

$$\mathbf{willMerge}(x, y) := x \neq y \wedge \Diamond(x = y).$$

In our example in Fig. 2, we have that in the first world $\sigma, \{x \mapsto a_0, y \mapsto c_0\} \vDash \mathbf{willMerge}(x, y)$, but clearly $\sigma, \{x \mapsto c_0, y \mapsto d_0\} \nvDash \mathbf{willMerge}(x, y)$.

Remark 2.2 *(Quantifier elision for unbound variables).* A relevant difference with standard quantified logics is that in QLTL we cannot elide quantifications where the introduced variable does not appear in the sub-formula. Assuming $\equiv$ to denote semantical equivalence and taking any ϕ with $x \notin \mathrm{fv}(\phi)$, we have that in general $\exists x.\phi \not\equiv \phi$ and, similarly, $\forall x.\phi \not\equiv \phi$. More precisely, the above equivalences hold if ϕ contains no temporal operator and the current world $D(\omega)$ is not empty. A similar phenomenon arises in intuitionistic and constructive logic: given a type τ and a formula ϕ where x does not appear free, the formula $\exists(x : \tau).\phi$ is not equivalent to ϕ since the existential quantification implicitly carries the information that the type τ is inhabited.

We describe here a concrete example: consider a world ω with a single individual $D(\omega) = \{s\}$ and a looping counterpart relation $C\langle \omega, \omega \rangle = \{C\}$, where $C = \emptyset$ is the empty counterpart relation. The trace is given by $\sigma = (C, C, \dots)$. By taking the empty assignment $\{\,\}$ and the closed formula $\phi = \mathsf{O}(\mathsf{true})$, one can easily check that $\sigma, \{\,\} \vDash \mathsf{O}(\mathsf{true})$, but $\sigma, \{\,\} \nvDash \exists x.\mathsf{O}(\mathsf{true})$. The reason is that, once an assignment is extended with some element, stepping from one world to the next one requires every individual of the assignment to be preserved and have a counterpart in the next world.

Alternatively, we could have restricted assignments in the semantics so that counterparts are required only for the free variables occurring in the formula. For example, the definition for the *next* $\mathsf{O}\phi$ operator would become

- $\sigma, \mu \vDash \mathsf{O}\phi$ if there is $\mu_1 \in \mathcal{A}_{\omega_1}^{\mathrm{fv}(\phi)}$ such that $\langle \mu_{|\mathrm{fv}(\phi)}, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$

For ease of presentation both in this work and with respect to our Agda implementation, we consider the case where all elements in the context must have a counterpart. Moreover, this alternative definition would not naturally align with the categorical semantics presented in Section 3: the intuition is that, for any given world ω and counterpart relation R from it, the cartesian product of presheaves in that world has a counterpart through R if and only if *every* element of the product has a counterpart through R.

Remark 2.3 *(No self-duality for* next*).* We observe that, contrary to classical LTL, the *next* $\mathsf{O}\phi$ operator in our counterpart-style semantics in general is not self-dual with respect to negation, i.e. $\neg\mathsf{O}\phi \not\equiv \mathsf{O}\neg\phi$. As we will see in Section 2.3, to provide a positive normal form for QLTL it is necessary to introduce a separate next operator that allows us to adequately capture the notion of negation. This absence of duality is again due to the fact that we use relations in our counterpart model, which forces us to talk about the existence as well as the possible *absence* of a counterpart.

Consider the counterpart model in Fig. 2: it is easy to see that $\sigma_1, \{x \mapsto c_1\} \vDash \neg\mathsf{O}(\mathsf{B}(x))$, but $\sigma_1, \{x \mapsto c_1\} \nvDash \mathsf{O}(\neg\mathsf{B}(x))$ since no counterpart for c_1 exists after one step. The idea is that, since the *next* operator requires a counterpart at the next step to exist, its negation must express that either all counterparts at the next step do not satisfy the formula or that a counterpart does not exist altogether.

Remark 2.4 *(*Until *and* weak until *are incompatible).* In standard LTL, the *until* $\phi_1 \mathsf{U} \phi_2$ and *weak until* $\phi_1 \mathsf{W} \phi_2$ operators have the same expressivity, and can be defined in terms of each other by the equivalences

$$\phi_1 \mathsf{U} \phi_2 \equiv_{\mathsf{LTL}} \neg(\neg\phi_2 \mathsf{W}(\neg\phi_1 \wedge \neg\phi_2))$$
$$\phi_1 \mathsf{W} \phi_2 \equiv_{\mathsf{LTL}} \neg(\neg\phi_2 \mathsf{U}(\neg\phi_1 \wedge \neg\phi_2))$$

However, this is *not* the case in QLTL. Similarly, in QLTL we have that $\Box\phi \not\equiv \neg\Diamond\neg\phi$, as for the semantics provided in Remark 2.1. This characteristic of QLTL is again due to the fact we are working in the setting of (possibly deallocating) relations, and we will formally explain and present an intuition for this when we introduce the semantics of positive normal forms for QLTL in Section 2.3. The usual equivalences for LTL can be obtained by restricting to models whose counterpart relations are total functions: this allows us to consider a unique trace of always-defined counterpart individuals, which in turn brings our models back to a notion similar to LTL traces.

2.3. Positive normal form for QLTL

Positive normal forms are a standard presentation of temporal logics and can be used to simplify constructions and algorithms on both the theoretical and implementation side [18,17]. This presentation is crucial to define the semantics of a logic based on fixed points, such as in [13], while still preserving the full expressiveness of the original presentation. As we will remark in Section 4, explicitly providing a negation-free semantics for our logic also ensures that it can be more easily manipulated in a proof assistant where definitions and proofs are *constructive*. In this section we present an explicit semantics for the positive normal form of QLTL, which we denote as PNF.

2.3.1. Semantics of PNF

As observed in Remark 2.3 and Remark 2.4, to present the positive normal form we need additional operators to adequately capture the negation of each of the temporal operators previously described. Thus, we introduce a new flavour of the next operator, called *next-forall* $\mathsf{A}\phi$. Similarly, we introduce a negative dual for the *until* $\phi_1 \mathsf{U} \phi_2$ and *weak until* $\phi_1 \mathsf{W} \phi_2$ operators, which we indicate as the *then* $\phi_1 \mathsf{T} \phi_2$ and *until-forall* $\phi_1 \mathsf{F} \phi_2$ operators, respectively.

Definition 2.12 *(QLTL in PNF).* Let $\mathcal{X}$ be a set of variables with $x, y \in \mathcal{X}$ and $\mathcal{P}$ be a set of (unary) predicates with $P \in \mathcal{P}$. The set $\mathcal{F}^{\mathsf{PNF}}$ of formulae of QLTL in **positive normal form** is generated by the following rules

$$\psi ::= \mathsf{true} \mid x = y \mid P(x)$$
$$\phi ::= \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi \mid \mathsf{O}\phi \mid \mathsf{A}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{F} \phi_2 \mid \phi_1 \mathsf{W} \phi_2 \mid \phi_1 \mathsf{T} \phi_2$$

We now provide a satisfiability relation for PNF formulae by specifying the semantics for the additional operators, omitting the ones that do not change.

Definition 2.13 *(QLTL in PNF satisfiability).* We inductively define the satisfiability relation for the additional constructs as follows

- $\sigma, \mu \vDash \neg\psi$ if $\sigma, \mu \nvDash \psi$;
- $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ if $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;

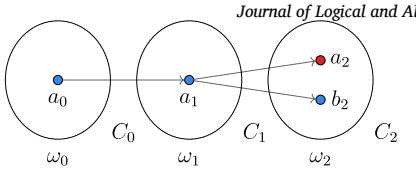


Fig. 3. A counterpart model where $\neg(\mathsf{B}(x)\mathsf{T}\mathsf{R}(x)) \not\equiv (\neg\mathsf{R}(x))\mathsf{U}(\neg\mathsf{B}(x) \wedge \neg\mathsf{R}(x))$.

- $\sigma, \mu \vDash \forall x.\phi$ if for any $s \in D(\omega_0)$ we have that $\sigma, \mu[x \mapsto s] \vDash \phi$;
- $\sigma, \mu \vDash \mathsf{A}\phi$ if for any $\mu_1 \in \mathcal{A}^{\Gamma}_{\omega_1}$ such that $\langle \mu, \mu_1 \rangle \in C_0$ we have that $\sigma_1, \mu_1 \vDash \phi$;
- $\sigma, \mu \vDash \phi_1 \mathsf{F} \phi_2$ if there is an $\bar{n} \geq 0$ such that

  1. for any $i < \bar{n}$ and $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have $\sigma_i, \mu_i \vDash \phi_1$;
  2. for any $\mu_{\bar{n}} \in \mathcal{A}^{\Gamma}_{\omega_{\bar{n}}}$ such that $\langle \mu, \mu_{\bar{n}} \rangle \in C_{\leq \bar{n}}$ we have that $\sigma_{\bar{n}}, \mu_{\bar{n}} \vDash \phi_2$;.

- $\sigma, \mu \vDash \phi_1 \mathsf{T} \phi_2$ if one of the following holds

  – the same conditions for $\phi_1 \mathsf{F} \phi_2$ apply;
  – for any i and $\mu_i \in \mathcal{A}^{\Gamma}_{\omega_i}$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \vDash \phi_1$.

The intuition for the *next-forall* $\mathsf{A}\phi$ operator is that it allows us to capture the case where a counterpart of an individual does not exist at the next step: if any counterpart exists, it is required to satisfy the formula ϕ.

Similarly to the *until* $\phi_1 \mathsf{U} \phi_2$ operator, the *until-forall* $\phi_1 \mathsf{F} \phi_2$ operator allows us to take a sequence of worlds where ϕ_1 is satisfied for some steps until ϕ_2 holds. The crucial observation is that all the intermediate counterparts satisfying ϕ_1 and the conclusive counterparts must satisfy ϕ_2. Such counterparts are not required to exist, and indeed any trace consisting of all empty counterpart relations always satisfies both $\phi_1 \mathsf{F} \phi_2$ and $\phi_1 \mathsf{T} \phi_2$.

Similarly to the *weak until* $\phi_1 \mathsf{W} \phi_2$ operator, the *then* $\phi_1 \mathsf{U} \phi_2$ operator corresponds to a *weak until-forall*, where the formula can be validated by a trace where all counterparts satisfy ϕ_1 without ever satisfying ϕ_2.

Example 2.5 *(*Until-forall*, then, and* next-forall*).* In our running example in Fig. 2, we illustrate the possibility for $\mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$ and $\mathsf{A}\mathsf{B}(x)$ to be satisfied even when a counterpart does not exist after one or more steps. In particular, it can be verified that $\sigma, \{x \mapsto c_0\} \vDash \mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$ holds since $\mathsf{R}(x)$ is eventually satisfied while $\mathsf{B}(x)$ holds, just like the *until* operator. We have that both $\sigma, \{x \mapsto a_0\} \vDash \mathsf{A}\mathsf{R}(x)$ and $\sigma_1, \{x \mapsto c_1\} \vDash \mathsf{A}\mathsf{R}(x)$ hold, since no counterpart for c_1 exists after one step. Finally, we have that $\sigma, \{x \mapsto d_0\} \vDash \mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$ holds since $\mathsf{B}(x)$ holds but no counterpart exists after two steps, and $\sigma_2, \{x \mapsto c_2\} \vDash \mathsf{B}(x)\mathsf{T}\mathsf{R}(x)$ since a counterpart always exists but $\mathsf{B}(x)$ holds forever.

2.3.2. Negation of QLTL and PNF

The crucial observation that validates the PNF presented in Section 2.3 is that the negation of *next* $\mathsf{O}\phi$, *until* $\phi_1 \mathsf{U} \phi_2$, and *weak until* $\phi_1 \mathsf{W} \phi_2$ formulae can now be expressed inside the logic. We will explicitly indicate with $\vDash_{\mathsf{QLTL}}$ and $\vDash_{\mathsf{PNF}}$ the satisfiability relations defined for formulae in standard QLTL and QLTL in PNF, respectively.

Proposition 2.1 *(Negation is expressible in PNF).* (Relational.Negation)
Let ψ be an atomic formula in PNF. Then we have

$$\begin{aligned}
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \vDash_{QLTL} \neg \mathsf{O}(\psi) && \iff \quad \sigma, \mu \vDash_{PNF} \mathsf{A}(\neg\psi) \\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \vDash_{QLTL} \neg(\psi_1 \mathsf{U} \psi_2) && \iff \quad \sigma, \mu \vDash_{PNF} (\neg\psi_2)\mathsf{T}(\neg\psi_1 \wedge \neg\psi_2) \\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \vDash_{QLTL} \neg(\psi_1 \mathsf{W} \psi_2) && \iff \quad \sigma, \mu \vDash_{PNF} (\neg\psi_2)\mathsf{F}(\neg\psi_1 \wedge \neg\psi_2).
\end{aligned}$$

However, a converse statement that similarly expresses the negation of these newly introduced operators in PNF does not hold: the only exception is the easy case of the *next-forall* $\mathsf{A}\phi$ operator, whose negation directly corresponds with the *next* $\mathsf{O}\phi$ operator.

Proposition 2.2 *(Negation of new operators is* not *in PNF). Let ψ be an atomic formula in PNF. Then we have*

$$\begin{aligned}
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \nvDash_{PNF} \mathsf{A}(\psi) && \iff \quad \sigma, \mu \vDash_{PNF} \mathsf{O}(\neg\psi) \\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \nvDash_{PNF} \psi_1 \mathsf{T} \psi_2 && \not\iff \quad \sigma, \mu \vDash_{PNF} (\neg\psi_2)\mathsf{U}(\neg\psi_1 \wedge \neg\psi_2) \\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \quad \sigma, \mu \nvDash_{PNF} \psi_1 \mathsf{F} \psi_2 && \not\iff \quad \sigma, \mu \vDash_{PNF} (\neg\psi_2)\mathsf{W}(\neg\psi_1 \wedge \neg\psi_2).
\end{aligned}$$

Proof. We provide a single direct counterexample for both the *then* and *until-forall* cases. Take for example the formula $\mathsf{B}(x)\mathsf{T}\mathsf{R}(x)$ and consider the counterexample in Fig. 3

Clearly, we have that $\sigma, \{x \mapsto a_0\} \models \neg(\mathsf{B}(x)\mathsf{T}\mathsf{R}(x))$. However, $\sigma, \{x \mapsto a_0\} \not\models (\neg\mathsf{R}(x))\mathsf{U}(\neg\mathsf{B}(x) \wedge \neg\mathsf{R}(x))$ since the *until* operator requires a *single* counterpart to exist where both $\neg\mathsf{B}(x)$ and $\neg\mathsf{R}(x)$ after n steps. The case of $\phi_1\mathsf{F}\phi_2$ and its negated form using $\phi_1\mathsf{W}\phi_2$ follows similarly. □

It turns out that we can recover the previous equivalences by considering the case where each counterpart relation is a *partial function*, following the definition of counterpart models given in [14,13].

Proposition 2.3 *(Negation for partial functions).* (Functional.Negation)
Let ψ be an atomic formula in PNF and $\sigma = (C_0, C_1, \dots)$ a trace where each counterpart relation C_i is a partial function $C_i : D(w_i) \rightharpoonup D(w_{i+1})$. Then we have

$$\begin{aligned}
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}.\ \ \sigma, \mu \not\models_{PNF} \mathsf{A}(\psi) &&\iff \sigma, \mu \models_{PNF} \mathsf{O}(\neg\psi)\\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}.\ \ \sigma, \mu \not\models_{PNF} \psi_1\mathsf{T}\psi_2 &&\iff \sigma, \mu \models_{PNF} (\neg\psi_2)\mathsf{U}(\neg\psi_1 \wedge \neg\psi_2)\\
&\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}.\ \ \sigma, \mu \not\models_{PNF} \psi_1\mathsf{F}\psi_2 &&\iff \sigma, \mu \models_{PNF} (\neg\psi_2)\mathsf{W}(\neg\psi_1 \wedge \neg\psi_2).
\end{aligned}$$

Notice how the previous results can be easily generalised to the case where we consider the negation of full formulae ϕ.

The equivalences presented in Proposition 2.1 allow us to define a formal translation $\overline{\cdot} : \mathcal{F}^{\mathsf{QLTL}} \to \mathcal{F}^{\mathsf{PNF}}$ from the QLTL syntax presented in Definition 2.11 to the current one in PNF, preserving the equivalence of formulae. This is done with the obvious syntactical transformation that pushes the negation in QLTL formulae down to elementary predicates and replaces temporal operators with their negated counterpart. For example

$$\begin{array}{rcl@{\qquad}rcl}
\overline{\mathsf{O}\phi} & := & \mathsf{O}\overline{\phi} & \overline{\phi_1\mathsf{U}\phi_2} & := & \overline{\phi_1}\mathsf{U}\overline{\phi_2}\\
\overline{\neg\mathsf{O}\phi} & := & \mathsf{A}\overline{\neg\phi} & \overline{\phi_1\mathsf{W}\phi_2} & := & \overline{\phi_1}\mathsf{W}\overline{\phi_2}\\
\overline{\phi_1 \vee \phi_2} & := & \overline{\phi_1} \vee \overline{\phi_2} & \overline{\neg(\phi_1\mathsf{U}\phi_2)} & := & (\overline{\neg\phi_2})\mathsf{T}(\overline{\neg\phi_1} \wedge \overline{\neg\phi_2})\\
\overline{\neg(\phi_1 \vee \phi_2)} & := & \overline{\neg\phi_1} \wedge \overline{\neg\phi_2} & \overline{\neg(\phi_1\mathsf{W}\phi_2)} & := & (\overline{\neg\phi_2})\mathsf{F}(\overline{\neg\phi_1} \wedge \overline{\neg\phi_2})
\end{array}$$

Theorem 2.1 *(PNF equivalence).* (Relational.Conversion) *Let $\overline{\cdot} : \mathcal{F}^{QLTL} \to \mathcal{F}^{PNF}$ be the aforementioned syntactical translation that replaces negated temporal operators with their equivalent ones in PNF. For any QLTL formula $[\Gamma]\phi \in \mathcal{F}^{QLTL}$ we have*

$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}.\ \ \sigma, \mu \models_{QLTL} \phi \iff \sigma, \mu \models_{PNF} \overline{\phi}.$$

Now that we have defined the complete set of temporal operators, the second condition of *then* $\phi_1\mathsf{T}\phi_2$ can similarly be expressed by a derived *always-forall* $\Box^*\phi$ operator, which we present along with a *eventually-forall* $\Diamond^*\phi$ operator.

Similarly as with the *then* and *until-forall* operators, the difference with their standard versions *eventually* $\Diamond\phi$ and *always* $\Box\phi$ is that they require for all counterparts to satisfy the formula ϕ, if any exists.

Remark 2.5 *(*Eventually-forall *and* always-forall*).* The *eventually-forall* $\Diamond^*\phi$ and *always-forall* $\Box^*\phi$ operators are defined as $\Diamond^*\phi := \mathsf{true}\mathsf{F}\phi$ and $\Box^*\phi := \phi\mathsf{T}\mathsf{false}$, respectively. Their semantics can be explicitly presented as follows

- $\sigma, \mu \models \Diamond^*\phi$ if there is $i \geq 0$ such that for any $\mu_i \in A^{\Gamma}_{w_i}$ with $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \models \phi$;
- $\sigma, \mu \models \Box^*\phi$ if for any i and $\mu_i \in A^{\Gamma}_{w_i}$ with $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \models \phi$.

Proposition 2.4 *(Equivalences between operators in PNF).* (Relational.Equivalences)
The following equivalences hold in PNF

$$\begin{aligned}
&\phi_1\mathsf{U}\phi_2 \equiv \phi_1\mathsf{W}\phi_2 \wedge \Diamond\phi_2 && \phi_1\mathsf{W}\phi_2 \equiv \phi_1\mathsf{U}\phi_2 \vee \Box\phi_1\\
&\phi_1\mathsf{F}\phi_2 \equiv \phi_1\mathsf{T}\phi_2 \wedge \Diamond^*\phi_2 && \phi_1\mathsf{T}\phi_2 \equiv \phi_1\mathsf{F}\phi_2 \vee \Box^*\phi_1.
\end{aligned}$$

Contrary to what happens in LTL, the usual expansion laws where each operator is defined in terms of itself do not hold in QLTL for the case of counterpart relations, as shown by the following result.

Proposition 2.5 *(Expansion laws do* not *hold in QLTL). We have the following statements in PNF*

$$\begin{aligned}
&\phi_1\mathsf{U}\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge \mathsf{O}(\phi_1\mathsf{U}\phi_2)) && \phi_1\mathsf{F}\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge \mathsf{A}(\phi_1\mathsf{F}\phi_2))\\
&\phi_1\mathsf{W}\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge \mathsf{O}(\phi_1\mathsf{W}\phi_2)) && \phi_1\mathsf{T}\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge \mathsf{A}(\phi_1\mathsf{T}\phi_2)).
\end{aligned}$$

Proof. We provide direct counterexamples for the *until* $\phi_1\mathsf{U}\phi_2$ and *until-forall* $\phi_1\mathsf{F}\phi_2$ cases, the *weak until* $\phi_1\mathsf{W}\phi_2$ and *then* $\phi_1\mathsf{T}\phi_2$ cases obviously following.

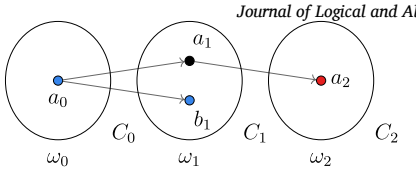


Fig. 4. A counterexample model where, in the case of counterpart relations, we have that $\mathsf{B}(x)\mathsf{U}\mathsf{R}(x) \not\equiv \mathsf{R}(x) \vee (\mathsf{B}(x) \wedge \mathsf{O}(\mathsf{B}(x)\mathsf{U}\mathsf{R}(x)))$.

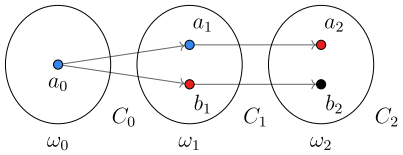


Fig. 5. A counterpart model where $\mathsf{B}(x)\mathsf{F}\mathsf{R}(x) \not\equiv \mathsf{R}(x) \vee (\mathsf{B}(x) \wedge \mathsf{A}(\mathsf{B}(x)\mathsf{F}\mathsf{R}(x)))$.

Consider the *until* case with the formula $\mathsf{B}(x)\mathsf{U}\mathsf{R}(x)$. In the model shown in Fig. 4 we have that $\sigma, \{x \mapsto a_0\} \vDash \mathsf{B}(x)\mathsf{U}\mathsf{R}(x)$ since there is a counterpart after two steps with $\mathsf{R}(x)$ and for all worlds before it there is a counterpart with $\mathsf{B}(x)$. However, clearly a_0 does not satisfy the expanded formula since neither $\sigma_1, \{x \mapsto a_1\} \nvDash \mathsf{B}(x)\mathsf{U}\mathsf{R}(x)$ nor $\sigma_1, \{x \mapsto b_1\} \nvDash \mathsf{B}(x)\mathsf{U}\mathsf{R}(x)$.

Consider the *until-forall* case with the formula $\mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$. In the model shown in Fig. 5 we have that a_0 satisfies the expanded formula, since the one-step counterparts a_1 and b_1 are such that both $\sigma, \{x \mapsto a_0\} \vDash \mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$ and $\sigma, \{x \mapsto a_0\} \vDash \mathsf{B}(x)\mathsf{F}\mathsf{R}(x)$, with the world where all counterparts satisfy $\mathsf{R}(x)$ being reached after two and one steps, respectively. However, we have that $\sigma, \{x \mapsto a_0\} \nvDash \mathsf{A}(\mathsf{B}(x)\mathsf{F}\mathsf{R}(x))$ since there is no single world ω_n where all counterparts after n steps satisfy $\mathsf{R}(x)$. □

Similarly as with Proposition 2.3, we can recover the expansion laws by restricting ourselves to the case of partial functions as counterpart relations.

Proposition 2.6 *(Expansion laws for partial functions).* (Functional.ExpansionLaws)
The following equivalences hold in PNF if we restrict ourselves to traces where each counterpart relation C_i is a partial function $C_i : D(\omega_i) \rightharpoonup D(\omega_{i+1})$

$$\phi_1 \mathsf{U} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \mathsf{O}(\phi_1 \mathsf{U} \phi_2)) \qquad \phi_1 \mathsf{F} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \mathsf{A}(\phi_1 \mathsf{F} \phi_2))$$
$$\phi_1 \mathsf{W} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \mathsf{O}(\phi_1 \mathsf{W} \phi_2)) \qquad \phi_1 \mathsf{T} \phi_2 \equiv \phi_2 \vee (\phi_1 \wedge \mathsf{A}(\phi_1 \mathsf{T} \phi_2)).$$

Remark 2.6 *(Temporal operators as fixed points).* Consider the same counterpart models with partial functions of Proposition 2.6: contrary to the relational case, we recover that the *until* $\phi_1 \mathsf{U} \phi_2$ and *until-forall* $\phi_1 \mathsf{F} \phi_2$ operators correspond to least fixed points of their expansion shown in Proposition 2.6, and *weak until* $\phi_1 \mathsf{F} \phi_2$ and *then* $\phi_1 \mathsf{T} \phi_2$ correspond to greatest fixed points.

Remark 2.7 *(Functional counterparts collapse the semantics).* As briefly mentioned in Remark 2.3, when our counterpart model is restricted to relations that are *total functions* we actually have that the pairs of operators previously introduced collapse and provide the same semantics and dualities of the classical operators. In particular, we obtain that $\mathsf{O}\phi \equiv \mathsf{A}\phi$, $\phi_1 \mathsf{U} \phi_2 \equiv \phi_1 \mathsf{F} \phi_2$, $\phi_1 \mathsf{W} \phi_2 \equiv \phi_1 \mathsf{T} \phi_2$, and this fact in turn allows us to obtain a notion of trace similar to the one classically presented in LTL.

3. Categorical semantics

In this chapter we provide a categorical presentation of the logic introduced in Section 2, by generalising both its models and semantics through the use of relational presheaves, counterpart $\mathcal{W}$-models, and classical attributes.

3.1. Relational presheaves models

The crucial definition of Kripke frame presented in Definition 2.1 admits a natural generalisation in the categorical setting. Given a category $\mathcal{W}$, its objects $A, B, C, \ldots$ can be considered as worlds or instants of time, and the arrows $f : A \to B$ of the category represent the Kripkean notion of *temporal developments* or *ways of accessibility*. Notice that in the usual definition of Kripke frame the accessibility relation R is a binary relation on the set W of worlds. Two worlds can thus be connected with at most one possible evolution from one world to another. This is an undesirable constraint from the point of applications, where one might be interested in having multiple different ways to evolve to a next world. Categories naturally generalise this by allowing an arbitrary set of morphisms between worlds in the model.

Following this correspondence in the context of category theory, the definition of counterpart model could be represented with the notion of *presheaf* $D : \mathcal{W}^{op} \to \mathbf{Set}$ on the desired category $\mathcal{W}$. The use of the opposite category $\mathcal{W}^{op}$ in the definition of presheaf stems from its traditional use in the setting of categorical logic and hyperdoctrines [25,26].

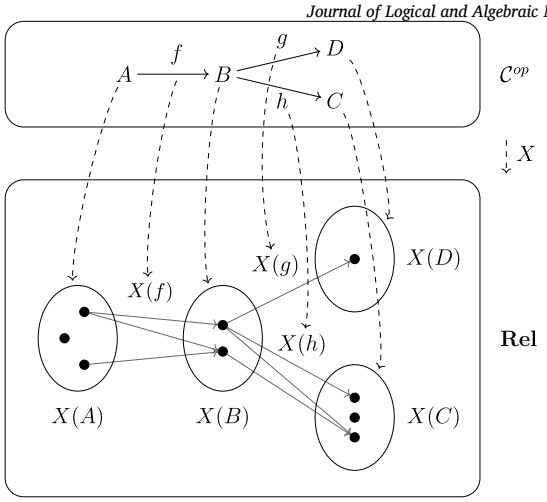


Fig. 6. An example of a relational presheaf X on a category C.

Concretely, a presheaf assigns to each world $\omega \in C$ a set $D(\omega)$ of individuals, and to each time development $f : \omega \to \sigma$ a function $D(f) : D(\sigma) \to D(\omega)$ in the opposite direction between the individuals in the two worlds. From the counterpart perspective, given two elements $a \in D(\omega)$ and $b \in D(\sigma)$, the equality $a = D(f)(b)$ intuitively represents the fact that *b is a future development of a with respect to f*. In other words, *a presheaf represents the categorification of a counterpart model whose counterpart relation is functional*. In practice, this means that each individual in the target world ω is forced to have a counterpart in the previous world σ, thus disallowing the creation of new elements. Considering a standard covariant functor $D : \mathcal{W} \to \mathbf{Set}$ would similarly allow for the creation of elements to be modelled, but not their deallocation, since the morphisms in **Set** are taken to be total functions.

To adequately capture the notion of counterpart model from the categorical perspective, we therefore generalise presheaves to the case of *relations* instead of functions, thus introducing the notion of *relational presheaf*.

Definition 3.1 *(Relational presheaf).* Given a category C, a **relational presheaf** is a functor $D : C^{op} \to \mathbf{Rel}$, where **Rel** is the category of sets and relations.

Given a relational presheaf D and a temporal development $f : \omega \to \sigma$, we can consider the relation $D(f) \subseteq D(\sigma) \times D(\omega)$ as the counterpart relation associated to the evolution step f. In this context, given two elements $a \in D(\omega)$ and $b \in D(\sigma)$ we say that *b is a future development of a with respect to f* whenever $\langle b, a \rangle \in D(f)$.

Example 3.1. We present in Fig. 6 a pictorial example of relational presheaf on a category C.

With the notion of relational presheaf, we can redefine counterpart models in the categorical setting.

Definition 3.2 *(Counterpart $\mathcal{W}$-model).* A **counterpart $\mathcal{W}$-model** is a pair $M = \langle \mathcal{W}, D \rangle$ such that

- $\mathcal{W}$ is a category of worlds;
- $D : \mathcal{W}^{op} \to \mathbf{Rel}$ is a relational presheaf on $\mathcal{W}$.

A crucial difference with classical counterpart models is that counterpart $\mathcal{W}$-models introduce considerably more morphisms than those that might be desirable. In particular, categories are required to always have identity morphisms: this practically means that, for each world, there must be an idle time development remaining in the same world where no entity is either created or destroyed. Similarly, having all compositions in a category means that one can always directly skip to a world if a path of time evolutions can be constructed to reach it.

In order to adequately restrict the models to only a specific set of desirable arrows, the notion of *temporal structure* is introduced.

3.2. Temporal structures

Definition 3.3 *(Temporal structure).* A **temporal structure** T on a category $\mathcal{W}$ is a class of selected morphisms of $\mathcal{W}$.

The intuition behind temporal structures is that they select only the *atomic transitions*, or *indecomposable operations* of $\mathcal{W}$, and are precisely the arrows we consider as relevant in the semantics of our logic. Temporal structures and categories can be bundled up together to form a specific kind of model which we call *temporal counterpart $\mathcal{W}$-model*.

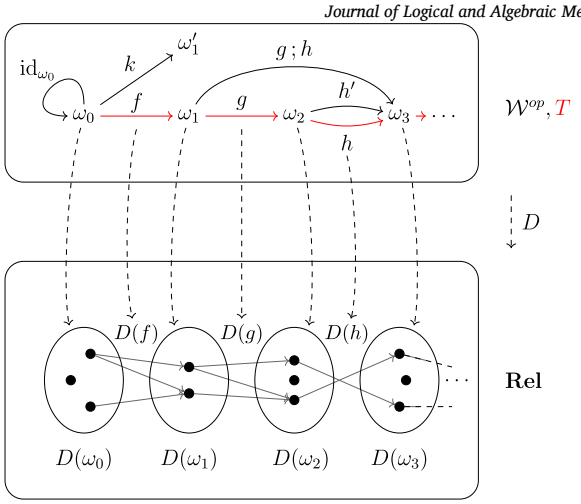


Fig. 7. An example of temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}, D, T \rangle$.

Definition 3.4. A **temporal counterpart $\mathcal{W}$-model** is defined as a tuple $\langle \mathcal{W}, D, T \rangle$, where $\langle \mathcal{W}, D \rangle$ is a counterpart $\mathcal{W}$-model and T is a temporal structure on $\mathcal{W}$.

Example 3.2 *(Temporal counterpart $\mathcal{W}$-model).* We introduce a concrete example of temporal counterpart $\mathcal{W}$-model in Fig. 7.

Temporal counterpart $\mathcal{W}$-models are sufficiently flexible to express and obtain as particular instance the classical models of LTL and CTL, for example retrieving the usual notion of trace described in Definition 2.5, which we employ in Definition 3.10 for the semantics of our logic.

Definition 3.5 *(Paths).* Notationally, given a temporal structure T we denote by $\mathsf{path}(T, \omega)$ the class of possible sequences of arrows $t = (t_0, t_1, t_2, \dots)$ such that $t_i \in T$ and $\mathrm{cod}(t_i) = \mathrm{dom}(t_{i+1})$ for any $i \geq 0$, with the sequence starting with $\mathrm{dom}(t_0) = \omega$. Whenever the path $t \in \mathsf{path}(T, \omega)$ is clear from the context, we indicate with $\omega_i = \mathrm{cod}(t_i)$ the i-th world of the path.

Equipping counterpart $\mathcal{W}$-models with temporal structures allows to formally link classical counterpart models with their categorical version.

Proposition 3.1. *Given a classical counterpart model $\langle W, D, C \rangle$, one can construct a temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}, D, T \rangle$ as follows*

- *$\mathcal{W}$ is the category with $\mathrm{Obj}(\mathcal{W}) := W$ as objects and whose arrows are freely generated (i.e. adding identities and compositions) by introducing a morphism $r : \omega_1 \to \omega_2$ for each relation $R \in C\langle \omega_1, \omega_2 \rangle$;*
- *D is the relational presheaf $D : \mathcal{W}^{op} \to \mathbf{Rel}$ defined as $D(\omega) := d(\omega)$, and assigning to each arrow $r : \omega_1 \to \omega_2$ its generating relation $R \in C\langle \omega_1, \omega_2 \rangle$ with $D(r) := R$ (it is straightforward to verify that this is indeed a functor);*
- *T is the temporal structure identifying as class of morphisms all arrows $r : \omega_1 \to \omega_2$ given by the one-step relations $R \in C\langle \omega_1, \omega_2 \rangle$ of the model.*

Proposition 3.2. *Given a temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}, D, T \rangle$, one can construct a classical counterpart model $\langle W, D, C \rangle$ as follows*

- *$W := \mathrm{Obj}(\mathcal{W})$ is the set of worlds given by the objects of the category;*
- *$d(\omega) := D(\omega)$ is a function assigning to each $\omega \in W$ the action on objects of the relational presheaf D;*
- *C is the function assigning to each tuple $\langle \omega_1, \omega_2 \rangle$ the set of relations $C\langle \omega_1, \omega_2 \rangle := \{ D(r) \mid r : \omega_1 \to \omega_2 \ \wedge \ r \in T \}$, where each morphism r of $\mathcal{W}$ must be selected by the temporal structure T.*

Remark 3.1. We remark how these two constructions are not one the inverse of the other, e.g., going from a temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}, D, T \rangle$ to a classical one loses information about morphisms which are not part of the temporal structure. However, we will recover in Proposition 3.3 how these notions of model are equivalent with respect to the semantics of the logic.

3.3. Presheaf semantics for QLTL

Having introduced the categorical perspective on counterpart models, we now give the definitions required to present the semantics in the categorical setting by means of relational presheaves and classical attributes.

3.3.1. Classical attributes

In Definition 2.11 and Definition 2.13, we associated a meaning to each formula using an inductively defined logical relation. In the context of our categorical semantics, the satisfiability of a formula is instead denoted by assigning to each formula ϕ a *classical attribute* [16,27]: the intuition is that a classical attribute is simply a family of sets indexed on worlds, associating to each world ω the subset of individuals in ω that satisfy a given property.

Definition 3.6 *(Classical attributes).* Let $X : \mathcal{W}^{op} \to \mathbf{Rel}$ be a relational presheaf. A **classical attribute on** X is a family of sets $A := \{A_\omega\}_{\omega \in \mathcal{W}}$ such that $A_\omega \subseteq X(\omega)$ for any $\omega \in \mathcal{W}$. The set of all classical attributes on X is denoted with $\mathcal{T}(X) := \{\{A_\omega\}_{\omega \in \mathcal{W}} \mid A_\omega \subseteq X(\omega)\}$.

Intuitively, the base relational presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$ provides a *common universe* of elements that can be reasoned about, while a classical attribute gives a specific subset of elements in each world $X(\omega)$ for which a property is satisfied. Thus, given a relational presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$ and a classical attribute $A \in \mathcal{T}(X)$, whenever an element $s \in X(\omega)$ is such that $s \in A_\omega$, we can say that in the world ω the individual s satisfies the property A.

For any relational presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$, the set $\mathcal{T}(X)$ of classical attributes has a natural structure of complete boolean algebra with respect to inclusion, where the top element is given by $\top = \{X(\omega)\}_{\omega \in \mathcal{W}}$ and the bottom element by $\bot = \{\emptyset\}_{\omega \in \mathcal{W}}$.

Remark 3.2 *(Classical attributes are presheaves).* A classical attribute can alternatively be considered as a (relational) presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$ in the intuitive way, since it assigns sets to worlds of the category. However, to be consistent with the Agda formalization and for our restricted purpose of providing a categorical semantics for QLTL, we simply consider a classical attribute as a family of sets indexed by the worlds of the category $\mathcal{W}$, and we highlight this difference notationally by using a subscript for classical attributes A_ω and using function application for presheaves $X(\omega)$.

3.3.2. Semantics with classical attributes

Having fixed a relational presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$, we can define the temporal operators of our logic as operators that combine classical attributes and return other classical attributes. This mirrors the intuition that the set of individuals satisfying a composite formula $\phi_1 \mathsf{U} \phi_2$ is obtained by knowing which elements satisfy the subformulae ϕ_1, ϕ_2 composing it.

Definition 3.7 *(Temporal operators on classical attributes).* Let $X : \mathcal{W}^{op} \to \mathbf{Rel}$ be a relational presheaf and T a temporal structure on $\mathcal{W}$. Let $A \in \mathcal{T}(X)$ and $B \in \mathcal{T}(X)$ be two classical attributes on X. Given a world $\omega \in \mathcal{W}$ and an element $s \in X(\omega)$, we define the following **temporal classical attributes**

- $s \in (\mathsf{O}A)_\omega$ if for any arrow $r : \omega \to \sigma$ of T there is an element $z \in X(\sigma)$ such that $\langle z, s\rangle \in X(r)$ and $z \in A_\sigma$;
- $s \in (\mathsf{A}A)_\omega$ if for any arrow $r : \omega \to \sigma$ of T and element $z \in X(\sigma)$ such that $\langle z, s\rangle \in X(r)$ we have that $z \in A_\sigma$;
- $s \in (A \mathsf{U} B)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ there is an $\bar{n} \geq 0$ such that

  1. for any $i < \bar{n}$ there is $z_i \in X(\omega_i)$ such that $\langle z_i, s\rangle \in X(t_{\leq i})$ and $z_i \in A_{\omega_i}$;
  2. there is $z_{\bar{n}} \in X(\omega_{\bar{n}})$ such that $\langle z_{\bar{n}}, s\rangle \in X(t_{\leq \bar{n}})$ and $z_{\bar{n}} \in B_{\omega_{\bar{n}}}$.

- $s \in (A \mathsf{F} B)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ there is an $\bar{n} \geq 0$ such that

  1. for any $i < \bar{n}$ and $z_i \in X(\omega_i)$ such that $\langle z_i, s\rangle \in X(t_{\leq i})$ we have that $z_i \in A_{\omega_i}$;
  2. for any $z_{\bar{n}} \in X(\omega_{\bar{n}})$ such that $\langle z_{\bar{n}}, s\rangle \in X(t_{\leq \bar{n}})$ we have that $z_{\bar{n}} \in B_{\omega_{\bar{n}}}$.

- $s \in (A \mathsf{W} B)_\omega$ if one of the following holds

  – the same conditions for $s \in (A \mathsf{U} B)_\omega$ apply;
  – for any i there is $z_i \in X(\omega_i)$ such that $\langle z_i, s\rangle \in X(t_{\leq i})$ and $z_i \in A_{\omega_i}$.

- $s \in (A \mathsf{T} B)_\omega$ if one of the following holds

  – the same conditions for $s \in (A \mathsf{F} B)_\omega$ apply;
  – for any i and $z_i \in X(\omega_i)$ with $\langle z_i, s\rangle \in X(t_{\leq i})$ we have that $z_i \in A_{\omega_i}$.

Remark 3.3 *(Eventually and always).* Given a relational presheaf $X : \mathcal{W}^{op} \to \mathbf{Rel}$, a temporal structure T on $\mathcal{W}$ and a classical attribute $A \in \mathcal{T}(X)$ on X, it is easy to see that the semantics of our derived temporal operators *always* and *eventually* can be given directly as

- $s \in (\Diamond A)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ there is an $\bar{n} \geq 0$ and an element $z_{\bar{n}} \in X(\omega_{\bar{n}})$ such that $\langle z_{\bar{n}}, s\rangle \in X(t_{\leq \bar{n}})$ and $z_{\bar{n}} \in A_{\omega_{\bar{n}}}$;
- $s \in (\Diamond^* A)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ there is an $\bar{n} \geq 0$ for which any element $z_{\bar{n}} \in X(\omega_{\bar{n}})$ with $\langle z_{\bar{n}}, s\rangle \in X(t_{\leq \bar{n}})$ is such that $z_{\bar{n}} \in A_{\omega_{\bar{n}}}$;
- $s \in (\Box A)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ and $i \geq 0$ there is an element $z_i \in X(\omega_i)$ such that $\langle z_i, s\rangle \in X(t_{\leq i})$ and $z_i \in A_{\omega_i}$;

- $s \in (\Box^* A)_\omega$ if for any $t \in \mathsf{path}(T, \omega)$ and $i \geq 0$ any element $z_i \in X(\omega_i)$ with $\langle z_i, s \rangle \in X(t_{\leq i})$ is such that $z_i \in A_{\omega_i}$.

We have presented this semantics in full generality by quantifying over all possible time development traces $t \in \mathsf{path}(T, \omega)$ provided by the model: notice, however, that the positive normal form equivalences for QLTL and PNF expressed in Section 2.3.2 clearly only hold in the case where the temporal structure T forms a linear order, hence implicitly induces a QLTL trace.

We will use these temporal operators on classical attributes when providing the full semantics of the logic in Definition 3.10.

3.3.3. Semantics of QLTL

We can now show how the semantics of formulae-in-context is provided in our categorical models. Throughout the rest of this section we consider a fixed temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}, D, T \rangle$.

To introduce the categorical semantics of formulae through classical attributes we provide the following intuition: given a formula ϕ with a single free variable, there is an associated classical attribute A_ϕ which assigns to each world ω the set of individuals in ω that satisfy the formula. In fact, *classical attributes are the categorical generalisation of the notion of assignment* presented in Definition 2.9. Similarly, given a formula with n free variables, we consider classical attributes that assign to each world ω the set of n-tuples of individuals in ω such that the formula is satisfied. Notice how this is a subset of elements among all possible tuples given by the n-iterated cartesian product of $D(\omega)$, which is a set, and is therefore in line with the notion of classical attribute. A relatively minor difference with assignments is that variable names in a formula are assumed to refer to indices in the tuple, instead of the proper names given by an assignment, hence a context Γ is properly described as a list of variables.

To make this intuition precise, we introduce products and a terminal object for relational presheaves.

Definition 3.8 *(Product of presheaves).* Given two relational presheaves $X, Y : \mathcal{W}^{op} \to \mathbf{Rel}$, the **product of relational presheaves** $X \times Y : \mathcal{W}^{op} \to \mathbf{Rel}$ is defined as the relational presheaf that uses point-wise the standard set product on worlds. The action on objects is defined as $(X \times Y)(\omega) := X(\omega) \times Y(\omega)$, and for a given morphism $r : \omega_1 \to \omega_2$ we define the relation $(X \times Y)(r) = \{\langle\langle x, y\rangle, \langle x', y'\rangle\rangle \mid \langle x, x'\rangle \in X(r) \wedge \langle y, y'\rangle \in Y(r)\}$.

Definition 3.9 *(Terminal relational presheaf).* The **terminal relational presheaf** $\bot : \mathcal{W}^{op} \to \mathbf{Rel}$ is defined as the relational presheaf $\bot(\omega) = \{*\}$ assigning the singleton set $\{*\}$ to all worlds $\omega \in \mathcal{W}$, and assigning the identity relation on $\{*\}$ to every morphism.

Notation 3.1 *(Relational presheaf of a context).* For any context $\Gamma = [x_1, \ldots, x_n]$ as presented in Definition 2.7, we indicate with $[\![\Gamma]\!]$ the presheaf defined by

$$[\![\Gamma]\!] = \underbrace{D \times \cdots \times D}_{n \text{ times}}$$

where $\times$ denotes the product of relational presheaves given in Definition 3.8. If the context is empty, we define $[\![\emptyset]\!] := \bot$. Given a variable $x \in \Gamma$, we indicate with $\pi_x : [\![\Gamma]\!] \to D$ the corresponding set-based projection on the x-th variable.

As we mentioned, the intuition is that a classical attribute on a product of presheaves identifies *tuples of individuals* that satisfy a given property.

Remark 3.4 *(Classical attribute on a singleton).* Consider the terminal relational presheaf $\bot : \mathcal{W}^{op} \to \mathbf{Rel}$. Then each classical attribute A on $\bot$ has only two possible assignments for any given world $\omega \in \mathcal{W}$ by either having $A(\omega) = \{*\}$ or $A(\omega) = \{\}$, thus indicating that either A holds or does not *for the entire world*. In light of Notation 3.1, classical attributes on $\bot$ correspond with the semantics of closed formulae.

The interpretation of a formula-in-context $[\Gamma]\phi$ is given by a classical attribute $[\![[\Gamma]\phi]\!]$ on the relational presheaf $[\![\Gamma]\!]$ where the formula is defined.

Definition 3.10 *(Satisfiability of a formula).* Given a formula-in-context $[\Gamma]\phi$ and an interpretation $P(\omega) \subseteq D(\omega)$ for each unary predicate $P \in \mathcal{P}$ and world ω, the classical attribute $[\![[\Gamma]\phi]\!]$ on $[\![\Gamma]\!]$ is a function defined by induction on the formula $[\Gamma]\phi$ as follows

- $[\![[\Gamma]\mathsf{false}]\!]_\omega := \emptyset$;
- $[\![[\Gamma]\mathsf{true}]\!]_\omega := [\![\Gamma]\!](\omega)$;
- $[\![[\Gamma]P(x)]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid \pi_x(a) \in P(\omega)\}$;
- $[\![[\Gamma]x = y]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid \pi_x(a) = \pi_y(a)\}$;
- $[\![[\Gamma]\neg\psi]\!]_\omega := [\![\Gamma]\!](\omega) \setminus [\![[\Gamma]\psi]\!]_\omega$;
- $[\![[\Gamma]\phi_1 \vee \phi_2]\!]_\omega := [\![[\Gamma]\phi_1]\!]_\omega \cup [\![[\Gamma]\phi_2]\!]_\omega$;
- $[\![[\Gamma]\phi_1 \wedge \phi_2]\!]_\omega := [\![[\Gamma]\phi_1]\!]_\omega \cap [\![[\Gamma]\phi_2]\!]_\omega$;
- $[\![[\Gamma]\exists x.\phi]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid \exists b \in D(\omega).\langle a, b\rangle \in [\![[\Gamma, x]\phi]\!]_\omega\}$;
- $[\![[\Gamma]\forall x.\phi]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid \forall b \in D(\omega).\langle a, b\rangle \in [\![[\Gamma, x]\phi]\!]_\omega\}$.

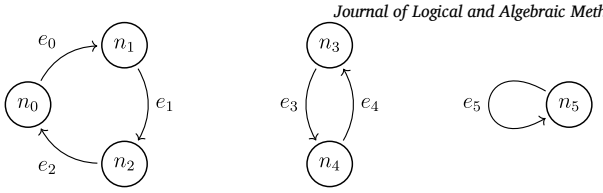


Fig. 8. Examples of three Gr-algebras $\mathbf{G}_0$ (left), $\mathbf{G}_1$ (middle), $\mathbf{G}_2$ (right).

In the case of temporal operators, the classical attribute $[\![[\Gamma]\phi]\!]$ is given directly by the operators defined in Definition 3.7

- $[\![[\Gamma]\mathsf{O}\phi]\!] := \mathsf{O}[\![[\Gamma]\phi]\!]$;
- $[\![[\Gamma]\phi_1 \mathsf{U} \phi_2]\!] := [\![[\Gamma]\phi_1]\!] \mathsf{U} [\![[\Gamma]\phi_2]\!]$;
- $[\![[\Gamma]\phi_1 \mathsf{W} \phi_2]\!] := [\![[\Gamma]\phi_1]\!] \mathsf{W} [\![[\Gamma]\phi_2]\!]$.

Since the definitions of temporal operators are given for any relational presheaf X, we take here a specific case where the base presheaf X is simply given by the product of presheaves $[\![\Gamma]\!]$.

The categorical semantics of QLTL is equivalent to the classical semantics given in Definition 2.11 and Definition 2.13 in the following sense.

Proposition 3.3. *For every classical counterpart model $\langle W, D, C\rangle$ with a trace σ the temporal counterpart $\mathcal{W}$-model $\langle \mathcal{W}', D', T'\rangle$ constructed in Proposition 3.1 is such that for every formula φ, assignment μ and index i*

$$\sigma_i, \mu \vDash_{QLTL}^{\langle W,D,C\rangle} \phi \iff \mu \in [\![[\Gamma]\phi]\!]_{\omega_i}^{\langle W',D',T'\rangle}$$

for each world ω_i in the i-th position of σ.

By following the construction in Proposition 3.2, the above correspondence similarly holds in the case where a temporal W-counterpart model $\langle \mathcal{W}, D, T\rangle$ is given with a path $t \in \mathsf{path}(T, \omega)$ for some world ω.

3.4. Multi-sorted algebra models

In this section we consider a specialisation of counterpart $\mathcal{W}$-models to the case where states are algebras on a signature Σ. This considerably increases the expressiveness of our logic and, by considering for example the signature of graphs, extends it to the case of graph logics. We briefly recall in this section the fundamentals of multi-sorted algebras and signatures.

Definition 3.11 *(Signature).* A **many-sorted signature** Σ is a pair $\langle \Sigma_S, \Sigma_F\rangle$ where

- $\Sigma_S = \{\tau_1, \ldots, \tau_m\}$ is a set of sorts;
- $\Sigma_F = \{f : \tau_1 \times \cdots \times \tau_n \to \tau \mid \tau_i, \tau \in \Sigma_S\}$ is a set of **function symbols** typed over Σ_S^*.

Definition 3.12 *(Algebra).* A **many-sorted algebra** S with signature Σ, i.e. a Σ-algebra, is a pair $\langle S, F\rangle$ where

- $S = \{\mathrm{S}_\tau\}_{\tau \in \Sigma_S}$ is a family of sets for each sort in Σ_S;
- $F := \{f^{\mathrm{A}} : \mathrm{S}_{\tau_1} \times \cdots \times \mathrm{S}_{\tau_n} \to \mathrm{S}_\tau \mid f \in \Sigma_F \wedge f : \tau_1 \times \cdots \times \tau_n \to \tau\}$ is a set of typed functions for every function symbol $f \in \Sigma_F$.

Example 3.3 *(Signature of graphs).* The signature of graphs $\mathsf{Gr} = \langle \mathsf{Gr}_S, \mathsf{Gr}_F\rangle$ is given by

- $\mathsf{Gr}_S = \{\mathsf{N}, \mathsf{E}\}$;
- $\mathsf{Gr}_F = \{s : \mathsf{E} \to \mathsf{N}, t : \mathsf{E} \to \mathsf{N}\}$, representing the source and target functions on edges.

Example 3.4 *(Example of graph).* Concretely, an algebra on the signature of graphs (Gr-algebra) is a directed graph. We present as an example a graphical representation of three Gr-algebras on the signature of graphs in Fig. 8. Similarly, a relational homomorphism of Gr-algebras is exactly a homomorphism of directed graphs where the relation between nodes and edges does not need to be functional.

Definition 3.13 *(Relational homomorphism of algebras).* Given two algebras A and B, a **relational homomorphism of algebras** ρ is a family of relations $\rho := \{\rho_\tau \subseteq \mathrm{A}_\tau \times \mathrm{B}_\tau \mid \tau \in \Sigma_S\}$ typed over Σ_S such that, for every function symbol $f : \tau_1 \times \cdots \times \tau_n \to \tau$ and every list of elements $(a_1, \ldots, a_n) \in \mathrm{A}_{\tau_1} \times \cdots \times \mathrm{A}_{\tau_n}$ and $(b_1, \ldots, b_n) \in \mathrm{B}_{\tau_1} \times \cdots \times \mathrm{B}_{\tau_n}$ we have that

$$(\forall i \in [1..n].\langle a_i, b_i\rangle \in \rho_{\tau_i}) \implies \langle f^{\mathrm{A}}(a_1, \ldots, a_n), f^{\mathrm{B}}(b_1, \ldots, b_n)\rangle \in \rho_\tau.$$

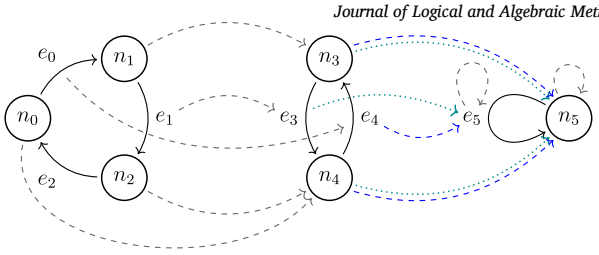


Fig. 9. Graphical representation of an algebraic counterpart $\mathcal{W}$-model.

In our example with the signature of graphs, this amounts to requiring that whenever an edge e has a counterpart e' in the next world, the source nodes (and target nodes) of e and e' must also be in counterpart relation. A relational homomorphism can similarly be considered as a *partial homomorphism* whenever there is at most a single counterpart in the codomain.

To introduce the notion of term on an algebra, we redefine the notion of context since variables are now typed over a set of sorts given by the signature. Finally, we give an inductive definition of terms defined in a typed context.

Definition 3.14 *(Typed context).* Given a denumerable set of variables X, a **typed context** Γ over a signature Σ is a finite subset $[x_1 : \tau_1, \dots, x_n : \tau_n]$ of pairs $(x_i, \tau_i) \in X \times \Sigma_S$ such that $x_1, \dots, x_n$ are distinct.

Definition 3.15 *(Term-in-context).* Let Γ be a typed context over a multi-sorted signature Σ. A **term-in-context** $[\Gamma]\, t : \tau$ is inductively generated by the rules

$$\frac{(x : \tau) \in \Gamma}{[\Gamma]\, x : \tau} \qquad \frac{f : \tau_1 \times \cdots \tau_n \to \tau \in \Sigma_F \qquad [\Gamma]\, t_1 : \tau_1 \quad \cdots \quad [\Gamma]\, t_n : \tau_n}{[\Gamma]\, f(t_1, \dots, t_n) : \tau}$$

where $f : \tau_1 \times \cdots \times t_n \to \tau$ is a function symbol of Σ_F.

We show now how to extend the categorical presentation of counterpart models with relational presheaves to the setting of states as algebras.

3.5. Algebraic counterpart $\mathcal{W}$-models

The intuition to extend our models to the algebraic setting is to consider a relational presheaf for each sort of the algebra: the algebra associated to each world ω is then taken to be the family of sets given by each presheaf on ω. For each function symbol, algebras also provide a notion of *set functions* sending the product of sets to a single set. Similarly, to capture algebra functions categorically we need to send the product of relational presheaves to a single relational presheaf using *set functions*. We capture this idea with the general definition of *relational morphism* between any two relational presheaves.

Definition 3.16 *(Relational morphisms).* A **relational morphism** between two relational presheaves $X, Y : \mathcal{W}^{op} \to \mathbf{Rel}$ is a family of set functions $\eta = \{\eta_\omega : X(\omega) \to Y(\omega)\}_{\omega \in W}$ such that for every morphism $f : A \to B$ of the base category we have that

$$\langle a, b \rangle \in X(f) \implies \langle \eta_A(a), \eta_B(b) \rangle \in Y(f).$$

Definition 3.17 *(Algebraic counterpart $\mathcal{W}$-model).* Let Σ be a many-sorted signature. An **algebraic counterpart $\mathcal{W}$-model** on the signature Σ is a tuple $\langle \mathcal{W}, T, \mathcal{S}, \mathcal{F} \rangle$ such that

- $\mathcal{W}$ is a category of worlds;
- T is a temporal structure on $\mathcal{W}$;
- $\mathcal{S} = \{[\![\tau]\!] : \mathcal{W}^{op} \to \mathbf{Rel}\}_{\tau \in \Sigma_S}$ is a set of relational presheaves on $\mathcal{W}$, assigning a relational presheaf to each sort in Σ_S;
- $\mathcal{F} = \{\mathcal{I}(f) : [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \to [\![\tau]\!]\}_{f \in \Sigma_F}$ is a set of relational morphisms, assigning a relational morphism to each function symbol $f : \tau_1 \times \cdots \times t_n \to \tau$ given in Σ_F by the signature Σ.

Example 3.5 *(Example of algebraic counterpart $\mathcal{W}$-model).* Following Example 3.4, we provide in Fig. 9 our running example of algebraic counterpart $\mathcal{W}$-model on the signature of graphs Gr. We use blue dashed and green dash-dotted lines to distinguish f_1 and f_2, respectively.

We provide the categorical data given by the model by describing explicitly each component. A concrete perspective of our model is shown in Fig. 10.

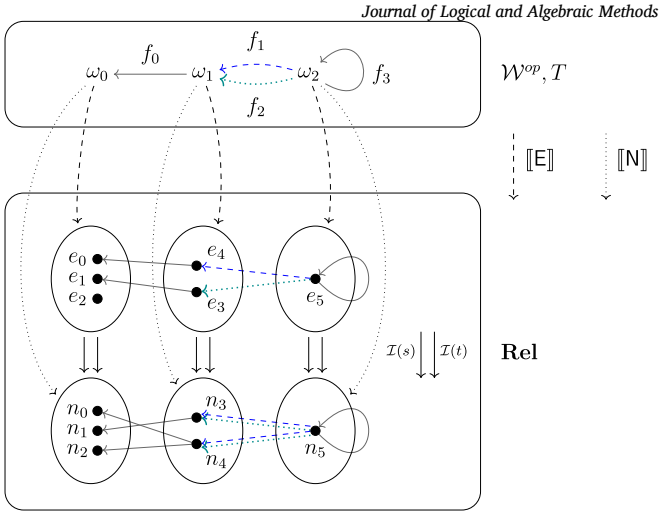
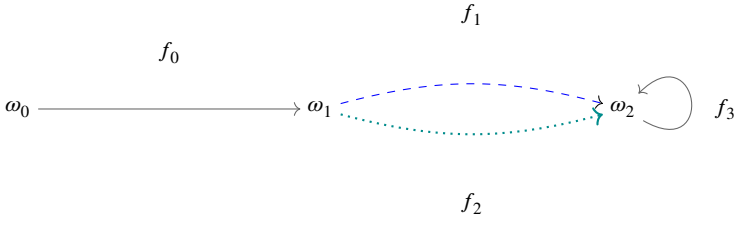


Fig. 10. Explicit categorical data given by Example 3.5.

- The category $\mathcal{W}$ is given as the free category on the following graph



- The temporal structure T is a family of morphisms that selects all one-step arrows of the graph, i.e. $T = \{f_0, f_1, f_2, f_3\}$;
- The relational presheaves associated to the sorts $\{\mathsf{E}, \mathsf{N}\}$ are given by the following data. We first consider the action on objects

$$\begin{array}{ll} [\![\mathsf{E}]\!](\omega_0) = \{e_0, e_1, e_2\}, & [\![\mathsf{N}]\!](\omega_0) = \{n_0, n_1, n_2\}; \\ [\![\mathsf{E}]\!](\omega_1) = \{e_3, e_4\}, & [\![\mathsf{N}]\!](\omega_1) = \{n_3, n_4\}; \\ [\![\mathsf{E}]\!](\omega_2) = \{e_5\}, & [\![\mathsf{N}]\!](\omega_2) = \{n_5\}. \end{array}$$

  By considering the action on morphisms, we define the assignments

$$\begin{array}{ll} [\![\mathsf{E}]\!](f_0) = \{(e_4, e_0), (e_3, e_1)\}, & [\![\mathsf{N}]\!](f_0) = \{(n_4, n_0), (n_3, n_1), (n_4, n_2)\}; \\ [\![\mathsf{E}]\!](f_1) = \{(e_5, e_4)\}, & [\![\mathsf{N}]\!](f_1) = \{(n_5, n_3), (n_5, n_4)\}; \\ [\![\mathsf{E}]\!](f_2) = \{(e_5, e_3)\}, & [\![\mathsf{N}]\!](f_2) = \{(n_5, n_3), (n_5, n_4)\}; \\ [\![\mathsf{E}]\!](f_3) = \{(e_5, e_5)\}, & [\![\mathsf{N}]\!](f_3) = \{(n_5, n_5)\}. \end{array}$$

- The relational morphisms associated to each function symbol $\{s, t\}$ of the signature are given in the intuitive way, and one can easily check that these are indeed relational morphisms

$$\begin{array}{l} \mathcal{I}(s)_{\omega_0} = \{(e_0 \mapsto n_0), (e_1 \mapsto n_1), (e_2 \mapsto n_2)\}, \\ \mathcal{I}(t)_{\omega_0} = \{(e_0 \mapsto n_1), (e_1 \mapsto n_2), (e_2 \mapsto n_0)\}; \\ \mathcal{I}(s)_{\omega_1} = \{(e_3 \mapsto n_3), (e_4 \mapsto n_4)\}, \\ \mathcal{I}(t)_{\omega_1} = \{(e_3 \mapsto n_4), (e_4 \mapsto n_3)\}; \\ \mathcal{I}(s)_{\omega_2} = \{(e_5 \mapsto n_5)\}, \\ \mathcal{I}(t)_{\omega_2} = \{(e_5 \mapsto n_5)\}. \end{array}$$

3.6. Semantics of algebraic QLTL

We can now leverage the algebraic structure of the models to increase the expressiveness of our logic QLTL. We briefly summarise the crucial differences between the non-algebraic and algebraic case with respect to the syntax and semantics of our logic

- formulae are now defined in typed contexts instead of untyped contexts;
- instead of having an atomic formula $x = y$ that models standard equality of individuals in the world, we can directly equate two *terms* $s =_\tau t$ in a world, with the terms s, t both having type $\tau \in \Sigma_S$ in the signature Σ;

- similarly, predicates P_τ are now typed over a generic sort $\tau \in \Sigma_S$ of the signature, and well-typed terms can appear as arguments of predicates;
- existence of individuals $\exists x.\phi$ is typed over a generic sort $\exists_\tau x.\phi$.

In this section we assume to be working with a fixed algebraic counterpart $\mathcal{W}$-model $\langle \mathcal{W}, T, \mathcal{S}, \mathcal{F} \rangle$. We start by generalising Notation 3.1 and similarly provide the interpretation of typed contexts as relational presheaves.

Notation 3.2 *(Typed contexts as relational presheaves).* Given a typed context $\Gamma = [x_1 : \tau_1, \dots, x_n : \tau_n]$, we indicate with $[\![\Gamma]\!]$ the relational presheaf

$$[\![\Gamma]\!] := [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$$

where $\times$ denotes the product of relational presheaves in Definition 3.8.

Definition 3.18 *(Interpretation of a term).* Given a typed context Γ and a term $[\Gamma]\ t : \tau$, we define the **interpretation** $[\![t]\!]$ as the relational morphism given by induction on the structure of the derivation of the term, as following

- if $t = x_i$ with $(x_i, \tau_i) \in \Gamma$, then $[\![t]\!]$ is given by the relational morphism

$$[\![\Gamma]\!] \xrightarrow{\pi_i} [\![\tau]\!]$$

  where π_i is the i-th projection out of the product of relational presheaves;
- if $t = f(t_1, \dots, t_n)$, then $[\![t]\!]$ is given by the composition of relational morphisms

$$[\![\Gamma]\!] \xrightarrow{\langle [\![t_i]\!], \dots, [\![t_n]\!] \rangle} [\![\Gamma']\!] \xrightarrow{\mathcal{I}(f)} [\![\tau]\!]$$

  where $\langle [\![t_i]\!], \dots, [\![t_n]\!] \rangle$ denotes the n-ary product of relational presheaves, intuitively mapping the relational morphisms $[\![t_i]\!]$ to each component of the product.

Finally, we extend the interpretation of QLTL in the algebraic setting by generalising formulae-in-context to typed contexts.

Definition 3.19 *(Algebraic QLTL).* Given a set of denumerable variables $\mathcal{X}$ with $x \in \mathcal{X}$, a family of (unary) predicates $\mathcal{P} := \{P_\tau\}_{\tau \in \Sigma_S}$ indexed by sorts τ, the syntax of **algebraic QLTL** formulae-in-context is given by

$$\psi := \mathsf{true} \mid s =_\tau t \mid P_\tau(s) \qquad \phi := \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists_\tau x.\phi \mid \mathsf{O}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{W} \phi_2$$

where $[\Gamma]\ s : \tau$ and $[\Gamma]\ t : \tau$ are terms defined in the context Γ of the formula with $\tau \in \Sigma_S$. We will omit subscripts whenever the type used in an operator can be inferred from the context.

We now provide only the semantic rules for which their interpretation differs from the non-algebraic case.

Definition 3.20 *(Semantics of algebraic QLTL).* Given the semantic interpretation of QLTL formulae in Definition 3.10, by extending the interpretation to typed predicates $P_\tau(\omega) \subseteq [\![\tau]\!](\omega)$ for each sort τ, predicate symbol $P_\tau \in \mathcal{P}$, and world ω, we can consider the following additional definitions

- $[\![[\Gamma] P_\tau(s)]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid [\![s]\!]_\omega(a) \in P_\tau(\omega)\}$;
- $[\![[\Gamma] s = t]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid [\![s]\!]_\omega(a) = [\![t]\!]_\omega(a)\}$;
- $[\![[\Gamma] \exists_\tau x.\phi]\!]_\omega := \{a \in [\![\Gamma]\!](\omega) \mid \exists b \in [\![\tau]\!](\omega).\langle a, b \rangle \in [\![[\Gamma, x : \tau]\phi]\!]_\omega\}$.

The intuitive semantics for the first rule is that it identifies the set of assignments in the context such that the typed (unary) predicate $P_\tau(s)$ holds for the term s with type τ in the world ω.

Notice how the definitions of temporal operators given in Definition 3.7 to deal with the temporal operators $\mathsf{O}-$, $-\mathsf{U}-$, $-\mathsf{W}-$ and their universally quantified counterparts remain unchanged, since the base relational presheaf X is simply the relational presheaf associated to a typed context $[\![\Gamma]\!](\omega)$.

3.7. Examples

We provide some examples of satisfiability for simple algebraic QLTL formulae on the running example in Fig. 9. Taking for example the formulae

$$\begin{aligned} \mathbf{present}_\tau(x) &:= \exists_\tau y.x =_\tau y, \\ \mathbf{nextStepPreserved}_\tau(x) &:= \mathbf{present}_\tau(x) \wedge \mathsf{O}\mathbf{present}_\tau(x), \\ \mathbf{nextStepDeallocated}_\tau(x) &:= \mathbf{present}_\tau(x) \wedge \neg\mathsf{O}\mathbf{present}_\tau(x), \end{aligned}$$

we obtain the set of edges of graphs $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ surviving at the next step

$$[\![[x : \mathsf{E}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_0} = \{e_0, e_1\}$$
$$[\![[x : \mathsf{E}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_1} = \{\}$$
$$[\![[x : \mathsf{E}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_2} = \{e_5\}$$

Notice that no edge is **nextStepPreserved** in the second case, since the development $[\![\mathsf{E}]\!](f_1)$ deallocates the edge e_4 and, similarly, $[\![\mathsf{E}]\!](f_2)$ deallocates the edge e_3. Following the semantics presented in Definition 3.7, we require that a given property has to hold for *every* time development of length one.

By considering deallocations we have the following

$$[\![[x : \mathsf{E}]\ \mathbf{nextStepDeallocated}(x)]\!]_{w_0} = \{e_2\}$$
$$[\![[x : \mathsf{E}]\ \mathbf{nextStepDeallocated}(x)]\!]_{w_1} = \{\}$$
$$[\![[x : \mathsf{E}]\ \mathbf{nextStepDeallocated}(x)]\!]_{w_2} = \{\}$$

In the second case, we have that again no edge is fully deallocated since it is present in some temporal developments. On nodes we have that

$$[\![[x : \mathsf{N}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_0} = \{n_1, n_2\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_1} = \{n_3\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{nextStepPreserved}(x)]\!]_{w_2} = \{n_5\}$$

We can define formulae that exploit the algebraic structure of worlds and combine them with the temporal operators to consider their evolution in time. For example, we can construct a formula modelling if an edge e is a loop or a node n possesses a loop, the existence of a loop in the current graph, and finally if an edge e will become a loop after at least one step

$$\begin{aligned} \mathbf{loop}(e) &:= s(e) =_{\mathsf{N}} t(e), \\ \mathbf{nodeHasLoop}(n) &:= \exists_{\mathsf{E}} e.s(e) =_{\mathsf{N}} n \wedge \mathbf{loop}(e) \\ \mathbf{hasLoop} &:= \exists_{\mathsf{E}} e.\mathbf{loop}(e) \\ \mathbf{willBecomeLoop}(e) &:= \neg\mathbf{loop}(e) \wedge \Diamond\mathbf{loop}(e) \end{aligned}$$

We can verify that the only node having a loop is n_5

$$[\![[x : \mathsf{N}]\ \mathbf{nodeHasLoop}(x)]\!]_{w_0} = \{\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{nodeHasLoop}(x)]\!]_{w_1} = \{\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{nodeHasLoop}(x)]\!]_{w_2} = \{n_5\}$$

We can also express this by stating that the loop belongs to the entire world

$$[\![\emptyset\ \mathbf{hasLoop}]\!]_{w_0} = \{\}$$
$$[\![\emptyset\ \mathbf{hasLoop}]\!]_{w_1} = \{\}$$
$$[\![\emptyset\ \mathbf{hasLoop}]\!]_{w_2} = \{*\}$$

Since **hasLoop** is a closed formula, the classical attribute only provides binary information either with the empty set or the singleton set, as described in Remark 3.4.

Notice that again there is no node that becomes a loop after some steps, since we require that this is the case for all temporal developments

$$[\![[x : \mathsf{N}]\ \mathbf{willBecomeLoop}(x)]\!]_{w_0} = \{\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{willBecomeLoop}(x)]\!]_{w_1} = \{\}$$
$$[\![[x : \mathsf{N}]\ \mathbf{willBecomeLoop}(x)]\!]_{w_2} = \{\}$$

Finally, we consider whether a node will develop a new loop after some time. Since all nodes have a counterpart in the next world, they all converge to the case of n_5, which already has a loop

$$[\![[x : \mathsf{N}]\ \neg\mathbf{nodeHasLoop}(x) \wedge \Diamond\mathbf{nodeHasLoop}(x)]\!]_{w_0} = \{n_0, n_1, n_2\}$$
$$[\![[x : \mathsf{N}]\ \neg\mathbf{nodeHasLoop}(x) \wedge \Diamond\mathbf{nodeHasLoop}(x)]\!]_{w_1} = \{n_3, n_4\}$$
$$[\![[x : \mathsf{N}]\ \neg\mathbf{nodeHasLoop}(x) \wedge \Diamond\mathbf{nodeHasLoop}(x)]\!]_{w_2} = \{\}$$

3.8. Remarks on second-order extensions

We conclude this section by discussing a possible second-order extension of QLTL and, in particular, its semantics.

First, recall that having a second-order logic allows us to reason about, quantify, and prove properties of *subsets* of elements. Such an expressiveness would be relevant and useful in the formal setting of QLTL, as it would provide us with a formal way to deal with statements like “there exists a set of nodes that are always connected” or “there exists a subset of nodes such that they are all connected and after one step they will be all not connected”. In other words, this would allow us to formally reason about *local* properties of our graphs and their preservation over time.

While it is straightforward to extend the syntax and set-based semantics of QLTL with second-order features, extending our categorical presentation along this line is quite challenging.

In general, the problem of dealing with second (or higher) order features via category theory is that this requires introducing a suitable *power object*, i.e. an object that satisfies the universal properties holding for the powerset in **Set**. The leading example of a category with power objects is that of a topos, e.g. **Set** and the category of presheaves over **Set**. However, in our context the problem of higher-order features is non-trivial because the category relational presheaves (on a given category) rarely have such a topos-like structure, as observed in [28]. This is due to the fact that the category **Rel** is not a topos, but just an *power-allegory* [29]. For the formal definition and the proof that **Rel** is a power-allegory we refer the reader to [29, Prop. 2.414].

Therefore, since we can not employ the formal categorical notion of power object, if we want to properly deal with higher-order features we have to devise a suitable relational presheaf playing a similar role.

We briefly discuss how one can define such a *power-set relational presheaf* $\mathrm{P}(X) : \mathcal{C}^{op} \to \mathbf{Rel}$ of a given presheaf $X : \mathcal{C}^{op} \to \mathbf{Rel}$, and how this can be used to interpreted a second-order extension of QLTL. To this purpose, we employ the known equivalence between relations and Galois connections (or maps) on power-sets [30], i.e. the equivalence between **Rel** and Map(**Pow**), where the latter denotes the category of maps on power-sets.

Recall from [30, Ex. 2] that once we have a relation $R \subseteq A \times B$, we can define a function $\mathscr{P}_R : \mathscr{P}(B) \to \mathscr{P}(A)$ (preserving arbitrary unions) by assigning $\mathscr{P}_R(S) := \{a \in A \mid \exists b \in S \ : \ aRb\}$ to every subset $S \subseteq B$. Moreover, given the equivalence $\mathbf{Rel} \equiv \mathrm{Map}(\mathbf{Pow})$, and using the fact that $\mathbf{Pow} = (\mathbf{Pow}^{op})^{op}$, we can immediately conclude that the assignment $R \mapsto \mathscr{P}_R$ preserves compositions and identities.

Definition 3.21. Let $X : \mathcal{C}^{op} \to \mathbf{Rel}$ be a relational presheaf. The **relational power-set presheaf** $\mathrm{P}(X) : \mathcal{C}^{op} \to \mathbf{Rel}$ is the functor defined as

- for every object $\sigma \in \mathcal{C}$, $\mathrm{P}(X)(\sigma) := \mathscr{P}(X_\sigma)$ is the power-set of X_σ,
- for every arrow $f : \sigma \to \omega$ of $\mathcal{C}$ the relation $\mathrm{P}(X)_f \subseteq \mathrm{P}(X)_\omega \times \mathrm{P}(X)_\sigma$ is defined as $\mathrm{P}(X)_f := \mathscr{P}_{X_f}$.

The relational presheaf $\mathrm{P}(X) : \mathcal{C}^{op} \to \mathbf{Rel}$ is thus an ordinary presheaf over **Set**. Finally, given a relational presheaf $X : \mathcal{C}^{op} \to \mathbf{Rel}$ we define the *epsiloff relational presheaf* $\in_X : \mathcal{C}^{op} \to \mathbf{Rel}$, following the idea and notation in [29].

Definition 3.22. Let $X : \mathcal{C}^{op} \to \mathbf{Rel}$ be a relational presheaf. The **epsiloff relational presheaf** is the functor $\in_X : \mathcal{C}^{op} \to \mathbf{Rel}$ defined as

- for every $\sigma \in \mathcal{C}$, $\in_X(\sigma) := \{(a, A) \in X_\sigma \times \mathrm{P}(X)_\sigma \mid a \in A\}$,
- for every $f : \sigma \to \omega$, $(\in_X)_f$ is the relation given by $\langle (b, B), (a, A) \rangle \in (\in_X)_f$ if $\langle b, a \rangle \in X_f$ and $\mathrm{P}(X)_f(B) = A$ where $B \subseteq X_\omega$ and $A \subseteq X_\sigma$.

These relational presheaves allow us to interpret, for examples, the following second-order extension of QLTL (which is based on the logics considered in [13,14]): the syntax is now equipped with a set of second order (typed) variables $\mathcal{Y}$, a set of (atomic) predicates ψ including the predicates $\varepsilon \in_\tau \chi$ (where ε is a first-order variable and χ is a second-order variable, both of sort τ), and ϕ with second order quantifier $\exists_\tau \chi.\psi$. Following the previous notation, we denote by $[\Gamma, \Delta]$ the contexts where Γ is the context of first-order variables, while Δ represents the second-order context.

In this setting, we can interpret (with respect to a fixed counterpart $\mathcal{W}$-model as in the previous section) a second order context Δ as

$$[\![\Delta]\!] := \mathrm{P}([\![\tau_1]\!]) \times \cdots \times \mathrm{P}([\![\tau_n]\!])$$

and then a context $[\Gamma, \Delta]$ as

$$[\![\Gamma, \Delta]\!] := [\![\Gamma]\!] \times [\![\Delta]\!]$$

The interpretation of the second-order formulae at a given world is as follows

- $[\![[\Gamma, \Delta]\varepsilon \in_\tau \chi]\!]_\omega := \langle \pi_\varepsilon, \pi_\chi \rangle^*_\omega(\in_{[\![\tau]\!]}(\omega))$ where $\langle \pi_\varepsilon, \pi_\chi \rangle : [\![\Gamma, \Delta]\!] \to [\![\varepsilon : \tau, \chi : \tau]\!]$ denotes the opportune projection of relation presheaves;
- $[\![[\Gamma, \Delta]\exists_\tau \chi.\phi]\!]_\omega := \{a \in [\![\Gamma, \Delta]\!](\omega) \mid \exists b \in \mathrm{P}([\![\tau]\!])(\omega).\langle a, b \rangle \in [\![[\Gamma, \chi : \tau, \Delta]\phi]\!]_\omega\}$.

Therefore, even if the category of relational presheaves is not equipped with power objects, we can define suitable relational presheaves allowing us to provide a meaningful interpretation of second-order extension of QLTL.

Again, we stress the fact that this is just a possible solution to the problem of the absence of power objects, and other possible solutions could be adopted. Defining and studying the properties of other powerset-like relational presheaves is an interesting line of investigation for future works.

4. Agda formalisation

In this chapter we present an overview of an additional contribution of this work, a formalisation of the categorical semantics presented in Section 3 using the dependently typed programming language and proof assistant Agda [20]. We introduce here just a part of the formalised codebase, and we focus on the most important structures and definitions by providing the semantics of our logic directly with algebraic counterpart $\mathcal{W}$-models. The full code of the categorical formalisation is available at [31], whereas the code for the set-based formalization for positive normal forms is available at [23].

We assume in this chapter that the reader is familiar with Agda and its notation, but we believe that most of the definitions presented in the following sections can be understood even with little or no familiarity with the language and its syntax. For a complete introduction to Agda we refer to [32].

4.1. Formalisation aspects

Our formalisation work consists in the mechanisation of all the aspects presented so far: we start by defining the notions of counterpart relations and traces of relational morphisms as models of the logic, and provide a representation for (well-typed and well-scoped) syntax for formulae of QLTL and PNF along with their satisfiability semantics. Then, we provide a conversion function from QLTL to PNF along with proofs of correctness and completeness of the procedure; finally, using the defined framework, we prove among other equivalences the relevant expansion laws introduced in Section 2.3.2 for the functional setting. The implementation is general enough to consider the case of algebras over any generic multi-sorted signature. In practice, this means that by specifying a suitably defined signature the class of models (and formulae) considered by the logic can be extended to the case of any graphical formalism that admits an algebraic representation on a multi-sorted signature, such as trees, hypergraphs, and so on.

Moreover, given the constructive interpretation of the formalisation, proving that the correctness and completeness of PNF with respect to QLTL also doubles-down as a concrete procedure that can convert formulae into their positive normal form version, while at the same time providing a proof of the correctness of the conversion.

We describe now how the main components provided by our formalisation can be employed by the user to interact with the proof assistant.

- *Signature definition.* Exploiting the definitions given in our formalisation, the user can write their own algebraic signature that will be used to represent the system of interest as algebras on the signature. For example, by defining the signature of graphs Gr the user can reason on the temporal evolution of graphs, using (relational) graph homomorphisms as counterpart relations between worlds.
- *Formula construction.* After having provided the signature of interest, the user can construct formulae using the full expressiveness of QLTL and can reason on equality of terms constructing according to the signature. This allows the user to express properties that combine both logical quantifiers as well as exploiting the specific structure of the system, possibly composing and reusing previously defined formulae. The infrastructure provided by the formalisation is such that the formulae constructed by the user are inherently checked to be well-scoped and well-typed with respect to the sorts of the signature, e.g. edges and nodes in the case of graphs. The user can freely use negation in formulae, and can (optionally) use the procedures we defined to automatically convert formulae to their PNF, which we have seen in Section 2.3 how can be particularly counterintuitive in the counterpart setting with respect to standard temporal logics.
- *Model definition.* The models of the system at various time instances can be constructed by the user, following again the signature provided. Then, the user specifies a series of symbolic worlds and indicates the possible transitions that can be taken by defining a relation on the worlds. Then, an algebra of the signature must be assigned to each world, and the connection between worlds is translated into a morphism between the algebras provided by the user. The transitions of the models are checked by Agda to preserve the algebraic structure of the worlds considered, thus corresponding to the notion of graph morphisms; this step is relatively straightforward as the automation available in Agda helps with proving the structure-preservation of the maps. Traces between worlds are given using a coinductive definition of traces using sized types [33], allowing for infinite (repeating) traces to be modelled and defined by the user.
- *Validation of formulae in the model.* Using the library the user can prove that a specific model satisfies a given formula; our formalisation automatically simplifies the goal that must be proven to verify the formula, and the user is guided by the proof assistant by automatically constructing the skeleton of the proof term.

4.2. Logics in a constructive proof assistant

In our setting, some crucial usability issues need to be mentioned. Agda is a proof assistant based on the Curry-Howard correspondence, where types are connected with propositions and elements of a type are viewed as its proofs [34]. This paradigm gives an *intuitionistic* interpretation of mathematics, where proving a theorem amounts to being able to show a concrete witness of its validity. In practice, this means that some useful logical principles often used in the setting of temporal logics, such as the law of excluded middle, double negation elimination, or the De Morgan laws to switch connectives and quantifiers whenever negation appears in subformulae, are *not provable* in the system. Consequently, both implementer and user would not be able to prove that for example in our logic QLTL the formula $\neg\neg\phi \implies \phi$ always holds for any choice of models and formula ϕ, since it is not provable in the metalanguage provided by the proof assistant. Thus, without explicitly assuming any other logical principle, the embedding of our temporal logic is

actually restricted to the *intuitionistic* fragment of QLTL. In practice, this is not particularly problematic since classical reasoning can simply be assumed as axiom, and allows the equivalences previously mentioned to be recovered: this however would be undesirable from the user's perspective, as they would have to explicitly use these classical axioms in their proofs.

The negation of logical connectives can also be cumbersome to handle practically. In constructive mathematics, negation is defined as the implication $\neg\phi := \phi \implies \bot$, where $\bot$ indicates the empty type, i.e. falsity. This forces the user to always apply a *reductio-ad-absurdum* technique to prove the validity of any proposition involving negation, first assuming that the formula is valid and then deriving a contradiction. On the other hand, it is often desirable to directly express the negation of a formula using other formulae available in the logic that are easier to manipulate, while still maintaining the full expressivity of the original logic with no additional power. This is a core use case of positive normal forms such as the one presented in Section 2.3.

As an example specific to our logic, consider the case where we try to prove

$$\sigma, \mu \vDash_{\mathsf{QLTL}} \neg(\neg\phi_1 \mathsf{U} \neg\phi_2)$$

In order to prove this in the constructive setting, one needs to show that a contradiction can be derived by assuming there exists an n with the desired properties. It can often be easier to directly work with its negated formula

$$\sigma, \mu \vDash_{\mathsf{QLTL}} \phi_2 \mathsf{W} (\phi_1 \wedge \phi_2)$$

since, by construction, directly provides two cases to be analysed where either ϕ_2 always holds or a concrete n is given where both formulae are satisfied.

Working with the negation of simple connectives such as $\neg(\phi_1 \vee \phi_2)$ can also be problematic, since converting disjunctions in conjunctions uses the classical direction of the De Morgan law which implicitly relies on *double negation elimination*. A similar mechanism happens with first-order quantifiers, such as those used in our logic, as well as in the case of temporal operators.

In order to tackle these usability issues and the treatment of negation in the intuitionistic setting, we take the following approach: the formulae of the logic are expressed in Agda using a full *positive normal form* similar to the one presented in Section 2.3, giving the user complete accessibility over the extended set of quantifiers. This lifts the user from having to deal with negation in subformulas, which can be problematic as we mentioned. On the other hand, the positive normal form is supported by the equiexpressivity results shown in Theorem 2.1, which, to be formally proven in Agda, *do* require classical principles to be postulated. This effectively shifts the burden of dealing (classically) with negation from the user to the implementer, while providing a theoretical guarantee that no expressive power is either gained or lost in the presentation of the logic.

4.2.1. Automation

Embedding a temporal logic in a proof assistant allows the user to exploit the *assistant* aspect of the tool, e.g. by aiding the user in showing (or even proving automatically) that a certain formula in a model is satisfied or not.

In Agda, this automation aspect is limited, especially if compared to proof assistants where automation and the use of tactics is a core aspect of the software environment, such as Coq [35], Lean [36], and Isabelle [37]. The Agda synthesizer Agsy [38] is the main helper tool in Agda implementing a form of automated proof search. Unfortunately, Agsy only provides general-purpose searching procedures and its theorem proving capabilities are nowhere near those of specialised model checking algorithms. Still, the goal-oriented interactivity available in Agda is an invaluable tool in proving theorems step-by-step and *manually* verify formulae in our setting, and the assisted introduction of constructors allows the user to quickly generate the proof structure needed to validate temporal formulae.

4.2.2. Category theory

The `agda-categories` library [21] is a category theory library implemented in Agda, using proof-relevance and setoid-based reasoning as core design choices. In our context of temporal logics, we show that the library provides a solid foundation to use category-theoretical notions even in a practical context that does not necessarily touch upon the theoretical aspects of category theory, but where the categorical perspective is simply used to provide the appropriate data for our models. We will explain the notation used in `agda-categories` whenever required, but we do not provide details ifor the definitions and structures offered by the library.

5. Agda code

In the following sections we present more in detail the main components of our formalisation work. A preliminary part which captures multi-sorted signatures and algebras is left in Appendix A, and we focus here only on the formalisation of the categorical semantics. We start by describing the formalisation of relational presheaves and algebraic counterpart $\mathcal{W}$-models, highlighting the constructions that allow to convert the categorical models into classical ones and viceversa. After showing how we formalise the notion of classical attribute, we introduce the syntax and semantics of our quantified temporal logic, and formalise the examples shown in Section 3.7. The code presented in this chapter has been structured and checked as an Agda literate file and we will omit for simplicity several details, imports, and proofs.

5.1. Relational presheaves

We start by formally capturing the notion of relational presheaf on a given category C, which we provide by parameterising the RelPresheaves module with respect to any category C:

```
module RelPresheaves {co cℓ ce} (C : Category co cℓ ce) where
```

A relational presheaf is simply a presheaf with the category of sets and relations Rels as target:

```
RelPresheaf : Set (sucℓ co ⊔ sucℓ cℓ ⊔ ce)
RelPresheaf = Presheaf C (Rels co cℓ)
```

Given two relational presheaves, a relational morphism between them is given by the pointwise map between their sets η, along with an imply property stating that target elements are related by Y whenever they were related by X at the source. For any given functor X, the functions X.$_0$ and X.$_1$ refer to the action on objects and morphisms, respectively. The notation C [ω_1 , ω_2] refers to the type of morphisms in the category C between ω_1 and ω_2.

```
record RelPresheaf⇒ (X : RelPresheaf) (Y : RelPresheaf)
                    : Set (co ⊔ cℓ) where
  private
    module X = Functor X
    module Y = Functor Y
  open Category C

  field
    η : ∀ {ω} → X.₀ ω → Y.₀ ω
    imply : ∀ {ω₁ ω₂ t s} {f : C [ ω₁ , ω₂ ]}
          → X.₁ f t s
          → Y.₁ f (η t) (η s)
```

Finally, relational presheaves and relational morphisms form a category where identity and composition are defined in the intuitive way. We omit the proofs of associativity and identity required to prove that RelPresheaves is a Category since they follow definitionally.

```
RelPresheaves : Category _ _ _
RelPresheaves = record
  { Obj = RelPresheaf
  ; _⇒_ = RelPresheaf⇒
  ; _≈_ = λ F G → ∀ {ω} x → F.η {ω} x ≡ G.η {ω} x
  ; id =
    record { η = id
           ; imply = id
           }
  ; _∘_ = λ F G →
    record { η = F.η ∘ G.η
           ; imply = F.imply ∘ G.imply
           }
  }
```

5.2. Counterpart models

We now provide the definition of counterpart model from the non-categorical perspective. First, we define a standard counterpart model, as given in Definition 2.2:

```
module CounterpartClassical {ℓ} where
  record LewisCounterpartModel : Set (sucℓ ℓ) where
    field
      W : Set ℓ
      D : W → Set ℓ
      R : Rel W ℓ
      C : ∀ {w₁ w₂}
        → R w₁ w₂
        → REL (D w₁) (D w₂) ℓ
```

The function C is to be interpreted as assigning a relation between two worlds whenever they are themselves connected using the accessibility relation R. Note that this does not impose a restriction on having multiple counterpart relations between worlds since, in Agda, the relation R can be inhabited with multiple witnesses for the same pair of worlds.

Having introduced the notions of algebras and relational homomorphisms, we now extend the standard version of counterpart models to the algebraic case, using algebras as worlds and relational homomorphisms of algebras instead of counterpart relations:

```
record CounterpartModel (Σ : Signature {ℓ}) : Set (sucℓ ℓ) where
  field
    W : Set ℓ
    d : W → Σ-Algebra Σ
    _⇝_ : Rel W ℓ
    f : ∀ {w₁ w₂}
      → w₁ ⇝ w₂
      → Σ-Rel (d w₁) (d w₂)
```

Similarly as with the case of a standard LewisCounterpartModel, this definition of CounterpartModel allows for worlds to be connected through multiple relational homomorphisms.

5.3. Algebraic counterpart $\mathcal{W}$-model

We now provide the definition of an algebraic counterpart $\mathcal{W}$-model.

```
module CounterpartCategorical where
```

First, we need to define the relational presheaf associated to a context, which we again consider here as a Ctx. For simplicity, we omit here the proofs of identity and functoriality of the presheaf defined, and we only show the actions on objects F_0 and F_1:

```
module ContextPresheaf {ℓ} {W : Category ℓ ℓ ℓ} {S : Set ℓ}
                       (⟦_⟧ : S → RelPresheaf W) where

  ⟦_⟧* : ∀ {n} → Vec S n → RelPresheaf W
  ⟦ Γ ⟧* =
    record
      { F₀ = λ ω → mapT (λ Σ → F₀ (⟦ Σ ⟧) ω) Γ
      ; F₁ = λ f → zip (λ {Σ} → F₁ (⟦ Σ ⟧) f)
      }
```

An algebraic counterpart $\mathcal{W}$-model on a given signature is simply the collection of the three fields given in Definition 3.17: a category W, a presheaf ⟦ τ ⟧ on W for each sort τ, and a family I of relational morphisms for each function symbol. Each relational morphism I f has as source the relational presheaf associated to the product of input types of the function, and as target the relational presheaf of the return type:

```
record CounterpartWModel {ℓ} (Σ : Signature {ℓ}) : Set (sucℓ ℓ) where
field
  W : Category ℓ ℓ ℓ
  ⟦_⟧ : ∀ (τ : S) → RelPresheaf W
  I : ∀ (f : F) → RelPresheaf⇒ ⟦ args f ⟧* ⟦ ret f ⟧
```

Given a counterpart $\mathcal{W}$-model, we also obtain the following definitions associated to it. The projection relational morphism, which corresponds exactly with the lookup operation in a context, is given by the following:

```
πᵢ : ∀ {n} {Γ : Ctx n}
  → (i : Fin n)
  → RelPresheaf⇒ (⟦ Γ ⟧*) ⟦ Vec.lookup Γ i ⟧
πᵢ i = record { η = lookup i
              ; imply = lookup-zip i
              }
```

Moreover, we have a relational morphism given by the uniqueness property of the cartesian product of a context. This essentially allows us to apply a Vec of relational presheaves to each sort of a context Γ′. This is given by induction on the structure of the context Γ′ on which the mapping is applied:

```
⟨_⟩* : ∀ {n m} {Γ : Ctx n} {Γ′ : Ctx m}
  → mapT (λ τ → RelPresheaf⇒ (⟦ Γ ⟧*) ⟦ τ ⟧) Γ′
  → RelPresheaf⇒ (⟦ Γ ⟧*) (⟦ Γ′ ⟧*)
⟨_⟩* {Γ′ = []} * =
    record { η = λ _ → *
           ; imply = λ _ → *
           }
⟨_⟩* {Γ′ = _ :: _} (x , xs) =
  let module x = RelPresheaf⇒ x
      module xs = RelPresheaf⇒ (⟨ xs ⟩*)
  in record { η = < x.η , xs.η >
            ; imply = < x.imply , xs.imply >
            }
```

Finally, following Definition 3.18 we have the relational morphism associated to a term, given by induction on the term structure. We use the superscript $^{\prime}$ to indicate that this is the semantic interpretation of terms.

```
⟦_⟧′ : ∀ {i n τ} {Γ : Ctx n}
  → Γ ⊢ τ ⟨ i ⟩
  → RelPresheaf⇒ (⟦ Γ ⟧*) ⟦ τ ⟧
⟦ var i ⟧′ = π₁ i
⟦ fun f x ⟧′ = ! f ∘ ⟨ map ⟦_⟧′ x ⟩*
```

5.4. Temporal structure

The last piece of data for our models is the notion of temporal structure on a category W. A temporal structure is implemented as a (unary) predicate of arrows of the category, thus selecting a specific family of one-step morphisms:

```
record TemporalStructure {co cℓ ce}
                         (W : Category co cℓ ce)
                         : Set (sucℓ (co ⊔ cℓ)) where
  constructor TStructure
  open Category W

  field
    T : ∀ {A B} → Pred (A ⇒ B) cℓ
```

For any given temporal structure T, we define a Path from some object A to be a *coinductive datatype* containing an arrow of the category $arr : A \Rightarrow B$, an implicit proof $arr \in \mathsf{T}$ indicating that arr is selected by the temporal structure T, and a successor path. We again use **sized types** and the Thunk comonad instead of *coinductive records* since we will need to pattern match on Paths and reason by cases on the arrow $A \Rightarrow B$ provided by the path:

```
data Path (A : Obj) (i : Size) : Set (co ⊔ cℓ) where
  _→_ : ∀ {B}
      → (arr : A ⇒ B)
      → {arr ∈ T}
      → Thunk (Path B) i
      → Path A i
```

Given a path we define some self-explanatory accessors on its components:

```
next : ∀ {A i} → Path A i → Obj
next (_→_ {B} _ _) = B

arr : ∀ {A} → (p : Path A ∞) → A ⇒ next p
arr (a → _) = a

tail : ∀ {A i} {j : Size< i} → (p : Path A i) → Path (next p) j
tail (_ → p) = p .force
```

We can take for any i the world given by the trace after i steps, and the arrow compose≤ p i obtained by composing the first i arrows together, noting that this arrow is not necessarily part of the temporal structure.

```
obj : ∀ {A} → Path A ∞ → ℕ → Obj
obj {A} p zero = A
obj p (suc i) = obj (tail p) i

compose≤ : ∀ {A} → (p : Path A ∞) → (n : ℕ) → A ⇒ obj p n
compose≤ p zero = id
compose≤ p (suc i) = compose≤ (tail p) i ∘ arr p
```

5.5. From classical to categorical models

A temporal counterpart $\mathcal{W}$-model is simply the definition of CounterpartWModel endowed with an additional temporal structure T on its category W:

```
record TemporalCounterpartWModel {ℓ} (Σ : Signature {ℓ}) : Set (sucℓ ℓ) where
  field
    M : CounterpartWModel Σ

  open CounterpartWModel M public

  field
    T : TemporalStructure W
```

Given a classical CounterpartModel on algebras, we can obtain the corresponding categorical model by defining a procedure that constructs a TemporalCounterpartWModel following Proposition 3.1. Since our logic QLTL is defined using the categorical presentation with presheaf semantics, this procedure can be exploited to leverage the categorical semantics on classical models, which are easier to describe and do not refer to presheaves:

```
module ClassicalToCategorical {ℓ} {Σ : Signature {ℓ}} where

  open import Relation.Binary.Construct.Composition using (_;_)
  open import Relation.Binary.Construct.Closure.ReflexiveTransitive
        using (Star; ε; _◅_; _◅◅_; _▻▻_)
  open import Categories.Category.Construction.PathCategory
        using (PathCategory)
```

The construction follows precisely the idea described in Proposition 3.1.

```
ClassicalToCategorical : CounterpartModel Σ
                       → TemporalCounterpartWModel Σ
ClassicalToCategorical M =
```

We elucidate the four fields W, ⟦_⟧, I, and T provided by the construction. Given a classical model, the category W is given by the free category (indicated here by PathCategory) induced by the set of worlds W of the model with the accessibility relation _↝_ on it:

```
record
  { M = record
    { W = PathCategory
          record
            { Obj = W
            ; _⇒_ = _↝_
            ; _≈_ = _≡_
            ; equiv = isEquivalence
            }
```

For any sort τ, its corresponding presheaf ⟦ τ ⟧ takes each world ω to the set of objects of the algebra d ω. Similarly, for any function symbol $\mathcal{F}$, the relational morphism I takes each world ω to the corresponding function given by the algebra d ω on the symbol $\mathcal{F}$. In order to show that this is a proper relational morphism, an additional lemma star-imply is shown later in this section using the homomorphism property ρ-homo of relational homomorphisms between algebras:

```
; ⟦_⟧ =
```

```
    λ τ →
      record
        { F₀ = λ ω → S (d ω) τ
        ; F₁ = StarRel
        ; identity    = (λ { refl → lift refl }) , λ { (lift refl) → refl }
        ; homomorphism = λ { {g = g} → star-homomorphism {f = g}}
        ; F-resp-≈ = star-resp-≈*
        }
  ; I =
    λ 𝓕 →
      record
        { η = λ {ω} → F (d ω) 𝓕
        ; imply = λ { {f = f} → star-imply f }
        }
  }
```

The temporal structure T associated to this model is a simple predicate that returns the unit type ⊤ for all morphisms of the free category with length *exactly* one. The empty type ⊥ representing falsity is given in the other cases:

```
  ; T = TStructure
      λ { ε → ⊥
        ; (_ ◁ ε) → ⊤
        ; (_ ◁ (_ ◁ _)) → ⊥
        }
  }
```

The action on arrows of the relational presheaf ⟦ τ ⟧ is given by the function StarRel, which lifts the arrows of the free category to relations between the sets of the algebra, for any sort τ. This lifting is defined by cases on the length of the morphism of the free category:

```
  where
  StarRel : ∀ {τ A B}
  → Star _⇝_ B A
  → REL (S (d A) τ) (S (d B) τ) ℓ
  StarRel ε = _≡_
  StarRel (B⇝C ◁ C⇝*A) = StarRel C⇝*A ; flip (ρ (f B⇝C))
```

where the base case is the identity relation _≡_ and composition of relations _;_ is applied in the inductive case. The use of flip is dictated by the fact that presheaves are functors in the opposite category W^{op}, thus the relation given by the counterpart model needs to be inverted before composing it.

We briefly recap here the obligations that must be proved for the previous construction, omitting their proofs.

```
  star-homomorphism : ∀ {τ X Y Z} {g : Star _⇝_ Y X} {f : Star _⇝_ Z Y}
  → StarRel {τ} (g ▷▷ f) ≈ StarRel {τ} f ∘ StarRel {τ} g

  star-imply : ∀ {𝓕 σ τ t s} f
  → zip (StarRel f) t s
  → StarRel f (F (d τ) 𝓕 t) (F (d σ) 𝓕 s)

  star-resp-≈* : ∀ {τ} {A B} {f g : Star _⇝_ B A}
  → f ≈* g
  → Rels ℓ ℓ [ StarRel {τ} f ≈ StarRel {τ} g ]
```

In these last lemmas the function _▷▷_ and the relation _≈*_ indicate composition and morphism equality in the PathCategory, respectively.

5.6. Classical attributes

In order to define satisfiability, we introduce the notion of ClassicalAttribute. A classical attribute on a relational presheaf X is defined as a (unary) predicate which identifies a subset of X in ω, for each of the worlds ω:

```
module ClassicalAttributes {co cℓ ce} (W : Category co cℓ ce)
                                      (T : TemporalStructure W) where


ClassicalAttribute : RelPresheaf W → Set (sucℓ (co ⊔ cℓ ⊔ ce))
ClassicalAttribute X = ∀ {ω} → Pred (X.₀ ω) _
  where module X = Functor X
```

The action of temporal operators on classical attributes for a given relational presheaf X is defined according to Definition 3.7. We first provide some shorthands to capture the notion of existential and universal quantification of counterparts with a certain property A after i steps:

```
module _ (X : RelPresheaf W) where
  private module X = Functor X
  – Shorthand for:
  – "There exists a counterpart for s in the
  – path p after i steps which satisfies A"
  at∃ : ∀ {ω} → Path ω ∞ → X.₀ ω → ClassicalAttribute X → ℕ → Set _
  at∃ p s A i = ∃[ z ] X.₁ (compose≤ p i) z s × z ∈ A

  – Shorthand for:
  – "All counterparts of s in the path p
  – after i steps satisfy A"
  at∀ : ∀ {ω} → Path ω ∞ → X.₀ ω → ClassicalAttribute X → ℕ → Set _
  at∀ p s A i = ∀ z → X.₁ (compose≤ p i) z s → z ∈ A
```

The one-step classical attributes for the *next* Oϕ and *next-forall* Aϕ operators are defined in the intuitive way. Notice how we again require as implicit argument a proof $\rho \in$ T that the morphisms considered by the two operators are part of the temporal structure:

```
XO : ClassicalAttribute X → ClassicalAttribute X
XO A s = ∀ {σ}
         → (ρ : _ ⇒ σ)
         → {ρ ∈ T}
         → ∃[ z ] X.₁ ρ z s × s ∈ A

XA : ClassicalAttribute X → ClassicalAttribute X
XA A s = ∀ {σ}
         → (ρ : _ ⇒ σ)
         → {ρ ∈ T}
         → ∀ z → X.₁ ρ z s → s ∈ A
```

We use a set of standard predicates inspired by LTL in order to make subsequent definitions more readable:

```
– A holds for all i strictly before n steps
_before_ : ∀ {ℓ} (A : Pred ℕ ℓ) → Pred ℕ ℓ
A before n = ∀ i → i < n → i ∈ A

– A holds until B is satisfied
_until_ : ∀ {ℓ} (A B : Pred ℕ ℓ) → Set ℓ
A until B = ∃[ n ] (A before n × n ∈ B)

– A is always satisfied at each step
always : ∀ {ℓ} (A : Pred ℕ ℓ) → Set ℓ
always A = ∀ i → i ∈ A

– Either until or always hold
_weakUntil_ : ∀ {ℓ} (A B : Pred ℕ ℓ) → Set ℓ
A weakUntil B = A until B ⊎ always A
```

Finally, we define the classical attributes associated to each operator by combining the previous shorthands to provide any possibile operator. The predicates at∃ $p\ s\ A$ and at∃ $p\ s\ A$ are curried over the number of steps i, and are thus viewed as predicates on integers ℕ:

```
XU : ClassicalAttribute X → ClassicalAttribute X → ClassicalAttribute X
XU A B {ω} s = ∀ (p : Path ω ∞) → (at∃ p s A) until (at∃ p s B)

XF : ClassicalAttribute X → ClassicalAttribute X → ClassicalAttribute X
XF A B {ω} s = ∀ (p : Path ω ∞) → (at∀ p s A) until (at∀ p s B)

XW : ClassicalAttribute X → ClassicalAttribute X → ClassicalAttribute X
XW A B {ω} s = ∀ (p : Path ω ∞) → (at∃ p s A) weakUntil (at∃ p s B)

XT : ClassicalAttribute X → ClassicalAttribute X → ClassicalAttribute X
XT A B {ω} s = ∀ (p : Path ω ∞) → (at∀ p s A) weakUntil (at∀ p s B)
```

5.7. Syntax and semantics of QLTL

We now introduce the definition of QLTL formulae, which are intrinsically well-scoped and well-typed with respect to the algebra signature. Following the positive normal form given in Section 2.3 and the issues discussed in Section 4.1, we present the syntax of algebraic QLTL by explicitly providing the entire set of operators as well as negation:

```
module QLTL {ℓ} {Σ : Signature {ℓ}}
                (M : TemporalCounterpartWModel Σ) where
```

The type of QLTL formulae QLTL carries the context Γ in which the formula is defined. We start with the standard cases of formulae with constants, simple connectives and temporal operators:

```
data QLTL {n} (Γ : Ctx n) : Set ℓ where
  true  : QLTL Γ
  false : QLTL Γ
  !_    : QLTL Γ → QLTL Γ
  _∧_   : QLTL Γ → QLTL Γ → QLTL Γ
  _∨_   : QLTL Γ → QLTL Γ → QLTL Γ
  O_    : QLTL Γ → QLTL Γ
  A_    : QLTL Γ → QLTL Γ
  _F_   : QLTL Γ → QLTL Γ → QLTL Γ
  _U_   : QLTL Γ → QLTL Γ → QLTL Γ
  _W_   : QLTL Γ → QLTL Γ → QLTL Γ
  _T_   : QLTL Γ → QLTL Γ → QLTL Γ
```

Existential and universal quantification state explicitly the sort τ on which they quantify on. The context of the inner formula is then extended with a new free variable with type τ, which allows the terms in the sub-formula to refer to it:

```
∃<_>_ : (τ : S)
      → QLTL (τ :: Γ)
      → QLTL Γ
∀<_>_ : (τ : S)
      → QLTL (τ :: Γ)
      → QLTL Γ
```

Finally, the elementary formulae for equality of terms considers the two terms in the context of the formula:

```
_≡ᶠ_ : ∀ {i τ}
     → Γ ⊢ τ ⟨ i ⟩
     → Γ ⊢ τ ⟨ i ⟩
     → QLTL Γ
_≢ᶠ_ : ∀ {i τ}
     → Γ ⊢ τ ⟨ i ⟩
     → Γ ⊢ τ ⟨ i ⟩
     → QLTL Γ
```

We can define the usual syntactic sugar for derived temporal operators:

```
◇_ : ∀ {n} {Γ : Ctx n} → QLTL Γ → QLTL Γ
```

```
◇ ϕ = true U ϕ

□_ : ∀ {n} {Γ : Ctx n} → QLTL Γ → QLTL Γ
□ ϕ = ϕ W false

◇*_ : ∀ {n} {Γ : Ctx n} → QLTL Γ → QLTL Γ
◇* ϕ = true F ϕ

□*_ : ∀ {n} {Γ : Ctx n} → QLTL Γ → QLTL Γ
□* ϕ = ϕ T false
```

The semantics of QLTL formulae is simply a function $\langle \_ \rangle$ that assigns a predicate to formulae in each world. Each predicate $\langle\ \phi\ \rangle$ is defined to be true for a given tuple of elements a in a world ω whenever the formula ϕ is satisfied by that assignment of elements a. This definition corresponds exactly to the notion of classical attribute, with the latter being considered on the relational presheaf of the underlying context $[\![\ \Gamma\ ]\!]^*$.

```
⟨_⟩ : ∀ {n} {Γ : Ctx n} → QLTL Γ → ClassicalAttribute ([[ Γ ]]*)
⟨ true        ⟩ a = ⊤
⟨ false       ⟩ a = ⊥
⟨ ! ϕ         ⟩ a = ¬ ⟨ ϕ ⟩ a
⟨ ϕ₁ ∧ ϕ₂     ⟩ a = ⟨ ϕ₁ ⟩ a × ⟨ ϕ₂ ⟩ a
⟨ ϕ₁ ∨ ϕ₂     ⟩ a = ⟨ ϕ₁ ⟩ a ⊎ ⟨ ϕ₂ ⟩ a
⟨ ∃< τ > ϕ    ⟩ a = ∃[ b ] ⟨ ϕ ⟩ (b , a)
⟨ ∀< τ > ϕ    ⟩ a = ∀ b → ⟨ ϕ ⟩ (b , a)
⟨ t₁ ≡ᵗ t₂    ⟩ a = η ([[ t₁ ]]ᵗ) a ≡ η ([[ t₂ ]]ᵗ) a
⟨ t₁ ≢ᵗ t₂    ⟩ a = η ([[ t₁ ]]ᵗ) a ≢ η ([[ t₂ ]]ᵗ) a
⟨ O ϕ         ⟩ = XO ([[ _ ]]*) ⟨ ϕ ⟩
⟨ A ϕ         ⟩ = XA ([[ _ ]]*) ⟨ ϕ ⟩
⟨ ϕ₁ U ϕ₂     ⟩ = XU ([[ _ ]]*) ⟨ ϕ₁ ⟩ ⟨ ϕ₂ ⟩
⟨ ϕ₁ F ϕ₂     ⟩ = XF ([[ _ ]]*) ⟨ ϕ₁ ⟩ ⟨ ϕ₂ ⟩
⟨ ϕ₁ W ϕ₂     ⟩ = XW ([[ _ ]]*) ⟨ ϕ₁ ⟩ ⟨ ϕ₂ ⟩
⟨ ϕ₁ T ϕ₂     ⟩ = XT ([[ _ ]]*) ⟨ ϕ₁ ⟩ ⟨ ϕ₂ ⟩
```

6. Conclusion

We have shown how a set-based semantics and a categorical semantics for a first-order linear temporal logic can be presented in the counterpart setting. We have investigated some results on the positive normal forms of this logic in the case of relations and partial functions, and argued for their usefulness both in practice and in the case of constructive proof assistants. Finally, we saw how its models can be naturally extended to the algebraic setting, and how the notions and the categorical models presented in the previous chapters can be formalised and practically experimented with in a proof assistant based on dependent type theory such as Agda. We have investigated some results on the positive normal forms of this logic in the case of relations and partial functions, and argued for their usefulness both in practice and in the case of constructive proof assistants.

6.1. Related work

Up to the early 2010s, a series of papers argued for the use of quantified logics for expressing properties of graphs and of graph evolutions. Our models are inspired by the counterpart-based logics explored in the context of a μ-calculus with fixed points in [13], and we refer there for an overview of and a comparison with the by-then current proposals, all favouring an approach based on universal domains. Missing there is [39] and follow-ups such as [40,41], which illustrate one of the relevant tools developed in the graph community, GROOVE. To some extent, the present article and its companion [14], which introduces the categorical semantics of QTL, are summarising a previous thread of research concerning counterpart models, including its implementation. Indeed, the categorical semantics for counterpart models seems of interest in itself, as witnessed by the works surveyed in [14].

The formalisation of temporal logics in (constructive) proof assistants has a long history, see e.g. [42–44]. A practical application and comparison with modern model checkers is in [45], and a fully verified one for LTL is implemented in the Isabelle theorem prover. In [46], a verified proof-search program fro CTL is formalised in Agda, together with a toolbox to implement well-typed proof-searching procedures; a similar embedding of constructive LTL in Agda is provided in [47] for the verification of functional reactive programs. Our proof-of-concept implementation of QLTL witnesses the possibility to move towards the formalisation of quantified temporal logics for proof assistants, an issue sparsely tackled in the literature.

6.1.1. Comparison with graph computation formalisms

Section 3.4 shows how the counterpart paradigm allows for reasoning on the evolution of any formalism that can be presented by a multi-sorted signature. As exemplified in Fig. 9, a case of particular interest is the signature of (directed) graphs, which allows our proposal to be compared with formalisms that can express properties on graph topologies and their evolution.

The idea behind these graph computation formalisms (GCFs) is to use graph-specific definitions where syntactical statements on nodes, edges, sources and targets of edges, and their equalities are first-class citizens. The field has been quite active in the last decade, with a series of papers advocating quantified temporal logics for the specification of GCFs properties. We offer here a short review of some of the most recent proposals, focussing on the dichotomy between the universal domains and the counterpart-based approaches.

Graph programs/flow graphs. The use of monadic-second order logics to prove properties of graph-based programming languages has been advocated in [48,49], where the emphasis is placed on distilling post-conditions formulae from a graph transformation rule and a precondition formula. A more abstract meta-model for run-time verification is proposed in [50,51], where a control flow graph can be instantiated to concrete models and the properties are given by first-order formulae. Despite the differences, in both cases the resulting analysis is akin to the adoption of a universal domain approach.

Metric logics, I. The use of traces and first-order specifications is a key ingredient of runtime verification. A relevant proposal is the use of metric first-order temporal logic (MFOTL) [52,53], investigated with respect to the expressiveness of suitable fragments in [54] or to duality results akin to our PNF in [55]. These logics allow to reason on the individual components of states, using (arbitrary) sets of relations as models, which allows for different kinds of graphs to be encoded. The core difference with our line of work is that, contrary to standard models of MFOTL, we allow for variable domains in the temporal structure and for nodes to be created and destroyed.

Metric logics, II. A graph-oriented approach to MFOTL is given by Metric Temporal Graph Logic (MTGL) [56,57], which allows to model properties on the structures and the attributes of the state and has been used in the context of formal testing [58]. Here traces are pairs of injective spans representing a rule, and are equivalent to our partial graph morphisms. The syntax is tailored over such rules, so that ϕ_G refers to a formula over a given graph G, and a one step $\exists(f, \phi_H)$ is indexed over a mono $f : G \to H$, roughly representing the partial morphism induced by a rule. Identity and preservation/deletion of elements seem to be left implicit, and the exploration of the connection with counterpart-based QLTL is among our future endeavours.

6.2. Future work

We identify a variety of possible expansions for our work.

Second-order. Our theoretical presentation and formalisation work focuses on the first-order aspects of QLTL. The semantics in [13,14] allows also for the quantification over sets of elements. This is impractical in Agda due to the typical formalisation of subsets as predicates, which would be cumbersome to present in concrete examples, e.g. when expressing universal quantification and extensional equality over subsets of elements. A possible extension could be to investigate practical encodings and possible automation techniques to introduce second-order quantification for counterpart-based temporal logics.

CTL and other logics. The quantified temporal logics presented here focus on providing a restricted yet sufficiently powerful set of operators and structures. These logics could be extended to more expressive constructs and models, such as the case of CTL [7], by considering branching models and building more complex temporal structures on the notion of category. We believe that working along these lines would be a straightforward task, which might however cause a combinatorial explosion in the case of possible temporal operators required to obtained a positive normal form of the logic.

Automation and solvers. We highlighted how the proofs required to validate temporal formulae need to be provided manually by the user. Considerable amount of effort has been spent in interfacing proof assistants with external solvers and checkers to both reuse existing work and algorithms and to provide more efficient alternatives to the automation given by proof assistants. The traditional way of employing proof automation is through the use of *internal* and *external* solvers: the first technique uses the reflection capabilities of Agda to allow a (verified) solver and proof-searching procedure to be written in Agda itself, in the spirit of [46,45,59]. The second mechanism consists in writing bindings to external programs, such as external model checkers or SMT and SAT solvers, so that proving the formula or providing a counterexample is offloaded to a more efficient and specialised program. A possible extension of this work would be the implementation of either of these mechanisms to the setting of counterpart semantics.

Category theory. The category theoretical notions formalised in our work constitute a small part of the mechanisation, with categories and relational presheaves being mainly used as data instead of proper structures on which theorems can be stated and shown. Recall that the perspective given by categorical logic is to present the notion of syntax in terms of indexed categories and models as morphisms between them: a future expansion of this work could be to also formalise our notions of models and assignments in terms of morphisms between suitable indexed categories [26].

Finite traces. A current trend in artificial intelligence is the study of temporal formulas over *finite* traces [60], due to applications in planning and reinforcement learning. Our models seem to be well-suited to tackle such a development, since each finite trace can be thought of as an infinite one terminating with a cycle in an empty graph, thus inheriting all the issues we highlighted about positive normal forms for our logic.

CRediT authorship contribution statement

Fabio Gadducci: Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Andrea Laretto:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Davide Trotta:** Writing – review & editing, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Signatures and algebras

We describe how we captured in Agda the notion of signature, algebra, and term. To give the definition of signature of an algebra, we first package up the signature of a function, where $\mathcal{S}$ indicates a generic set of sorts:

```
module SortedAlgebra {ℓ} where

record FunctionSignature (𝒮 : Set ℓ) : Set ℓ where
  constructor _↦_
  field
    {arity} : ℕ
    τ*  : Vec 𝒮 arity
    τ   : 𝒮
```

An algebra signature gives a set of sorts $\mathcal{S}$, a set of function symbols $\mathcal{F}$, and a function sign$\mathcal{F}$ that associates a function signature to each symbol in $\mathcal{F}$:

```
record Signature : Set (suc ℓ ℓ) where
  field
    𝒮 : Set ℓ
    ℱ : Set ℓ
    signℱ : ℱ → FunctionSignature 𝒮

  args = τ* ∘ signℱ
  ret = τ ∘ signℱ
```

An algebra for a signature Σ amounts to providing a function S from symbols to sets and a function F from function symbols to actual functions with type given by the signature:

```
record Σ-Algebra (Σ : Signature) : Set (suc ℓ ℓ) where
  field
    𝒮 : 𝒮 → Set ℓ

  argType : ℱ → Set ℓ
  argType f = mapT S (args f)


  retType : ℱ → Set ℓ
  retType f = S (ret f)

  field
    F : ∀ (f : ℱ) → argType f → retType f
```

Notice that args only gives us a Vec of *symbols* with type $\mathcal{S}$. To consider the argument type of a concrete function given by the algebra, we need to apply the action S on each symbol of the signature, and then consider the cartesian product of the concrete sets obtained.

This is exactly the use of mapT, and we refer to lemmas and properties of types obtained this way as the module VecT. The unit type ⊤ and its single element * is used in the case of the empty vector, and in the inductive case we combine the set $f\,x$ given by the function using the cartesian product ×

```
mapT : (A → Set ℓ) → Vec A n → Set ℓ
mapT f [] = ⊤
mapT f (x :: v) = f x × mapT f v
```

Given some (possibly heterogeneous) relation R, we can relate two Vecs obtained with mapT if they are point-wise related with R

```
zip : ∀ {v : Vec A n} {f g : A → Set ℓ′}
  → (∀ {x} → f x → g x → Set ℓ)
  → mapT f v → mapT g v → Set ℓ
zip {v = []} R * * = ⊤
zip {v = _ :: _} R (x , xs) (y , ys) = R x y × zip R xs ys
```

We can define the type Σ-Rel of relational homomorphisms between algebras given in Definition 3.13. To specify the homomorphism property ρ-homo, we use zip to relate point-wise the function arguments given by the two algebras

```
record Σ-Rel {Σ} (A : Σ-Algebra Σ) (B : Σ-Algebra Σ) : Set (sucℓ ℓ) where
  open Signature Σ
  private
    module A = Σ-Algebra A
    module B = Σ-Algebra B


  field
    ρ : ∀ {τ} → REL (A.S τ) (B.S τ) ℓ
    ρ-homo :
      ∀ (f : 𝓕)
    → {as : A.argType f}
    → {bs : B.argType f}
    → zip ρ as bs
    → ρ (A.F f as) (B.F f bs)
```

A.1. Terms

Terms on a given signature are well-typed with respect to the algebra and use a well-scoped representation, with variables being indices in a context

```
module Terms {ℓ} (Σ : Signature {ℓ}) where
```

We define a context simply as a Vec of sort symbols $\mathcal{S}$ with a known length

```
Ctx : ℕ → Set ℓ
Ctx = Vec 𝒮
```

The type of terms-in-context _⊢_⟨_⟩ is given inductively and parameterised with both an underlying context Γ and with the type of the term being defined. Variables are implemented as de Bruijn indices, where a variable term contains an index pointing to its type in the context. In the var case, the type of the entire term is given using vector lookup to retrieve the type provided by the context. In the case of functions fun, the type of the term corresponds with the return type given by the function symbol. The use of **sized types** and the type Size is necessary in Agda to ensure that recursion on terms is terminating, but it is not essential to the formalisation of our temporal logics

```
data _⊢_⟨_⟩ {n} Γ : 𝒮 → Size → Set ℓ where
  var : (i : Fin n)
    → Γ ⊢ Vec.lookup Γ i ⟨ ∞ ⟩
```

```
  fun : ∀ {s}
    → (f : F)
    → mapT (Γ ⊢_⟨ s ⟩) (args f)
    → Γ ⊢ ret f ⟨ ↑ s ⟩
```

A substitution from a context Γ to a context Δ amounts to being able to derive a new term $t : \Delta \vdash \tau$ for each sort $\tau \in \Gamma$

```
Subst : ∀ {n m} → Ctx n → Ctx m → Set ℓ
Subst Γ Δ = ∀ i → Δ ⊢ Vec.lookup Γ i ⟨ ∞ ⟩
```

Substitutions can be applied to terms, and this consists in reframing a term into a different context:

```
sub : ∀ {n m} {Γ : Ctx n} {Δ : Ctx m}
    → Subst Γ Δ
    → (∀ {s A} → Γ ⊢ A ⟨ s ⟩ → Δ ⊢ A ⟨ s ⟩)
sub σ (var x) = σ x
sub σ (fun f x) = fun f (map (sub σ) x)
```

The identity substitution is given by replacing each variable with a term consisting of the same variable, and substitutions can be suitably composed:

```
id : ∀ {n} {Γ : Ctx n} → Subst Γ Γ
id i = var i

_∘_ : ∀ {n m o} {A : Ctx n} {B : Ctx m} {C : Ctx o}
    → Subst B C → Subst A B → Subst A C
(f ∘ g) i = sub f (g i)
```

References

[1] C. Baier, J. Katoen, Principles of Model Checking, MIT Press, 2008.
[2] A. Pnueli, The temporal logic of programs, in: FOCS 1977, IEEE Computer Society, 1977, pp. 46–57.
[3] B. Courcelle, The monadic second-order logic of graphs. I. Recognizable sets of finite graphs, Inf. Comput. 85 (1) (1990) 12–75.
[4] A. Dawar, P. Gardner, G. Ghelli, Expressiveness and complexity of graph logic, Inf. Comput. 205 (3) (2007) 263–310.
[5] P. Baldan, A. Corradini, B. König, A. Lluch-Lafuente, A temporal graph logic for verification of graph transformation systems, in: J.L. Fiadeiro, P. Schobbens (Eds.), WADT 2006, in: LNCS, vol. 4409, Springer, 2006, pp. 1–20.
[6] H. Kastenberg, A. Rensink, Model checking dynamic states in GROOVE, in: A. Valmari (Ed.), SPIN 2006, in: LNCS, vol. 3925, Springer, 2006, pp. 299–305.
[7] E.A. Emerson, Temporal and modal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier and MIT Press, 1990, pp. 995–1072.
[8] E. Franconi, D. Toman, Fixpoint extensions of temporal description logics, in: D. Calvanese, G. De Giacomo, E. Franconi (Eds.), DL 2003, in: CEUR Workshop Proceedings, vol. 81, 2003.
[9] I.M. Hodkinson, F. Wolter, M. Zakharyaschev, Monodic fragments of first-order temporal logics: 2000-2001 A.D, in: R. Nieuwenhuis, A. Voronkov (Eds.), LPAR 2001, in: LNCS, vol. 2250, Springer, 2001, pp. 1–23.
[10] A. Hazen, Counterpart-theoretic semantics for modal logic, J. Philos. 76 (6) (1979) 319–338.
[11] F. Belardinelli, Quantified modal logic and the ontology of physical objects, Ph.D. thesis, Scuola Normale Superiore of Pisa, 2004–2005.
[12] D.K. Lewis, Counterpart theory and quantified modal logic, J. Philos. 65 (5) (1968) 113–126.
[13] F. Gadducci, A. Lluch-Lafuente, A. Vandin, Counterpart semantics for a second-order μ-calculus, Fundam. Inform. 118 (1–2) (2012) 177–205.
[14] F. Gadducci, D. Trotta, A presheaf semantics for quantified temporal logics, in: A. Madeira, M.A. Martins (Eds.), WADT 2022, in: LNCS, vol. 13710, Springer, 2023, pp. 81–99.
[15] S. Ghilardi, G. Meloni, Modal and tense predicate logic: models in presheaves and categorical conceptualization, in: F. Borceux (Ed.), Categorical Algebra and Its Applications, in: LNM, vol. 1348, Springer, 1988, pp. 130–142.
[16] S. Ghilardi, G. Meloni, Relational and partial variable sets and basic predicate logic, J. Symb. Log. 61 (3) (1996) 843–872.
[17] S. Huang, R. Cleaveland, A tableau construction for finite linear-time temporal logic, J. Log. Algebr. Methods Program. 125 (2022) 100743.
[18] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, M. Vardi, Regular vacuity, in: D. Borrione, W.J. Paul (Eds.), CHARME 2005, in: LNCS, vol. 3725, Springer, 2005, pp. 191–206.
[19] A. Corradini, T. Heindel, F. Hermann, B. König, Sesqui-pushout rewriting, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.), ICGT 2006, in: LNCS, vol. 4178, Springer, 2006, pp. 30–45.
[20] U. Norell, Dependently typed programming in Agda, in: A. Kennedy, A. Ahmed (Eds.), TLDI 2009, ACM, 2009, pp. 1–2.
[21] J.Z.S. Hu, J. Carette, Formalizing category theory in Agda, in: C. Hritcu, A. Popescu (Eds.), CPP 2021, ACM, 2021, pp. 327–342.
[22] F. Gadducci, A. Laretto, D. Trotta, Specification and verification of a linear-time temporal logic for graph transformation, in: M. Fernández, C.M. Poskitt (Eds.), ICGT 2023, in: LNCS, vol. 13961, Springer, 2023, pp. 22–42.
[23] A. Laretto, Positive normal forms for counterpart-based temporal logics, [Software] SWHID: swh:1:dir:b3429f420a16c1d05b31e14387c72d9986450b6b;origin=https://github.com/iwilare/qltl-pnf;visit=swh:1:snp:c87315cde6c3208c0d4dba447e65e174584763d3;anchor=swh:1:rev:7dc3264b12885b0cba761d13ce67c05c122a999b, 2022.
[24] P. Blackburn, J. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, vol. 3, North-Holland, 2007.
[25] F.W. Lawvere, Adjointness in foundations, Dialectica 23 (1969) 281–296.
[26] B. Jacobs, Categorical Logic and Type Theory, North-Holland, 2001.

[27] S. Ghilardi, G. Meloni, Modal logics with n-ary connectives, Math. Log. Q. 36 (3) (1990) 193–215.
[28] S. Niefield, Lax presheaves and exponentiability, Theory Appl. Categ. 24 (12) (2010) 288–301.
[29] P. Freyd, A. Scedrov, Categories, Allegories, Elsevier, 1990.
[30] P. Gardiner, C. Martin, O. de Moor, An algebraic construction of predicate transformers, Sci. Comput. Program. 22 (1) (1994) 21–44.
[31] A. Laretto, Categorical semantics for counterpart-based temporal logics, [Software] SWHID: swh:1:dir:3517d066ed55e949b02871da83d44093af1f6548;origin=https://github.com/iwilare/categorical-qtl;visit=swh:1:snp:52feb0fe5667635f0247622e0b0ceff33b69a326;anchor=swh:1:rev:133b359b68cdee3f9824f266ab43886484b817b5, 2022.
[32] P. Wadler, W. Kokke, J.G. Siek, Programming language foundations in Agda, https://plfa.inf.ed.ac.uk/, 2022.
[33] N.A. Danielsson, Up-to techniques using sized types, in: POPL 2018, ACM, 2018, pp. 43:1–43:28.
[34] J. Girard, Y. Lafont, P. Taylor, Proofs and Types, Cambridge University Press, 1989.
[35] Coq Development Team, the Coq Proof Assistant Reference Manual, 2016.
[36] L. de Moura, S. Ullrich, The Lean 4 theorem prover and programming language, in: A. Platzer, G. Sutcliffe (Eds.), CADE 2021, in: LNCS, vol. 12699, Springer, 2021, pp. 625–635.
[37] T. Nipkow, L.C. Paulson, M. Wenzel, Isabelle/HOL - a Proof Assistant for Higher-Order Logic, LNCS, vol. 2283, Springer, 2002.
[38] F. Lindblad, M. Benke, A tool for automated theorem proving in Agda, in: J. Filliâtre, C. Paulin-Mohring, B. Werner (Eds.), TYPES 2004, in: LNCS, vol. 3839, Springer, 2006, pp. 154–169.
[39] A.H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, M. Zimakova, Modelling and analysis using GROOVE, Int. J. Softw. Tools Technol. Transf. 14 (1) (2012) 15–40.
[40] W. Smid, A. Rensink, Class diagram restructuring with GROOVE, in: P.V. Gorp, L.M. Rose, C. Krause (Eds.), TTC 2013, in: EPTCS, vol. 135, 2013, pp. 83–87.
[41] E. Zambon, A. Rensink, Recipes for coffee: compositional construction of JAVA control flow graphs in GROOVE, in: P. Müller, I. Schaefer (Eds.), Principled Software Development, Springer, 2018, pp. 305–323.
[42] S. Coupet-Grimal, An axiomatization of linear temporal logic in the calculus of inductive constructions, J. Log. Comput. 13 (6) (2003) 801–813.
[43] C. Sprenger, A verified model checker for the modal μ-calculus in Coq, in: B. Steffen (Ed.), TACAS 1998, in: LNCS, vol. 1384, Springer, 1998, pp. 167–183.
[44] D. Zanarini, C. Luna, L. Sierra, Alternating-time temporal logic in the calculus of (co)inductive constructions, in: R. Gheyi, D.A. Naumann (Eds.), SBMF 2012, in: LNCS, vol. 7498, Springer, 2012, pp. 210–225.
[45] J. Esparza, P. Lammich, R. Neumann, T. Nipkow, A. Schimpf, J. Smaus, A fully verified executable LTL model checker, in: N. Sharygina, H. Veith (Eds.), CAV 2013, in: LNCS, vol. 8044, Springer, 2013, pp. 463–478.
[46] L. O'Connor, Applications of applicative proof search, in: J. Chapman, W. Swierstra (Eds.), TyDe@ICFP 2016, ACM, 2016, pp. 43–55.
[47] A. Jeffrey, LTL types FRP: linear-time temporal logic propositions as types, proofs as functional reactive programs, in: K. Claessen, N. Swamy (Eds.), PLPV 2012, ACM, 2012, pp. 49–60.
[48] G.S. Wulandari, D. Plump, Verifying graph programs with monadic second-order logic, in: F. Gadducci, T. Kehrer (Eds.), ICGT 2021, in: LNCS, vol. 12741, Springer, 2021, pp. 240–261.
[49] C.M. Poskitt, D. Plump, Monadic second-order incorrectness logic for GP 2, J. Log. Algebr. Methods Program. 130 (2023) 100825.
[50] M. Búr, K. Marussy, B.H. Meyer, D. Varró, Worst-case execution time calculation for query-based monitors by witness generation, ACM Trans. Embed. Comput. Syst. 20 (6) (2021) 107:1–107:36.
[51] K. Marussy, O. Semeráth, A.A. Babikian, D. Varró, A specification language for consistent model generation based on partial models, J. Object Technol. 19 (3) (2020) 1–22.
[52] J. Schneider, D. Basin, S. Krstić, D. Traytel, A formally verified monitor for metric first-order temporal logic, in: B. Finkbeiner, L. Mariani (Eds.), RV 2019, in: LNCS, vol. 11757, Springer, 2019, pp. 310–328.
[53] J. Schneider, D. Traytel, Formalization of a monitoring algorithm for metric first-order temporal logic, Arch. Formal Proofs 2019 (2019).
[54] F. Hublet, D. Basin, S. Krstić, Real-time policy enforcement with metric first-order temporal logic, in: V. Atluri, R. Di Pietro, C.D. Jensen, W. Meng (Eds.), ESORICS 2022, in: LNCS, vol. 13555, Springer, 2022, pp. 211–232.
[55] J. Huerta y Munive, Relaxing safety for metric first-order temporal logic via dynamic free variables, in: T. Dang, V. Stolz (Eds.), RV 2022, in: LNCS, vol. 13498, Springer, 2022, pp. 45–66.
[56] H. Giese, M. Maximova, L. Sakizloglou, S. Schneider, Metric temporal graph logic over typed attributed graphs, in: R. Hähnle, W. van der Aalst (Eds.), FASE 2019, in: LNCS, vol. 11424, Springer, 2019, pp. 282–298.
[57] S. Schneider, L. Sakizloglou, M. Maximova, H. Giese, Optimistic and pessimistic on-the-fly analysis for metric temporal graph logic, in: F. Gadducci, T. Kehrer (Eds.), ICGT 2020, in: LNCS, vol. 12150, Springer, 2020, pp. 276–294.
[58] S. Schneider, M. Maximova, L. Sakizloglou, H. Giese, Formal testing of timed graph transformation systems using metric temporal graph logic, Int. J. Softw. Tools Technol. Transf. 23 (3) (2021) 411–488.
[59] W. Kokke, W. Swierstra, Auto in Agda - Programming proof search using reflection, in: R. Hinze, J. Voigtländer (Eds.), MPC 2015, in: LNCS, vol. 9129, Springer, 2015, pp. 276–301.
[60] G.D. Giacomo, M.Y. Vardi, Synthesis for LTL and LDL on finite traces, in: Q. Yang, M.J. Wooldridge (Eds.), IJCAI 2015, AAAI Press, 2015, pp. 1558–1564.