Specification and Verification of a Linear-Time Temporal Logic for Graph Transformation

Fabio Gadducci[1], Andrea Laretto[2(✉)], and Davide Trotta[1]

[1] Department of Computer Science, University of Pisa, Pisa, Italy
`fabio.gadducci@unipi.it, trottadavide92@gmail.com`
[2] Department of Software Science, Tallinn University of Technology, Tallinn, Estonia
`andrea.laretto@taltech.ee`

Abstract. We present a first-order linear-time temporal logic for reasoning about the evolution of directed graphs. Its semantics is based on the counterpart paradigm, thus allowing our logic to represent the creation, duplication, merging, and deletion of elements of a graph as well as how its topology changes over time. We then introduce a positive normal forms presentation, thus simplifying the actual process of verification. We provide the syntax and semantics of our logics with a computer-assisted formalisation using the proof assistant Agda, and we round up the paper by highlighting the crucial aspects of our formalisation and the practical use of quantified temporal logics in a constructive proof assistant.

Keywords: Counterpart semantics · Linear-time logics · Agda formalisation

1 Introduction

Among the many tools provided by formal methods, temporal logics have proven to be one of the most effective techniques for the verification of both large-scale and stand-alone programs. Along the years, the research on these logics focused on improving the algorithmic procedures for the verification process as well as on finding sufficiently expressive fragments of these logics for the specification of complex multi-component systems. Several models for temporal logics have been developed, with the leading example being transition systems, also known as Kripke structures. In a transition system, each state represents a configuration of the system and each transition identifies a possible state evolution. Often one is interested in enriching the states and transitions given by the model with more structure, for example by taking states as algebras and transitions as algebra homomorphisms. A prominent use case is that of graph logics [9,10,12], where states are specialised as graphs and transitions are families of (partial) graph

morphisms. These logics combine temporal and spatial reasoning and allow to express the possible transformation of a graph topology over time.

Quantified Temporal Logics. Under usual temporal logics, such as LTL and CTL [13], the states of the model are taken as atomic, with propositions holding for entire states: on the other hand, one of the defining characteristics of graph logics is that they permit to reason and to express properties on the individual elements of the graph. Despite their undecidability [15,26], 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.

The semantical models of these logics require some ingenuity, though. Consider a simple model with two states s_0, s_1, two transitions $s_0 \rightarrow s_1$ and $s_1 \rightarrow s_0$, and an item i that appears only in s_0. Is the item i being destroyed and recreated, or is it just an identifier being reused multiple times? This issue is denoted in the literature as the *trans-world identity problem* [2,25]. A solution 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.

Counterpart Semantics. A solution to this problem was proposed by Lewis [31] 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 us to speak formally about entities that are destroyed, duplicated, (re)created, or merged, and to adequately deal with the identity problem of individuals between graphs.

In [16], a counterpart-based semantics is used to introduce a set-theoretical semantics of a μ-calculus with second-order quantifiers. This modal logic provides a formalism that enriches states with algebras and transitions with partial homomorphisms, subsuming the case of graph logics. These models are generalised to a categorical setting in [17] by means of relational presheaves, building on the ideas presented in [20,21]. The models are represented with categories and (families of) relational presheaves, which give a categorical representation for the states-as-algebras approach with partial homomorphisms. The temporal advancement of a system is captured by equipping categories with the notion of *one-step* arrows for a model, and the categorical framework is then used to introduce a second-order linear temporal logic QLTL.

Classical Semantics and Positive Normal Form. We start by introducing the notion of counterpart models in Sect. 2, and present the (admittedly rather straightforward) syntax of our temporal logic QLTL as well as its counterpart-based semantics in Sect. 3, using a standard set-theoretic perspective, with satisfiability given inductively as a logical predicate. Unlike [16,17] where the models

use *partial functions*, we generalise to *relations*, thus modelling the duplication of elements by allowing it to have multiple counterparts. In Sect. 4 we present some results on the positive normal forms, where the models may use either partial morphisms or relations, and we highlight 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 facilitating model checking algorithms [5,27]. 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 [7], and thus worthy of investigation.

Temporal Logics in Agda. An additional contribution of our work is a computer-assisted formalisation using the dependently typed proof assistant Agda [37] of the models, semantics, and positive normal forms of the logic presented in this paper. We introduce the main aspects of the mechanisation in Sect. 5, which can be adapted for counterpart-based models whose worlds are algebras on any multi-sorted signature, even if for the sake of presentation in this paper we restrict our attention to graph signatures. A formal presentation of a temporal logic in a proof assistant has several advantages: it 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; moreover, the mechanisation effectively provides a playground in which the mechanisms and validity of these logics can be expressed, tested, and experimented with.

   To the best of our knowledge, few formalisations of temporal logics have been provided with a proof assistant, and none of these comes equipped with a counterpart-based semantics. This work constitutes a step towards the machine-verified use of temporal logics by embedding in an interactive proof assistant a quantified extension of LTL that can reason on individual elements of states.

2   Counterpart Models

This section introduces our models for system evolution. We consider the instantiation of counterpart models to the case where each world is not associated to a mere (structureless) set of individuals, but to a directed graph with its evolution in time being represented by suitable relations preserving the graph structure.

Definition 2.1. *A **(directed) graph** is a 4-tuple $G := \langle N, E, s, t \rangle$ such that N is a set of nodes, E is a set of edges, and $s, t : E \to N$ are two functions assigning a* source *node $s(e) \in N$ and a* target *node $t(e) \in N$ to each edge $e \in E$, respectively. The set of all directed graphs is denoted as* Graphs.

Definition 2.2. *A **graph (relational) morphism** between two graphs $G = \langle N, E, s, t \rangle$ and $G' = \langle N', E', s', t' \rangle$ is a pair $R := \langle R_N, R_E \rangle$ such that $R_N \subseteq N \times N'$ and $R_E \subseteq E \times E'$ are relations between nodes and edges of the graphs such that $e_1 R_E e_2$ implies $s(e_1) R_N s'(e_2)$ and $t(e_1) R_N t'(e_2)$. Given graphs G, G', the set of graph morphisms is denoted* GraphRel$(G, G') \subseteq \mathscr{P}((N \times N') \times (E \times E'))$.*

Definition 2.3. *A **counterpart model** is a triple $\mathfrak{M} := \langle W, D, \mathcal{C} \rangle$ such that*

- *W is a non-empty set of elements, called* worlds,
- *$D : W \rightarrow$ **Graphs** is a function assigning a directed graph to each world,*
- *$\mathcal{C} : W \times W \rightarrow \mathscr{P}(\textbf{GraphRel}(D(\omega), D(\omega')))$ is a function assigning to every pair $\langle \omega, \omega' \rangle$ a set of graph morphisms $\mathcal{C}\langle \omega, \omega' \rangle \subseteq \textbf{GraphRel}(D(\omega), D(\omega'))$, where every $C \in \mathcal{C}\langle \omega, \omega' \rangle$ is a graph morphism between the graphs associated to the two worlds. We refer to these as **atomic counterpart relations**.*

Given two worlds ω and ω', the set $\mathcal{C}\langle \omega, \omega' \rangle$ 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 $\mathcal{C}\langle \omega, \omega' \rangle$ is empty, there are no atomic transitions from ω to ω'. Each atomic relation $C \in \mathcal{C}\langle \omega, \omega' \rangle$ connects the nodes and edges between two worlds ω and ω', intuitively identifying them as the same component after one time evolution of the model. For example, if we consider two nodes $n \in D(\omega)_N$ and $n' \in D(\omega')_N$ and a relation $C \in \mathcal{C}\langle \omega, \omega' \rangle$, if $\langle n, n' \rangle \in C_N$ then n' represents a future development of the node n via C.

Definition 2.4. *A node $s' \in D(\omega')_N$ **is the counterpart of** $s \in D(\omega)_N$ through a counterpart relation C whenever $\langle s, s' \rangle \in C_N$, and similarly for edges.*

Example 2.1. (Counterpart model). We give an example of a counterpart model by indicating the set of worlds $\{\omega_0, \omega_1, \omega_2\}$ and the cardinality of the sets of relations $\mathcal{C}\langle \omega, \omega' \rangle$ in Fig. 1; the graph structures associated to each world and the graph morphisms connecting them are shown in Fig. 2. There are two counterpart relations $C_1, C_2 \in \mathcal{C}\langle \omega_1, \omega_2 \rangle$ between ω_1 and ω_2, and we use blue dashed and green dotted lines to distinguish C_1 and C_2, respectively.
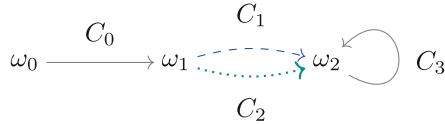
Fig. 1. Graphical representation of the worlds and accessibility relations of a model.
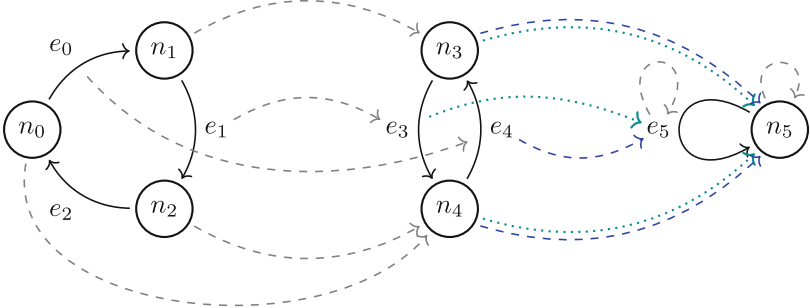
Fig. 2. Graphical representation of a model.

The use of *relations* as transitions allows us to model the removal of edges and nodes of a graph, by having no counterpart for them in the next state. For example, if there is no edge $e' \in D(\omega')_E$ such that $\langle e, e' \rangle \in C_E$, we conclude that the edge e has been removed by C. Similarly, the duplication of a node is represented by connecting it with two instances of the counterpart relation, e.g. by having elements $n'_1, n'_2 \in D(\omega')_N$ such that $\langle n, n'_1 \rangle \in C_N$ and $\langle n, n'_2 \rangle \in C_N$.

The simple counterpart model in Example 2.1 displays the effects of merging and deletion. The first counterpart relation C_0 merges the nodes n_0 and n_2 of ω_0, yet this act does not generate a cycle: in fact, e_2 is deleted, since it is not connected to a counterpart in ω_1. Similarly, both counterpart relations C_1 and C_2 merge the nodes n_3 and n_4 of ω_1, while they differ in which edge they remove when transitioning from ω_1 to ω_2, by deleting either the edge e_3 or e_4, respectively. Note that in both cases the nodes n_3 and n_4 need to be preserved, albeit possibly merged, in order to ensure that the relations considered are actually (relational) morphisms of graphs.

2.1 Counterpart Relations and Traces

We assume hereafter a fixed counterpart model $\mathfrak{M} := \langle W, D, \mathcal{C} \rangle$ and we formally introduce the idea behind counterpart relations. We indicate composition of graph morphisms in diagrammatic order: as an example, given $C \in$ GraphRel(G_1, G_2) and $C' \in$ GraphRel(G_2, G_3), the composite graph morphism is denoted with $C; C' \in$ GraphRel(G_1, G_3) and is such that $(C; C')_N = \{(a, c) \mid \exists b \in G_{2N}. \langle a, b \rangle \in C_N \wedge \langle b, c \rangle \in C'_N\}$, and similarly for edges.

Definition 2.5. *A graph morphism $C \in$ GraphRel$(D(\omega), D(\omega'))$ is a **counterpart relation** of the model if one of the following cases holds*

- – *C is the identity graph morphism;*
- – *$C \in \mathcal{C}\langle\omega, \omega'\rangle$ is an atomic graph morphism given by the model $\mathfrak{M}$;*
- – *C is the composite of a sequence of counterpart relations $C_0; \cdots ; C_n$ with $C_i \in \mathcal{C}\langle\omega_i, \omega_{i+1}\rangle$.*

Note that the composition $C; C' \in$ GraphRel$(D(\omega_1), D(\omega_3))$ of two atomic counterpart relations $C \in \mathcal{C}\langle\omega_1, \omega_2\rangle$ and $C' \in \mathcal{C}\langle\omega_2, \omega_3\rangle$ might not be atomic, and the models define only atomic transitions. Transitioning directly between two graphs might differ from transitioning through an intermediate one, since the direct transition is not necessarily the composition of the two counterpart relations. Moreover, the former requires one evolution step, the latter two.

As is the case of LTL where we can identify traces connecting linearly evolving states, see for example [3], we can consider linear sequences of counterpart relations providing a list of sequentially accessible worlds with associated graphs.

Definition 2.6. *A **trace** σ is an infinite sequence of atomic counterpart relations $(C_0, C_1, \dots)$ such that $C_i \in \mathcal{C}\langle\omega_i, \omega_{i+1}\rangle$ for any $i \geq 0$.*

In other words, a trace identifies a path in the graph induced by the counterpart model, with worlds as nodes and atomic counterpart relations as edges. Given a trace $\sigma = (C_0, C_1, \dots)$, we use $\sigma_i := (C_i, C_{i+1}, \dots)$ 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 of the trace σ whenever it is clear from the context. Similarly, 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. Should $i = 0$, the relation $C_{\leq 0}$ is the identity graph morphism on ω_0.

3   Quantified Linear Temporal Logic

We present the syntax and semantics of our (first-order) quantified linear temporal logic QLTL by adopting a standard set-theoretic presentation.

3.1   Syntax and Semantics of QLTL

Since free variables may appear inside formulae, we recall the usual presentation of context and terms-in-context. For the sake of simplicity we consider algebras whose terms represent directed graphs, as per Definition 2.1. More precisely, the signature has two sorts E and N and two functions symbols s and t, obviously representing the source and target functions on edges, respectively. We assume a set of sorted variables $\mathcal{X} = \mathcal{X}_N \uplus \mathcal{X}_E$ and define a typed context Γ as a finite subset of such variables, using the notation $[\Gamma] \, n : N$ to indicate that the term n has type N and it is constructed in a typed context Γ, that is, its free variables are contained among those occurring in Γ (and similarly for $[\Gamma] \, e : E$).

In order to give a simpler presentation for the semantics of temporal logics, it is customary to exclude the elementary constructs that can be expressed in terms of other operators. We thus present QLTL with a minimal set of standard operators and derive conjunction and universal quantification by using negation.

Definition 3.1. (QLTL). *Let Γ be a typed context on the set of variables $\mathcal{X}$. The set $\mathcal{F}_\Gamma^{QLTL}$ of QLTL formulae is generated by the following rules*

$$\psi := \mathsf{true} \mid e_1 =_E e_2 \mid n_1 =_N n_2$$

$$\phi := \psi \mid \neg\phi \mid \phi \vee \phi \mid \exists_N x.\phi \mid \exists_E x.\phi \mid \mathsf{O}\phi \mid \phi\mathsf{U}\phi \mid \phi\mathsf{W}\phi,$$

where $[\Gamma] \, e_i : E$ and $[\Gamma] \, n_i : N$ for $i = 1, 2$.

The above definition actually provides formulae-in-context: we use the notation $[\Gamma]\phi$ to indicate that a formula ϕ belongs to $\mathcal{F}_\Gamma^{QLTL}$ for the typed context Γ. Clearly, saying that $[\Gamma]\phi$ is the same as stating that $\mathrm{fv}(\phi) \subseteq \Gamma$, i.e. the free variables of ϕ are contained among those occurring in Γ. For both terms and formulae we omit the bracketed context whenever it is unnecessary to specify it.

The letter ψ denotes the set of *atomic formulae*, built out of two sorted equivalence predicates. Given two edge terms $e_1, e_2 : E$, the formula $e_1 =_E e_2$ indicates that the two edges coincide in the graph associated to the current world, and similarly for two node terms $n_1, n_2 : N$ and the formula $n_1 =_N n_2$.

The existential operators $\exists_N x.\phi$ and $\exists_E x.\phi$ can be used to express the existence of a node (edge, respectively) in the current graph satisfying a certain property ϕ, where the variable x is allowed to appear as a free variable of ϕ.

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 exist indefinitely without ever reaching a point where ϕ_2 holds, provided that ϕ_1 also keeps holding indefinitely.

The *dual* operators are syntactically expressed by $\mathsf{false} := \neg\mathsf{true}$, $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$, and $\forall_N x.\phi := \neg\exists_N x.\neg\phi$ and similarly for edges. 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 3.3.

3.2   Satisfiability

To present the notion of satisfiability of a formula with respect to a counterpart model we introduce the definition of *assignment* for a context in a world.

Definition 3.2. (Assignment). *An **assignment** in the world $\omega \in W$ for the typed context Γ is a pair of functions $\mu := \langle \mu_N, \mu_E \rangle$ such that $\mu_N : \Gamma_N \to D(\omega)_N$ and $\mu_E : \Gamma_E \to D(\omega)_E$. We use the notation $\mathcal{A}_\omega^\Gamma$ to indicate the set of assignments μ defined in ω for the typed context Γ.*

Moreover, we denote by $\mu[x \mapsto_\tau n] \in \mathcal{A}_\omega^{\Gamma,(x:\tau)}$ the assignment obtained by extending the domain of μ with $n \in D(\omega)_\tau$ at the variable $x \notin \Gamma$, omitting the type $\tau \in \{\mathsf{N}, \mathsf{E}\}$ whenever clear from context. We indicate with $\Gamma, (x : \tau)$ the context Γ extended with an additional variable x with sort τ.

An assignment μ for a typed context Γ provides the interpretation of terms-in-context whose context is (contained in) Γ: it allows for evaluating the free variables of the term, thus defining its semantics (with respect to that assignment). This will be used in Definition 3.5 to provide a meaning for the equalities $n_1 =_N n_2$ and $e_1 =_E e_2$ for nodes and edges, respectively.

Definition 3.3. (Assignment on terms). *Given an assignment $\mu \in \mathcal{A}_\omega^\Gamma$,we indicate with $\mu^* := \langle \mu_N^*, \mu_E^* \rangle$ the interpretation of μ on a term-in-context $[\Gamma] \, n :$ N or $[\Gamma] \, e : \mathsf{E}$, given inductively by the rules: $\mu_E^*(x) := \mu_E(x)$, $\mu_N^*(y) := \mu_N(y)$, $\mu_N^*(s(e)) := D(\omega)_s(\mu_E^*(e))$, and $\mu_N^*(t(e)) := D(\omega)_t(\mu_E^*(e))$.*

We now look at how to transport assignments over counterpart relations. The intuition is that we have to connect the nodes and the edges in the image of two assignments when there is a counterpart relation among the worlds, and the items of the underlying graphs are related point-wise.

Definition 3.4. (Counterpart relations on assignments). *Given a counterpart relation $C \in \mathsf{GraphRel}(D(\omega_1), D(\omega_2))$ and two assignments $\mu_1 \in \mathcal{A}_{\omega_1}^\Gamma$ and $\mu_2 \in \mathcal{A}_{\omega_2}^\Gamma$ on the context Γ, we say that the assignments μ_1 and μ_2*

are **counterpart related** if $\langle \mu_{1N}(x), \mu_{2N}(x) \rangle \in C_N$ for any $x \in \Gamma_N$ and $\langle \mu_{1E}(x), \mu_{2E}(x) \rangle \in C_E$ for any $x \in \Gamma_E$. We indicate this with the notation $\langle \mu_1, \mu_2 \rangle \in C$.

We can now introduce the notion of satisfiability of a QLTL formula with respect to a trace σ and an assignment μ.

Definition 3.5. (QLTL satisfiability). *Given a QLTL formula-in-context* $[\Gamma]\phi$, *a trace* $\sigma = (C_0, C_1, \dots)$, *and an assignment* $\mu \in \mathcal{A}_{\omega_0}^{\Gamma}$ *in the first world of* σ, *we inductively define the* satisfiability relation *as follows*

- $\sigma, \mu \vDash$ true;
- $\sigma, \mu \vDash e_1 =_E e_2$ *if* $\mu_E^*(e_1) = \mu_E^*(e_2)$;
- $\sigma, \mu \vDash n_1 =_N n_2$ *if* $\mu_N^*(n_1) = \mu_N^*(n_2)$;
- $\sigma, \mu \vDash \neg\phi$ *if* $\sigma, \mu \nvDash \phi$;
- $\sigma, \mu \vDash \phi_1 \lor \phi_2$ *if* $\sigma, \mu \vDash \phi_1$ *or* $\sigma, \mu \vDash \phi_2$;
- $\sigma, \mu \vDash \exists_N x.\phi$ *if there is a node* $n \in D(\omega_0)_N$ *such that* $\sigma, \mu[x \mapsto n] \vDash \phi$;
- $\sigma, \mu \vDash \exists_E x.\phi$ *if there is an edge* $e \in D(\omega_0)_E$ *such that* $\sigma, \mu[x \mapsto e] \vDash \phi$;
- $\sigma, \mu \vDash \mathsf{O}\phi$ *if there is* $\mu_1 \in \mathcal{A}_{\omega_1}^{\Gamma}$ *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 an* $\bar{n} \geq 0$ *such that*
  1. *for any* $i < \bar{n}$, *there is* $\mu_i \in \mathcal{A}_{\omega_i}^{\Gamma}$ *such that* $\langle \mu, \mu_i \rangle \in C_{\leq i}$ *and* $\sigma_i, \mu_i \vDash \phi_1$;
  2. *there is* $\mu_{\bar{n}} \in \mathcal{A}_{\omega_{\bar{n}}}^{\Gamma}$ *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; or*
  - *for any* i *there is* $\mu_i \in \mathcal{A}_{\omega_i}^{\Gamma}$ *such that* $\langle \mu, \mu_i \rangle \in C_{\leq i}$ *and* $\sigma_i, \mu_i \vDash \phi_1$.

3.3 Examples

We provide some examples of satisfiability for QLTL formulae on the running example in Fig. 2 to illustrate how our counterpart semantics works in practice. Take for example the trace given by $\sigma := (C_0, C_1, C_3, C_3, \dots)$, thus considering the case where e_4 is the only edge preserved when transitioning from ω_1.

Example 3.1. (Allocation and deallocation). As anticipated in Sect. 2, one of the main advantages of a counterpart semantics is the possibility to reason about existence, deallocation, duplication, and merging of elements in the system and its evolution. Consider for example the following shorthand formulae

$$\mathbf{present}_\tau(x) := \exists_\tau y.x =_\tau y$$
$$\mathbf{nextPreserved}_\tau(x) := \mathbf{present}_\tau(x) \land \mathsf{O}\mathbf{present}_\tau(x)$$
$$\mathbf{nextDealloc}_\tau(x) := \mathbf{present}_\tau(x) \land \neg\mathsf{O}\mathbf{present}_\tau(x)$$

The formula $\mathbf{present}_\tau(x)$ captures the existence of a node or an edge at the current moment. We can 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, which we condense with the $\mathbf{nextPreserved}_\tau(x)$ formula. We can similarly refer to elements that are now present but that will be deallocated at the next step by considering the formula $\mathbf{nextDealloc}_\tau(x)$. Indeed we have

$$\sigma_0, \{x : \mathsf{E} \mapsto e_0 \} \vDash \mathbf{nextPreserved}_E(x); \sigma_1, \{x : \mathsf{N} \mapsto n_3\} \nvDash \mathbf{nextDealloc}_N(x)$$
$$\sigma_0, \{x : \mathsf{N} \mapsto n_1\} \vDash \mathbf{nextPreserved}_N(x); \sigma_1, \{x : \mathsf{E} \mapsto e_3 \} \vDash \mathbf{nextDealloc}_E(x)$$
$$\sigma_0, \{x : \mathsf{E} \mapsto e_2 \} \nvDash \mathbf{nextPreserved}_E(x); \sigma_1, \{x : \mathsf{E} \mapsto e_4 \} \nvDash \mathbf{nextDealloc}_E(x)$$

Example 3.2. (Graph structure). Moreover, our syntax allows us to define formulae that exploit the algebraic structure of our graphs, and combine them with the temporal operators to state properties about how the graph evolves in time. We illustrate this by providing the following formulae

$$\mathbf{loop}(e) := s(e) =_N t(e)$$
$$\mathbf{hasLoop}(n) := \exists_E e.s(e) =_N n \wedge \mathbf{loop}(e)$$
$$\mathbf{composable}(x, y) := t(x) =_N s(y)$$
$$\mathbf{haveComposition}(x, y) := \mathbf{composable}(x, y) \wedge \exists_E e.(s(e) =_N s(x) \wedge t(e) =_N t(y))$$
$$\mathbf{adjacent}(x, y) := \exists_E e.((s(e) =_N x \wedge t(e) =_N y) \vee (t(e) =_N x \wedge s(e) =_N y))$$

which capture, respectively, the following scenarios: we can check whether a given edge of the graph is a loop with $\mathbf{loop}(x)$, or verify with $\mathbf{hasLoop}(x)$ that the only node having a loop is n_5; alternatively, we can express this fact by stating that the loop belongs to the entire world using $\mathbf{hasLoop}$

$$\sigma_0, \{x \mapsto e_0\} \nvDash \mathbf{loop}(x); \sigma_0, \{x \mapsto n_0\} \nvDash \mathbf{hasLoop}(x); \sigma_0, \{\} \nvDash \exists_N x.\mathbf{hasLoop}(x)$$
$$\sigma_1, \{x \mapsto e_3\} \nvDash \mathbf{loop}(x); \sigma_1, \{x \mapsto n_3\} \nvDash \mathbf{hasLoop}(x); \sigma_1, \{\} \nvDash \exists_N x.\mathbf{hasLoop}(x)$$
$$\sigma_2, \{x \mapsto e_5\} \vDash \mathbf{loop}(x); \sigma_2, \{x \mapsto n_5\} \vDash \mathbf{hasLoop}(x); \sigma_2, \{\} \vDash \exists_N x.\mathbf{hasLoop}(x)$$

Since $\exists_N x.\mathbf{hasLoop}(x)$ is a closed formula, the empty assignment is the only one the formula can be valued on. Thus, we obtain the classical notion of a formula simply providing the binary information of being true or false, with no choice of individuals needed to satisfy it. Finally, we can express some properties about the existence of intermediate nodes and composability of edges in the graph

$$\sigma_0, \{x \mapsto n_0, y \mapsto n_1\} \vDash \mathbf{adjacent}(x, y); \quad \sigma_0, \{x \mapsto e_0, y \mapsto e_1\} \vDash \mathbf{composable}(x, y)$$
$$\sigma_1, \{x \mapsto n_3, y \mapsto n_4\} \vDash \mathbf{adjacent}(x, y); \quad \sigma_0, \{x \mapsto e_0, y \mapsto e_1\} \vDash \mathsf{O}\mathbf{composable}(x, y)$$
$$\sigma_1, \{x \mapsto n_3, y \mapsto n_4\} \vDash \mathsf{O}\mathbf{adjacent}(x, y); \sigma_1, \{x \mapsto e_3, y \mapsto e_4\} \nvDash \mathsf{O}\mathbf{composable}(x, y)$$

$$\sigma_0, \{x \mapsto e_0, y \mapsto e_1\} \nvDash \mathbf{haveComposition}(x, y)$$
$$\sigma_0, \{x \mapsto e_0, y \mapsto e_1\} \nvDash \mathsf{O}\mathbf{haveComposition}(x, y)$$
$$\sigma_2, \{x \mapsto e_5, y \mapsto e_5\} \vDash \mathsf{O}\mathbf{haveComposition}(x, y)$$

*Remark 3.1. (*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. Alternatively, their semantics can be presented directly as

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

Example 3.3. (Temporal evolution). We can use these operators to express the evolution of the graph after an unspecified amount of steps

$$\mathbf{willMerge}_{\tau}(x, y) := x \neq_{\tau} y \wedge \Diamond(x =_{\tau} y)$$
$$\mathbf{alwaysPreserved}_{\tau}(x) := \Box\mathbf{present}_{\tau}(x)$$
$$\mathbf{willBecomeLoop}(e) := \neg\mathbf{loop}(e) \wedge \Diamond\mathbf{loop}(e)$$

In the example in Fig. 1 for the same trace $\sigma = (C_0, C_1, C_3, \dots)$ we have

$\sigma_0, \{\} \vDash \exists n. \exists m. \textbf{willMerge}_N(n, m),$ $\sigma_0, \{\} \nvDash \forall e. \Diamond \textbf{loop}(e),$

$\sigma_0, \{\} \nvDash \forall e. \textbf{alwaysPreserved}_E(e),$ $\sigma_0, \{\} \vDash \exists e. \Diamond \Box \textbf{loop}(e),$

$\sigma_0, \{\} \vDash \exists e. \textbf{willBecomeLoop}(e),$ $\sigma_0, \{\} \vDash \exists x. \exists y. \neg \Diamond \textbf{composable}(x, y),$

$\sigma_0, \{\} \vDash (\exists e. s(e) \neq t(e)) \mathsf{U} (\exists x. \forall y. x = y), \sigma_0, \{\} \vDash (\exists e. s(e) = t(e)) \mathsf{W} \neg (\exists x. \textbf{loop}(e)).$

Remark 3.2. (Quantifier elision for unbound variables). A relevant difference with standard quantified logics is that in QLTL we cannot elide quantifications where the variable introduced does not appear in the subformula. 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 whenever ϕ does not contain any temporal operator and the current world $D(\omega)$ being considered is not empty.

Consider a world ω with a single node $D(\omega)_N = \{s\}$, no edges, and a single looping counterpart relation $\mathcal{C}\langle \omega, \omega \rangle = \{C\}$ 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 $\sigma, \{\} \nvDash \exists_N x. \mathsf{O}(\mathsf{true})$. 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 $\mu_1 \in A_{\omega_1}^{\mathrm{fv}(\phi)}$ such that $\langle \mu_{|\mathrm{fv}(\phi)}, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi.$

For ease of presentation in this work and with respect to our Agda implementation, we consider the case where all elements in the context have a counterpart.

*Remark 3.3. (*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, it might at first seem reasonable to define the standard *always* operator in QLTL with $\Box\phi := \neg\Diamond\neg\phi$. However, this definition does not align with the semantics provided in Remark 3.1. This characteristic of QLTL is again due to the fact that we are in the setting of (possibly deallocating) relations, and we formally explain and present an intuition for this when we introduce the PNF semantics in Sect. 4. The LTL equivalences can be obtained again by restricting to models whose counterpart relations are total functions: this allows us to consider a unique trace of counterparts that are always defined, which brings our models back to a standard LTL-like notion of trace.

4   Positive Normal Form for QLTL

Positive normal forms are a standard presentation of temporal logics: they can be used to simplify constructions and algorithms on both the theoretical and implemention side [5,27]. Their use is crucial for semantics based on fixpoints, such

as in [16], while still preserving the expressiveness of the original presentation. As we remark in Sect. 5, 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*. Moreover, the positive normal form conversion serves as a concrete procedure that can be used interactively to automatically convert formulae into their positive normal form, which is proven in the proof assistant to be correct. In this section we present an explicit semantics for the positive normal form of QLTL, which we denote as PNF.

4.1   Semantics of PNF

As observed in Remark 3.3, to present the positive normal form we need additional operators to adequately capture the negation of temporal operators. Thus, we introduce a new flavour of the next operator, called *next-forall* $\mathsf{A}\phi$. Similarly, we have to introduce a dual for *until* $\phi_1\mathsf{U}\phi_2$ and *weak until* $\phi_1\mathsf{W}\phi_2$, which we indicate as the *then* $\phi_1\mathsf{T}\phi_2$ and *until-forall* $\phi_1\mathsf{F}\phi_2$ operators, respectively.

Definition 4.1. (QLTL in PNF). *Let Γ be a typed context on the set of variables $\mathcal{X}$. The set $\mathcal{F}_\Gamma^{PNF}$ of formulae of QLTL in **positive normal form** is generated by the following rules*

$$\psi := \mathsf{true} \mid e_1 =_E e_2 \mid n_1 =_N n_2$$

$$\phi := \psi \mid \neg\psi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists_\tau x.\phi \mid \forall_\tau x.\phi \mid \mathsf{O}\phi \mid \mathsf{A}\phi \mid \phi\mathsf{U}\phi \mid \phi\mathsf{F}\phi \mid \phi\mathsf{W}\phi \mid \phi\mathsf{T}\phi,$$

where $[\Gamma]\ e_i : E$ and $[\Gamma]\ n_i : N$ for $i = 1, 2$ and $\tau \in \{N, E\}$.

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 graphs where ϕ_1 is satisfied for some steps until ϕ_2 holds. The crucial observation is that *every* intermediate counterpart 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* $\phi_1\mathsf{T}\phi_2$ operator corresponds to a *weak until-forall* and can be validated by a trace where all counterparts satisfy ϕ_1 without ever satisfying ϕ_2.

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

Definition 4.2. (QLTL in PNF satisfiability). *Given a QLTL formula-in-context $[\Gamma]\phi$ in positive normal form, a trace $\sigma = (C_0, C_1, \dots)$, and an assignment $\mu \in \mathcal{A}_{\omega_0}^\Gamma$ in the first world of σ, we inductively define the satisfiability relation with respect to the additional operators 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* $\sigma, \mu \vDash \phi_2$;

– $\sigma, \mu \vDash \forall_\tau x.\phi$ if for any $s \in D(\omega_0)_\tau$ 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 $\sigma_{\bar{n}}, \mu_{\bar{n}} \vDash \phi_2$;

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

    • the same conditions for $\phi_1 \mathsf{F} \phi_2$ apply; or

    • 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 $\sigma_i, \mu_i \vDash \phi_1$.
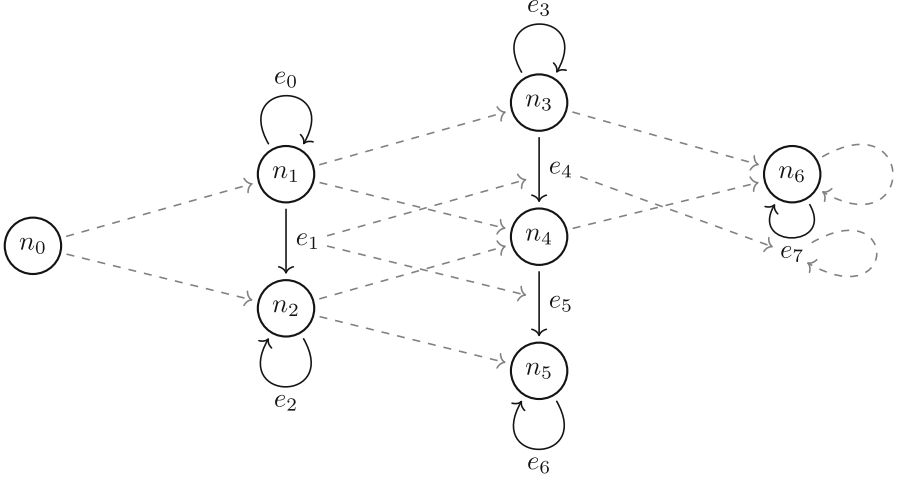
Fig. 3. Graphical representation of a counterpart model.

Example 4.1. We illustrate with the example in Fig. 3 the possibility for a counterpart relation to duplicate both edges and nodes of a graph, as well as providing some concrete cases for the new operators defined in Definition 4.2. For example, we have that $\sigma_0, \{x \mapsto n_0\} \vDash \mathsf{A}(\mathbf{hasLoop}(x))$, but $\sigma_1, \{x \mapsto n_1\} \nvDash \mathsf{A}(\mathbf{hasLoop}(x))$ since n_4 is a counterpart of n_1 but does not have a loop. Moreover, $\sigma_1, \{x \mapsto e_1\} \vDash \mathbf{hasLoop}(s(x))\mathsf{U}(\mathbf{loop}(x))$, but $\sigma_1, \{x \mapsto e_1\} \nvDash \mathbf{hasLoop}(s(x))\mathsf{F}(\mathbf{loop}(x))$ because we also require for e_5 to have a loop at its source since it is a counterpart of e_1. Notice that $\sigma_2, \{x \mapsto e_5\} \vDash \mathsf{A}(\mathbf{loop}(x))$ since there is no counterpart at the next step, and indeed we similarly have that $\sigma_2, \{x \mapsto n_5\} \vDash \mathbf{hasLoop}(x)\mathsf{F}(\mathsf{false})$. Finally, we have that $\sigma_2, \{x \mapsto e_4\} \vDash \mathbf{hasLoop}(s(x))\mathsf{T}(\neg\mathbf{loop}(x))$ because the intermediate condition always holds.

4.2   Negation of QLTL and PNF

The crucial observation that validates the PNF presented in Sect. 4 is that the negation of *next* $O\phi$, *until* $\phi_1 U\phi_2$, and *weak until* $\phi_1 W\phi_2$ formulae can now be expressed inside the logic. We indicate with $\vDash_{QLTL}$ and $\vDash_{PNF}$ the satisfiability relations for formulae in standard QLTL and QLTL in PNF, respectively.

Proposition 4.1. (Negation is expressible in PNF).
(✍ *Relational.Negation*)
Let ψ, ψ_1, ψ_2 *be atomic formulae in PNF. Then we have*

$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \vDash_{QLTL} \neg O(\psi) \qquad \Longleftrightarrow \sigma, \mu \vDash_{PNF} A(\neg\psi)$$
$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \vDash_{QLTL} \neg(\psi_1 U\psi_2) \Longleftrightarrow \sigma, \mu \vDash_{PNF} (\neg\psi_2)T(\neg\psi_1 \wedge \neg\psi_2)$$
$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \vDash_{QLTL} \neg(\psi_1 W\psi_2) \Longleftrightarrow \sigma, \mu \vDash_{PNF} (\neg\psi_2)F(\neg\psi_1 \wedge \neg\psi_2)$$

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

Proposition 4.2. (Negation of new operators is *not* in PNF). *Let* ψ, ψ_1, ψ_2 *be atomic formulae in PNF. Then we have*

$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \nvDash_{PNF} A(\psi) \qquad \Longleftrightarrow \sigma, \mu \vDash_{PNF} O(\neg\psi)$$
$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \nvDash_{PNF} \psi_1 T\psi_2 \quad \Longleftarrow\!\!\!\not\Longrightarrow \ \sigma, \mu \vDash_{PNF} (\neg\psi_2)U(\neg\psi_1 \wedge \neg\psi_2)$$
$$\forall \sigma, \mu \in \mathcal{A}^{\Gamma}_{\omega_0}. \ \sigma, \mu \nvDash_{PNF} \psi_1 F\psi_2 \quad \Longleftarrow\!\!\!\not\Longrightarrow \ \sigma, \mu \vDash_{PNF} (\neg\psi_2)W(\neg\psi_1 \wedge \neg\psi_2)$$

Proof. See [18, Proposition 4.2].

Let us now go back to Proposition 4.1. We exploit the correspondence between operators given there to define a translation $\overline{\cdot} : \mathcal{F}^{QLTL} \to \mathcal{F}^{PNF}$ from the QLTL syntax presented in Definition 3.5 to the current one in PNF. What is noteworthy is that such translation preserves the equivalence of formulae.

Theorem 4.3. (PNF equivalence). (✍ *Relational.Conversion*) *Let* $\overline{\cdot} :$ $\mathcal{F}^{QLTL} \to \mathcal{F}^{PNF}$ *be the syntactical translation that replaces negated temporal operators with their equivalent ones in PNF by pushing negation down to atomic formulae. For any QLTL formula* $[\Gamma]\phi \in \mathcal{F}^{QLTL}$ *we have*

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

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 4.4. (Expansion laws do *not* hold in QLTL). *We have the following inequalities in PNF*

$$\phi_1 U\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge O(\phi_1 U\phi_2)) \qquad \phi_1 F\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge A(\phi_1 F\phi_2))$$
$$\phi_1 W\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge O(\phi_1 W\phi_2)) \qquad \phi_1 T\phi_2 \not\equiv \phi_2 \vee (\phi_1 \wedge A(\phi_1 T\phi_2))$$

Proof. See [18, Proposition 4.4].

Remark 4.1. (Functional counterpart relations). The previous results can be reframed in the case in which each counterpart relation is a *partial function*, following the definition of counterpart models given in [16,17]. It turns out that under the assumption of partial functions we recover all the equivalences stated in Proposition 4.2 (✑ Functional.Negation) as well as the expansion laws of Proposition 4.4 (✑ Functional.ExpansionLaws). The latter can be used to provide a presentation of the temporal operators as *least fixpoint* ($\phi_1 \mathsf{U} \phi_2$, $\phi_1 \mathsf{F} \phi_2$) and *greatest fixpoints* ($\phi_1 \mathsf{W} \phi_2$, $\phi_1 \mathsf{T} \phi_2$) of a suitably defined operator; this definition based on fixpoints would coincide with the semantics of the operators given in Definition 4.2 only in the case of partial (or total) functions.

5   Agda Formalisation

This section presents an overview of an additional contribution of this work: a complete formalisation of the semantics of QLTL and its PNF using the dependently typed programming language and proof assistant Agda [37]. We provide a brief exposition and usage of our development in [18], showing how the temporal evolution of the running example in Fig. 2 can be concretely modelled in Agda. The complete formalisation of the logic along with the PNF results is available at https://github.com/iwilare/algebraic-temporal-logics.

5.1   Formalisation Aspects

Our formalisation work consists in the mechanisation of all the aspects presented in the paper: we start by defining the notion 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 Sect. 4.2 for the functional setting.

For the sake of presentation in this paper we restricted our attention to graph signatures, and a concrete example in Agda of our library instantiated on the signature of directed graphs is available in [18]. However, we remark that our implementation is general enough to model algebras over any generic multisorted signature. In particular, this means that, by specifying a suitable 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 the correctness and completeness of PNF with respect to QLTL also yields a concrete procedure that can convert formulae into their positive normal form version. 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. Using 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 constructed 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 as we have seen in Sect. 4 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 of worlds is translated into a morphism between the algebras which the user provides. 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 defined using a coinductive definition of traces using sized types [11], thus 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.

5.2   Intuitionistic Proof Assistant

In our setting, some crucial usability issues need to be mentioned. Agda is a proof assistant based on the *intuitionistic* interpretation of mathematics [24]. This means that some useful logical principles often used in the setting of temporal logics are *not provable* in the system, such as the law of excluded middle or the De Morgan laws to switch connectives and quantifiers whenever negation appears in subformulae. Thus, without assuming these logical principles, the embedding of our temporal logic QLTL would actually be restricted to the *intuitionistic*

fragment; in practice, this is not particularly problematic since classical reasoning can simply be assumed as axiom, and allows the equivalences 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. In order to tackle these usability aspects 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 Sect. 4, which we have proven correct in Agda in Theorem 4.3 by postulating classical principles. This effectively shifts the burden of dealing (classically) with negation from the user to the implementer, while also giving them complete accessibility over the extended set of correct quantifiers. Moreover, the correctness proof of the conversion to PNF constitutes both a theoretical guarantee that no expressive power is either gained or lost in the presentation of the logic, as well as a concrete algorithm that the user can execute to convert their formulae into equivalent ones in PNF, for which validity can be easier to prove.

5.3   Automation

Embedding a temporal logic in a proof assistant allows the user to exploit the *assistant* aspect of the tool, for example, by aiding the user in showing (or even prove 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 [6], Lean [34], and Isabelle [36]. The Agda synthesizer Agsy [32] 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.

6   Related Works

Up to the early 2010s,s, there has been a series of papers devoted to some variants 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 fixpoints in [16], and we refer there for an overview of and a comparison with the by-then current proposals, such as the well-known [1], all favouring an approach based on universal domains. Among the follow-ups of the works mentioned there, there is [19] and the related [44,47], which further explore one of the relevant tools developed in the graph community, GROOVE. To some extent, the present paper and its companion [17], which introduces the categorical semantics of second-order QLTL, are summarising a previous thread of research concerning counterpart models, including its implementation. And in

fact, the categorical semantics for counterpart models appears of interest in itself in the literature on modal logics, as witnessed by the works surveyed in [17].

Concerning the formalisation of temporal logics in (constructive) proof assistants, the topic has a long history, see e.g. [8,45,48]. A practical application and comparison with modern model checkers is in [14], and a fully verified LTL model checker is implemented in the Isabelle theorem prover. In [38], a verified proof-search program is formalised in Agda for standard CTL, together with a toolbox to implement well-typed proof-searching procedures; a similar embedding of constructive LTL in Agda is provided in [29] for the verification of functional reactive programs. Our proof-of-concept implementation of QLTL in Agda witnesses the possibility to move towards the formalisation of quantified temporal logics for proof assistants, an issue sparsely tackled in the literature.

Concerning graph computation models (GCMs), we find in the literature several formalisms that use graph-specific definitions where syntactical statements on nodes, edges, sources of edges, targets of edges, and equalities are first-class citizens in the logic to express properties on the system under analysis. The last decade has seen a series of papers advocating quantified temporal logics as a formalism for the specification of GCMs properties. We offer a short review of some of the most recent proposals appeared in the literature, 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 [39,46], 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 [4,33], 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) [40,41], investigated with respect to the expressiveness of suitable fragments in [28] or to duality results akin to our PNF in [35]. These logics allows 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 using counterpart relations.

Metric Logics, II. A graph-oriented approach to MFOTL is given by Metric Temporal Graph Logic (MTGL) [23,43], which allows to model properties on the structures and the attributes of the state and has been used in the context of formal testing [42]. 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. Thus, besides our use of relations, 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.

7   Conclusions and Future Works

We have seen how a set-theoretic semantics for a first-order linear-time temporal logic QLTL can be presented in the counterpart setting. We saw how its syntax and semantics can be naturally used in an algebraic setting to express properties of direct graphs and their evolution in time, and how the notions and models presented in the previous sections 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.

   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 [16,17] 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 alternative constructs and models, such as those of CTL [13]. Extending our logic to more complex models seems a straightforward task, which might however cause a combinatorial explosion in the temporal operators required to obtained a positive normal form.

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 [14,30,38]. The second mechanism consists in writing bindings to external programs, such as 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.

Finite Traces. A current trend in artificial intelligence is the study of temporal formulas over *finite* traces [22], 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.

References

1. Baldan, P., Corradini, A., König, B., Lluch Lafuente, A.: A temporal graph logic for verification of graph transformation systems. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) WADT 2006. LNCS, vol. 4409, pp. 1–20. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71998-4_1

2. Belardinelli, F.: Quantified Modal Logic and the Ontology of Physical Objects. Ph.D. work, Scuola Normale Superiore of Pisa (2004–2005)

3. Blackburn, P., van Benthem, J., Wolter, F. (eds.): Handbook of Modal Logic, vol. 3. North Holland (2007)

4. Búr, M., Marussy, K., Meyer, B.H., Varró, D.: Worst-case execution time calculation for query-based monitors by witness generation. ACM Trans. Embed. Comput. Syst. **20**(6), 1–36 (2021)

5. Bustan, D., Flaisher, A., Grumberg, O., Kupferman, O., Vardi, M.Y.: Regular vacuity. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 191–206. Springer, Heidelberg (2005). https://doi.org/10.1007/11560548_16

6. Coq Development Team: The Coq Proof Assistant Reference Manual (2016)

7. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer (2006)

8. Coupet-Grimal, S.: An axiomatization of linear temporal logic in the calculus of inductive constructions. J. Logic Comput. **13**(6), 801–813 (2003)

9. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inform. Comput. **85**(1), 12–75 (1990)

10. Courcelle, B.: The monadic second-order logic of graphs. XII. Planar graphs and planar maps. Theor. Comput. Sci. **237**(1), 1–32 (2000)

11. Danielsson, N.A.: Up-to techniques using sized types. In: POPL 2018, pp. 43:1–43:28. ACM (2018)

12. Dawar, A., Gardner, P., Ghelli, G.: Expressiveness and complexity of graph logic. Inf. Comput. **205**(3), 263–310 (2007)

13. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 995–1072. Elsevier and MIT Press (1990)

14. Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.-G.: A fully verified executable LTL model checker. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 463–478. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_31

15. Franconi, E., Toman, D.: Fixpoint extensions of temporal description logics. In: Calcanese, D., De Giacomo, G., Franconi, E. (eds.) DL 2003. CEUR Workshop Proceedings, vol. 81 (2003)

16. Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Counterpart semantics for a second-order μ-calculus. Fundamenta Informaticae **118**(1–2), 177–205 (2012)

17. Gadducci, F., Trotta, D.: A presheaf semantics for quantified temporal logics. CoRR abs/2111.03855 (2021)
18. Gadducci, F., Laretto, A., Trotta, D.: Specification and verification of a linear-time temporal logic for graph transformation. CoRR abs/2305.03832 (2023)
19. Ghamarian, A.H., de Mol, M., Rensink, A., Zambon, E., Zimakova, M.: Modelling and analysis using GROOVE. Int. J. Softw. Tools Technol. Trans. **14**(1), 15–40 (2012)
20. Ghilardi, S., Meloni, G.: Modal and tense predicate logic: models in presheaves and categorical conceptualization. In: Borceux, F. (ed.) Categorical Algebra and its Applications. LNM, vol. 1348, pp. 130–142. Springer (1988)
21. Ghilardi, S., Meloni, G.: Relational and partial variable sets and basic predicate logic. J. Symbol. Logic **61**(3), 843–872 (1996)
22. Giacomo, G.D., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: Yang, Q., Wooldridge, M.J. (eds.) IJCAI 2015, pp. 1558–1564. AAAI Press (2015)
23. Giese, H., Maximova, M., Sakizloglou, L., Schneider, S.: Metric temporal graph logic over typed attributed graphs. In: Hähnle, R., van der Aalst, W. (eds.) FASE 2019. LNCS, vol. 11424, pp. 282–298. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16722-6_16
24. Girard, J., Lafont, Y., Taylor, P.: Proofs and Types, Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press (1989)
25. Hazen, A.: Counterpart-theoretic semantics for modal logic. J. Philos. **76**(6), 319–338 (1979)
26. Hodkinson, I., Wolter, F., Zakharyaschev, M.: Monodic fragments of first-order temporal logics: 2000–2001 A.D. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 1–23. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45653-8_1
27. Huang, S., Cleaveland, R.: A tableau construction for finite linear-time temporal logic. J. Logic Algebr. Meth. Program. **125**, 100743 (2022)
28. Hublet, F., Basin, D., Krstić, S.: Real-time policy enforcement with metric first-order temporal logic. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds.) ESORICS 2022. LNCS, vol. 13555, pp. 211–232. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17146-8_11
29. Jeffrey, A.: LTL types FRP: linear-time temporal logic propositions as types, proofs as functional reactive programs. In: Claessen, K., Swamy, N. (eds.) PLPV 2012, pp. 49–60. ACM (2012)
30. Kokke, P., Swierstra, W.: Auto in Agda. In: Hinze, R., Voigtländer, J. (eds.) MPC 2015. LNCS, vol. 9129, pp. 276–301. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19797-5_14
31. Lewis, D.K.: Counterpart theory and quantified modal logic. J. Philos. **65**(5), 113–126 (1968)
32. Lindblad, F., Benke, M.: A tool for automated theorem proving in Agda. In: Filliâtre, J.-C., Paulin-Mohring, C., Werner, B. (eds.) TYPES 2004. LNCS, vol. 3839, pp. 154–169. Springer, Heidelberg (2006). https://doi.org/10.1007/11617990_10
33. Marussy, K., Semeráth, O., Babikian, A.A., Varró, D.: A specification language for consistent model generation based on partial models. J. Object Technol. **19**(3), 1–22 (2020)
34. Moura, L., Ullrich, S.: The lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) CADE 2021. LNCS (LNAI), vol. 12699, pp. 625–635. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_37

35. Huerta, Y., Munive, J.J.: Relaxing safety for metric first-order temporal logic via dynamic free variables. In: Dang, T., Stolz, V. (eds.) RV 2022. LNCS, vol. 13498, pp. 45–66. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17196-3_3

36. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283, pp. 67–104. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9_5

37. Norell, U.: Dependently typed programming in Agda. In: Kennedy, A., Ahmed, A. (eds.) TLDI 2009, pp. 1–2. ACM (2009)

38. O'Connor, L.: Applications of applicative proof search. In: Chapman, J., Swierstra, W. (eds.) TyDe@ICFP 2016, pp. 43–55. ACM (2016)

39. Poskitt, C.M., Plump, D.: Monadic second-order incorrectness logic for GP 2. J. Logic Algebr. Meth. Program. **130**, 100825 (2023)

40. Schneider, J., Basin, D., Krstić, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 310–328. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_18

41. Schneider, J., Traytel, D.: Formalization of a monitoring algorithm for metric first-order temporal logic. Archive of Formal Proofs (2019)

42. Schneider, S., Maximova, M., Sakizloglou, L., Giese, H.: Formal testing of timed graph transformation systems using metric temporal graph logic. Int. J. Softw. Tools Technol. Transf. **23**(3), 411–488 (2021). https://doi.org/10.1007/s10009-020-00585-w

43. Schneider, S., Sakizloglou, L., Maximova, M., Giese, H.: Optimistic and pessimistic on-the-fly analysis for metric temporal graph logic. In: Gadducci, F., Kehrer, T. (eds.) ICGT 2020. LNCS, vol. 12150, pp. 276–294. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51372-6_16

44. Smid, W., Rensink, A.: Class diagram restructuring with GROOVE. In: Gorp, P.V., Rose, L.M., Krause, C. (eds.) TTC 2013. EPTCS, vol. 135, pp. 83–87 (2013)

45. Sprenger, C.: A verified model checker for the modal μ-calculus in Coq. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 167–183. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054171

46. Wulandari, G.S., Plump, D.: Verifying graph programs with monadic second-order logic. In: Gadducci, F., Kehrer, T. (eds.) ICGT 2021. LNCS, vol. 12741, pp. 240–261. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78946-6_13

47. Zambon, E., Rensink, A.: Recipes for coffee: Compositional construction of JAVA control flow graphs in GROOVE. In: Müller, P., Schaefer, I. (eds.) Principled Software Development. LNCS, pp. 305–323. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98047-8_19

48. Zanarini, D., Luna, C., Sierra, L.: Alternating-time temporal logic in the calculus of (Co)inductive constructions. In: Gheyi, R., Naumann, D. (eds.) SBMF 2012. LNCS, vol. 7498, pp. 210–225. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33296-8_16