



University of Pisa
Computer Science Department
Master in Computer Science

Counterpart Semantics for Quantified Temporal Logics: Sets, Categories and Agda

Supervisor:
Fabio Gadducci
Co-supervisor:
Davide Trotta

Candidate:
Andrea Laretto

Examiner:
Roberto Bruni

Academic Year 2021/2022

Abstract

The aim of this thesis is to provide a complete presentation of the semantics of first-order temporal logics based on the counterpart paradigm: we start by introducing a standard set-based semantics and then we define a categorical semantics based on relational presheaves. The constructions are formalized using the dependently typed programming language and proof assistant Agda, and we employ the [agda-categories](#) library to adequately capture the notions of category theory required in our setting.

Contents

1	Introduction	1
1.1	Quantified temporal logics	2
1.2	Counterpart semantics	2
1.3	Contributions	3
1.4	Synopsis	5
2	Quantified Temporal Logics	6
2.1	Counterpart semantics	6
2.1.1	Temporal structures	9
2.2	Quantified linear temporal logic	9
2.2.1	Syntax and semantics of QLTL	10
2.2.2	Contexts and assignments	11
2.2.3	Satisfiability	11
2.3	Positive normal form for QLTL	15
2.3.1	Semantics of PNF	15
2.3.2	Negation of QLTL and PNF	17
3	Categorical semantics	22
3.1	Relational presheaves models	22
3.2	Temporal structures	24
3.3	Presheaf semantics for QLTL	27
3.3.1	Classical attributes	27
3.3.2	Semantics with classical attributes	28
3.3.3	Semantics of QLTL	29
3.4	Multi-sorted algebra models	32
3.5	Algebraic counterpart \mathcal{W} -models	34
3.6	Semantics of algebraic QLTL	36
3.7	Examples	38

4	Agda formalization	41
4.1	Formalization aspects	41
4.1.1	Automation	44
4.1.2	Category theory	44
4.2	Formalization	44
4.3	Signatures and algebras	45
4.4	Terms	47
4.5	Relational presheaves	48
4.6	Counterpart models	50
4.7	Algebraic counterpart \mathcal{W} -model	51
4.8	Temporal structure	53
4.9	From classical to categorical models	54
4.10	Classical attributes	57
4.11	Syntax and semantics of QLTL	59
4.12	Examples	62
5	Conclusion	74
5.1	Related work	74
5.2	Future work	75
A	Category Theory	81

Chapter 1

Introduction

During the construction of hardware and software design of complex systems, increasingly more time and effort is spent on the verification of the desired mechanisms rather than 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 provided in the field of formal methods, temporal logics have proven to be one of the most well-established and effective techniques for the verification of both large-scale and stand-alone programs, see for example [RS85; CGG02] among many others. After the foundational work by Pnueli [Pnu77], the research on temporal logics focused on both algorithmic procedures for the verification of properties as well as 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 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 of these models is that of graph logics [Cou90; Cou00; DGG07], where states are specialized as graphs and transitions are families of (partial) graph morphisms. These logics combine temporal and spatial reasoning and allow, for example, to express the possible transformation of a graph topology over time.

1.1 Quantified temporal logics

Under usual temporal logics, such as LTL and CTL [Eme90], 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 these 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 [FT03; HWZ01], 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 $s_0 \rightarrow s_1$ and $s_1 \rightarrow s_0$, and an item i that appears only in s_0 . Is the item i being destroyed and recreated 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* [Haz79; Bel04]. The typical solution provided by the so-called "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.

1.2 Counterpart semantics

A solution to this problem was proposed by Lewis [Lew68] 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 worlds.

In [GLV12], 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 generalized to a categorical setting in [GT21] by means of relational presheaves, building on the ideas presented in [GM88; GM96]. 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 notion of selected *one-step* arrows of the model called *temporal structure*, and the categorical framework is then used to introduce a second-order linear temporal logic QLTL.

1.3 Contributions

A first contribution of this thesis is to provide a comprehensive presentation and introduction to the setting of counterpart models and quantified temporal logics, as they are presented in [GLV12; GT21].

Classical semantics and positive normal form

We start in Chapter 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 [GLV12; GT21] where the models and semantics we are defined using *partial functions*, we generalize our case to the setting of *relations*, thus modeling the duplication of elements and allowing for an element to have multiple counterparts in the next world. We conclude the chapter by presenting some results and equivalences on the positive normal form of this logic, considering both the cases where the models use partial morphisms and relations and highlighting their differences. Both the classical semantics and the positive normal form results have been formalized using the dependently typed proof assistant Agda [Nor09].

Categorical semantics in Agda

In Chapter 3 we introduce the categorical semantics for QLTL as given in [GT21], first by motivating the use of the categorical formalism and relational presheaves with a standard non-algebraic case. In this thesis we only concentrate on the first-order fragment of the logic. We then present the semantics of the logic using the notion of classical attribute [GM88; GM96], following the intuition that the meaning of a formula is identified with the set of individuals that satisfy it in each

world. Finally, we introduce the additional mechanisms and definitions required to extend the categorical presentation and its semantics to the more general case of states as algebras and relational homomorphisms between them.

The central contribution presented in this thesis is a computer-assisted formalization in Agda of the categorical notions and constructions given by [GT21].

We introduce the main aspects of the formalization and its definitions in [Chapter 4](#). Providing a mechanized presentation of these constructions has several advantages:

- Formalizing 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 formalization, 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 mechanization aspects is absent in the literature. This thesis 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 formalization work employs the library [agda-categories](#) [HC21], a proof-relevant category theory library for Agda, as a practical foundation to formalize the results presented by [GT21] in the context of temporal logics and their models. Aside from the theoretical results and constructions provided by the library in the field of category theory, our work also establishes the usefulness and flexibility of [agda-categories](#) from the point of view of practical applications.

1.4 Synopsis

We briefly summarize the structure of this thesis:

- [Chapter 1](#): we start with a brief introduction to the context and main references on which this thesis builds upon, along with a general description of the contents and contributions presented in this work.
- [Chapter 2](#): we give an introduction to quantified temporal logics and counterpart semantics, presenting the central notions of our linear quantified temporal logic QLTL as well as some results on the positive normal forms of this logic.
- [Chapter 3](#): the categorical presentation of the models and the semantics of our logic are given by means of relational presheaves. The categorical models are then extended to the case of algebras-as-worlds.
- [Chapter 4](#): the concepts presented in the previous chapters are introduced using Agda, describing some general aspects of the formalization and highlighting its core constructions step-by-step.
- [Chapter 5](#): we briefly summarize the work presented and discuss related works along with possible extensions of this thesis.
- [Appendix A](#): we recall some basic notions about category theory to establish the theoretical setting required in the categorical presentation of the logics.

Chapter 2

Quantified Temporal Logics

In this chapter we introduce the counterpart paradigm and define the class of models used by the quantified temporal logics later introduced. 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 formalization files in each definition. The formalization of the set-based quantified temporal logic and the positive normal form results is available at:

<https://github.com/iwilare/qltl-pnf>

2.1 Counterpart semantics

We start by recalling the notion of *Kripke frame* as widely known in modal logics [Bel04; BBW07] and extend it for the case of counterpart semantics.

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

- W is a non-empty set;
- R is a binary relation on W ;
- D is a function assigning to every element $\omega \in W$ a set $D(\omega)$ called *domain*.

The set W is intuitively interpreted as the collection 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. Each *domain* $D(\omega)$ identifies the set of objects that exist in the world ω .

A crucial development in the presentation of Kripke models was introduced by Lewis [Lew68] 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 employ the notion of *counterpart relation* between worlds to connect the individuals that are being preserved from one world to the next one.

Inspired by Lewis's approach, a more general notion of counterpart model is considered in [GLV12], 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 $\langle W, D, \mathcal{C} \rangle$ defined as

- W and D are defined as for Kripke frames;
- \mathcal{C} is a function assigning to every 2-tuple $\langle \omega, \omega' \rangle$ a set of relations $\mathcal{C}\langle \omega, \omega' \rangle \in \mathcal{P}(\mathcal{P}(D(\omega) \times D(\omega')))$, where \mathcal{P} denotes the powerset, and every element $C \in \mathcal{C}\langle \omega, \omega' \rangle$ 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 $\mathcal{C}\langle \omega, \omega' \rangle$ 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 counterpart relation $C \in \mathcal{C}\langle \omega, \omega' \rangle$ connects the individuals between two given worlds ω and ω' , intuitively identifying them as the same element after one time evolution of the model. In particular, if we consider two elements $s \in D(\omega)$ and $s' \in D(\omega')$ and a relation $C \in \mathcal{C}\langle \omega, \omega' \rangle$, if $\langle s, s' \rangle \in C$ then s' represents a future development of s via C .

The use of relations as counterpart transitions 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 thesis. 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 $C_1; C_2 = \{(a, c) \mid \exists b. \langle a, b \rangle \in C_1 \wedge \langle b, c \rangle \in C_2\} \subseteq A \times C$.

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 can be obtained by composing together a suitable sequence of relations $C_0; C_1; \dots; C_n$ with $C_i \in \mathcal{C}\langle\omega_i, \omega_{i+1}\rangle$.

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. This intuitively represents the fact that transitioning through an intermediate state and transitioning directly between worlds can be regarded as two different possibilities, with the model defining only the direct one-step transitions.

Definition 2.4. We say that an individual $s' \in D(\omega')$ 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 [Figure 2.1](#) 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: $\mathcal{C}\langle\omega_0, \omega_1\rangle := \{C_0\}$ 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 dash-dotted](#) lines to distinguish C_1 and C_2 , respectively.

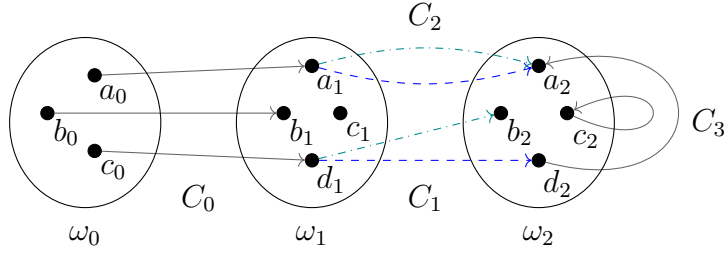


Figure 2.1: An example of counterpart model.

2.1.1 Temporal structures

As is the case of LTL where we can identify traces connecting linearly evolving states, see for example [BBW07], 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 $C_i \in \mathcal{C}(\omega_i, \omega_{i+1})$ 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 $C_0; \dots; C_{i-1}$ 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 (first-order) quantified linear temporal logic QLTL. We will assume hereafter 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

In order to give 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; for this reason, we will 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 and $x, y \in \mathcal{X}$. The set $\mathcal{F}^{\text{QLTL}}$ of QLTL formulae is generated by the following rules:

$$\psi := \text{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists x.\phi \mid \text{O}\psi \mid \phi_1 \text{U}\phi_2 \mid \phi_1 \text{W}\phi_2.$$

The *next* operator $\text{O}\phi$ expresses the fact that a certain property ϕ has to be true at the next state. The *until* operator $\phi_1 \text{U}\phi_2$ 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 $\phi_1 \text{W}\phi_2$ is similar to the $\phi_1 \text{U}\phi_2$ 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 a unary predicate symbol $P(x)$ that will be used in the running example in [Figure 2.2](#). The usual dual operators can be syntactically expressed by taking $\text{false} := \neg\text{true}$, $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$, and $\forall x.\phi := \neg\exists x.\neg\phi$.

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

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

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

$$\text{nextStepPreserved}(x) := \text{present}(x) \wedge \text{Opresent}(x).$$

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

$$\text{nextStepDeallocated}(x) := \text{present}(x) \wedge \neg\text{Opresent}(x).$$

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 [GT21; GLV12].

Definition 2.7 (Context). A **context** Γ over a set of variables \mathcal{X} is a 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 in which it is defined, and we indicate this decoration with $[\Gamma]\phi$. Any context is such that $\text{fv}(\phi) \subseteq \Gamma$, i.e., it contains the free variables of the formula ϕ .

We omit the bracketed context whenever it is unnecessary to specify it. To properly present the notion of satisfiability of a formula 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 $\mu : \Gamma \rightarrow D(\omega)$. We use the notation $\mathcal{A}_\omega^\Gamma$ to indicate the set of assignments μ defined in ω for the context Γ .

Moreover, we denote by $\mu[x \mapsto s] : \Gamma, x \rightarrow D(\omega)$ 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 behind this notion 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 $\mu_1 : \Gamma \rightarrow D(\omega_1)$ and $\mu_2 : \Gamma \rightarrow D(\omega_2)$ 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 in a given trace and assignment.

Definition 2.11 (QLTL satisfiability). Given a QLTL formula-in-context $[I]\phi$, a trace $\sigma = (C_0, C_1, \dots)$, and an assignment $\mu : \Gamma \rightarrow D(\omega_0)$ in the first world of σ , we inductively define the *satisfiability relation* as follows:

- $\sigma, \mu \models \text{true}$;
- $\sigma, \mu \models x = y$ iff $\mu(x) = \mu(y)$;
- $\sigma, \mu \models P(x)$ iff $P(\mu(x))$;
- $\sigma, \mu \models \neg\phi$ iff $\sigma, \mu \not\models \phi$;
- $\sigma, \mu \models \phi_1 \vee \phi_2$ iff $\sigma, \mu \models \phi_1$ or $\sigma, \mu \models \phi_2$;
- $\sigma, \mu \models \exists x.\phi$ iff there exists an individual $s \in D(\omega_0)$ such that $\sigma, \mu[x \mapsto s] \models \phi$;
- $\sigma, \mu \models \text{O}\phi$ iff there exists $\mu_1 \in \mathcal{A}_{\omega_1}^\Gamma$ such that $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \models \phi$;
- $\sigma, \mu \models \phi_1 \text{U} \phi_2$ iff there exists an $\bar{n} \geq 0$ such that:
 1. for any $i < \bar{n}$, there exists $\mu_i \in \mathcal{A}_{\omega_i}^\Gamma$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \models \phi_1$;
 2. there exists $\mu_{\bar{n}} \in \mathcal{A}_{\omega_{\bar{n}}}^\Gamma$ such that $\langle \mu, \mu_{\bar{n}} \rangle \in C_{\leq \bar{n}}$ and $\sigma_{\bar{n}}, \mu_{\bar{n}} \models \phi_2$;
- $\sigma, \mu \models \phi_1 \text{W} \phi_2$ iff one of the following holds:
 - the same conditions for $\phi_1 \text{U} \phi_2$ apply;
 - for any i there exists $\mu_i \in \mathcal{A}_{\omega_i}^\Gamma$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ and $\sigma_i, \mu_i \models \phi_1$.

Example 2.3. We present a running example in Figure 2.2 to better describe the expressiveness of QLTL and to illustrate the mechanisms of working in a counterpart-based semantics.

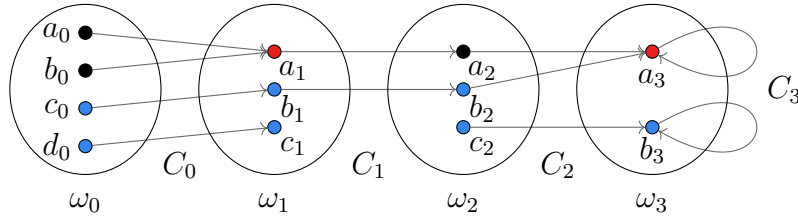


Figure 2.2: An example with four worlds $\omega_0, \omega_1, \omega_2, \omega_3$

We consider a fixed trace $\sigma = (C_0, C_1, C_2, C_3, C_3, \dots)$ and we indicate with $\mathbf{B}(x)$ and $\mathbf{R}(x)$ the unary predicates that hold for any individual colored in **blue** and **red**, respectively. As a concrete scenario for the temporal operators $\mathbf{O}\phi$ and $\phi_1 \mathbf{U} \phi_2$ we presented in Definition 2.11, we have for example that $\sigma, \{x \mapsto a_0\} \models \mathbf{O}(\mathbf{R}(x))$, and $\sigma, \{x \mapsto c_0\} \models \mathbf{B}(x) \mathbf{U} \mathbf{R}(x)$. Also, we have that a_0 is preserved at the next step with $\sigma, \{x \mapsto a_0\} \models \mathbf{nextStepPreserved}(x)$, whereas c_1 is removed and indeed $\sigma_1, \{x \mapsto c_1\} \models \mathbf{nextStepDeallocated}(x)$.

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

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

In our example in Figure 2.2, we have for instance that $\sigma, \{x \mapsto c_0\} \models \Diamond\mathbf{R}(x)$ but $\sigma, \{x \mapsto d_0\} \not\models \Diamond\mathbf{R}(x)$ and similarly $\sigma_2, \{x \mapsto c_2\} \not\models \Diamond\mathbf{R}(x)$. Moreover, we have that $\sigma, \{x \mapsto c_0\} \models \Diamond\Box\mathbf{R}(x)$ and $\sigma_2, \{x \mapsto c_2\} \models \Box\mathbf{B}(x)$. However, $\sigma, \{x \mapsto d_0\} \not\models \Box\mathbf{B}(x)$ 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 running example in Figure 2.2, we have that in the first world $\sigma, \{x \mapsto a_0, y \mapsto c_0\} \models \mathbf{willMerge}(x, y)$, but clearly $\sigma, \{x \mapsto c_0, y \mapsto d_0\} \not\models \mathbf{willMerge}(x, y)$.

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 variable introduced does not appear in the subformula: taking any ϕ with $x \notin \text{fv}(\phi)$ 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.

We describe here a concrete example: consider a world ω with a single individual $D(\omega) = \{s\}$ and a 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 $\phi = O(\text{true})$, one can easily check that $\sigma, \{\} \models O(\text{true})$, but $\sigma, \{\} \not\models \exists x.O(\text{true})$. 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 transported 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* $O\phi$ operator becomes:

- $\sigma, \mu \models O\phi$ iff there exists $\mu_1 \in A_{\omega_1}^{\text{fv}(\phi)}$ such that $\langle \mu|_{\text{fv}(\phi)}, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \models \phi$.

For ease of presentation both in this thesis 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 does not naturally align with the categorical semantics presented in [Chapter 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* $O\phi$ operator in our counterpart-style semantics in general is not self-dual with respect to negation, i.e.: $\neg O\phi \not\equiv O\neg\phi$. 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 [Figure 2.2](#): it is easy to see that $\sigma_1, \{x \mapsto c_1\} \models \neg O(\mathbf{B}(x))$, but $\sigma_1, \{x \mapsto c_1\} \not\models O(\neg \mathbf{B}(x))$ since no counterpart for c_1 exists after one step.

The idea is that, since the *next* operator requires that 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* $\phi_1 U \phi_2$ and *weak until* $\phi_1 W \phi_2$ operators have the same expressivity, and can be defined in terms of each other by the equivalences

$$\begin{aligned}\phi_1 U \phi_2 &\equiv_{\text{LTL}} \neg(\neg\phi_2 W (\neg\phi_1 \wedge \neg\phi_2)), \\ \phi_1 W \phi_2 &\equiv_{\text{LTL}} \neg(\neg\phi_2 U (\neg\phi_1 \wedge \neg\phi_2)).\end{aligned}$$

However, this is *not* the case in QLTL. Similarly, it seems reasonable to define the standard *always* and *eventually* operators in QLTL with $\Box\phi := \neg\Diamond\neg\phi$ and $\Diamond\phi := \neg\Box\neg\phi$, respectively; however, this definition does not align with 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 presented in LTL can be obtained again 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 more similar notion of LTL trace.

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 implementative side [[Bus+05](#); [HC22](#)]. This presentation is crucial to define the semantics of a logic based on fixpoints, such as the one given in [[GLV12](#)], while still preserving the full expressiveness of the original presentation. As we will remark in [Chapter 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* $A\phi$. Similarly, we have to introduce a negative dual for the *until* $\phi_1 U \phi_2$ and *weak until* $\phi_1 W \phi_2$ operators, which we indicate as the *then* $\phi_1 T \phi_2$ and *until-forall* $\phi_1 F \phi_2$ operators, respectively.

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

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

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 2.13 (QLTL in PNF satisfiability). We define the following additional constructs:

- $\sigma, \mu \models \neg\psi$ iff $\sigma, \mu \not\models \psi$;
- $\sigma, \mu \models \phi_1 \wedge \phi_2$ iff $\sigma, \mu \models \phi_1$ and $\sigma, \mu \models \phi_2$;
- $\sigma, \mu \models \forall x.\phi$ iff for every individual $s \in D(\omega_0)$ we have that $\sigma, \mu[x \mapsto s] \models \phi$;
- $\sigma, \mu \models A\phi$ iff for any $\mu_1 \in \mathcal{A}_{\omega_1}^\Gamma$ such that $\langle \mu, \mu_1 \rangle \in C_0$ we have that $\sigma_1, \mu_1 \models \phi$;
- $\sigma, \mu \models \phi_1 F\phi_2$ iff there exists an $\bar{n} \geq 0$ such that:
 1. for any $i < \bar{n}$, for any $\mu_i \in \mathcal{A}_{\omega_i}^\Gamma$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have $\sigma_i, \mu_i \models \phi_1$;
 2. for any $\mu_{\bar{n}} \in \mathcal{A}_{\omega_{\bar{n}}}^\Gamma$ such that $\langle \mu, \mu_{\bar{n}} \rangle \in C_{\leq \bar{n}}$ we have that $\sigma_{\bar{n}}, \mu_{\bar{n}} \models \phi_2$;
- $\sigma, \mu \models \phi_1 T\phi_2$ iff one of the following holds:
 - the same conditions for $\phi_1 F\phi_2$ apply;
 - for any i and any $\mu_i \in \mathcal{A}_{\omega_i}^\Gamma$ such that $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \models \phi_1$.

The intuition for the *next-forall* $A\phi$ 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* $\phi_1 U\phi_2$ operator, the *until-forall* $\phi_1 F\phi_2$ 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 $\phi_1 F\phi_2$ and $\phi_1 T\phi_2$.

Similarly to the *weak until* $\phi_1 W\phi_2$ operator, the *then* $\phi_1 U\phi_2$ 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 [Figure 2.2](#), we illustrate the possibility for $B(x)R(x)$ and $AB(x)$ to be satisfied even when a counterpart does not exist after one or more steps. In particular, it can

be verified that $\sigma, \{x \mapsto c_0\} \models \mathbf{B}(x)\mathbf{FR}(x)$ holds since $\mathbf{R}(x)$ is eventually satisfied while $\mathbf{B}(x)$ holds, just like the *until* operator. We have that both $\sigma, \{x \mapsto a_0\} \models \mathbf{AR}(x)$ and $\sigma_1, \{x \mapsto c_1\} \models \mathbf{AR}(x)$ hold, since no counterpart for c_1 exists after one step. Finally, we have that $\sigma, \{x \mapsto d_0\} \models \mathbf{B}(x)\mathbf{FR}(x)$ holds since $\mathbf{B}(x)$ holds but no counterpart exists after two steps, and $\sigma_2, \{x \mapsto c_2\} \models \mathbf{B}(x)\mathbf{TR}(x)$ since a counterpart always exists but $\mathbf{B}(x)$ 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* $\mathbf{O}\phi$, *until* $\phi_1 \mathbf{U}\phi_2$, and *weak until* $\phi_1 \mathbf{W}\phi_2$ formulae can now be expressed inside the logic. We will explicitly indicate with \models_{QLTL} and \models_{PNF} the satisfiability relations defined for formulae in standard QLTL and QLTL in PNF, respectively.

Theorem 2.1 (Negation is expressible in PNF). ([Relational.Negation](#))

Let ψ be an atomic formula in PNF. Then we have

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

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* $\mathbf{A}\phi$ operator, whose negation directly corresponds with the *next* $\mathbf{O}\phi$ operator.

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

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

Proof. We provide a single direct counterexample for both the *then* and *until-forall* cases. Take for example the formula $\mathbf{B}(x)\mathbf{TR}(x)$; we consider the counterexample in [Figure 2.3](#):

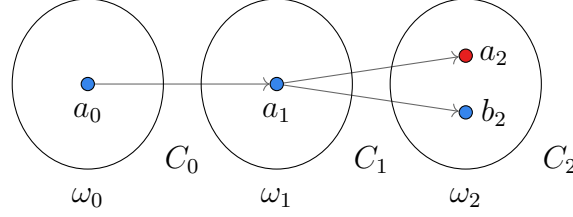


Figure 2.3: A counterexample model where, in the case of counterpart relations, we have that $\neg(\mathbf{B}(x)\mathbf{TR}(x)) \not\equiv (\neg\mathbf{R}(x))\mathbf{U}(\neg\mathbf{B}(x) \wedge \neg\mathbf{R}(x))$.

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

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 [GT21; GLV12].

Theorem 2.3 (Negation for partial functions). (\mathcal{U} **Functional.Negation**)


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

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

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

The equivalences presented in Theorem 2.1 allow us to define a formal translation $\overline{\cdot} : \mathcal{F}^{\text{QLTL}} \rightarrow \mathcal{F}^{\text{PNF}}$ from the QLTL syntax presented in Definition 2.11 to the current one in PNF, preserving 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:

$$\begin{aligned} \overline{\mathbf{O}\phi} &:= \mathbf{O}\overline{\phi} & \overline{\phi_1\mathbf{U}\phi_2} &:= \overline{\phi_1}\mathbf{U}\overline{\phi_2} \\ \overline{\neg\mathbf{O}\phi} &:= \mathbf{A}\neg\overline{\phi} & \overline{\phi_1\mathbf{W}\phi_2} &:= \overline{\phi_1}\mathbf{W}\overline{\phi_2} \\ \overline{\phi_1 \vee \phi_2} &:= \overline{\phi_1} \vee \overline{\phi_2} & \overline{\neg(\phi_1\mathbf{U}\phi_2)} &:= (\neg\overline{\phi_2})\mathbf{T}(\neg\overline{\phi_1} \wedge \neg\overline{\phi_2}) \\ \overline{\neg(\phi_1 \vee \phi_2)} &:= \neg\overline{\phi_1} \wedge \neg\overline{\phi_2} & \overline{\neg(\phi_1\mathbf{W}\phi_2)} &:= (\neg\overline{\phi_2})\mathbf{F}(\neg\overline{\phi_1} \wedge \neg\overline{\phi_2}) \end{aligned}$$


Theorem 2.4 (PNF equivalence). ( **Relational.Conversion**) Let $\overline{\cdot} : \mathcal{F}^{\text{QLTL}} \rightarrow \mathcal{F}^{\text{PNF}}$ be the aforementioned syntactical translation that replaces negated temporal operators with their equivalent ones in PNF. For any QLTL formula $[I]\phi \in \mathcal{F}^{\text{QLTL}}$, we can express the following result:

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

Now that we have defined the complete set of the 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 $\Diamond^* \phi := \text{trueF}\phi$ and $\Box^* \phi := \phi \mathsf{T} \text{false}$, respectively. Their semantics can be explicitly presented as follows:

- $\sigma, \mu \models \Diamond^* \phi$ iff there exists $i \geq 0$ such that for any $\mu_i \in A_{w_i}^I$ with $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \models \phi$.
- $\sigma, \mu \models \Box^* \phi$ iff for every i and any $\mu_i \in A_{w_i}^I$ with $\langle \mu, \mu_i \rangle \in C_{\leq i}$ we have that $\sigma_i, \mu_i \models \phi$;

Theorem 2.5 (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:

Theorem 2.6 (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* $\phi_1 \mathbf{U} \phi_2$ and *until-forall* $\phi_1 \mathbf{F} \phi_2$ cases, with the *weak until* $\phi_1 \mathbf{W} \phi_2$ and *then* $\phi_1 \mathbf{T} \phi_2$ cases obviously following.

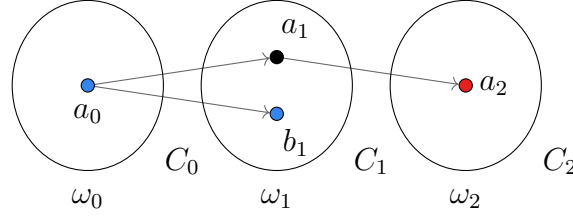


Figure 2.4: A counterexample model where, in the case of counterpart relations, we have that $\mathbf{B}(x)\mathbf{U}\mathbf{R}(x) \not\equiv \mathbf{R}(x) \vee (\mathbf{B}(x) \wedge \mathbf{O}(\mathbf{B}(x)\mathbf{U}\mathbf{R}(x)))$.

Consider the *until* case with the formula $\mathbf{B}(x)\mathbf{U}\mathbf{R}(x)$. In the model shown in Figure 2.4 we have that $\sigma, \{x \mapsto a_0\} \models \mathbf{B}(x)\mathbf{U}\mathbf{R}(x)$ since there exists a counterpart after two steps with $\mathbf{R}(x)$, and for all worlds before it there exists a counterpart with $\mathbf{B}(x)$. However, clearly a_0 does not satisfy the expanded formula since neither $\sigma_1, \{x \mapsto a_1\} \models \mathbf{B}(x)\mathbf{U}\mathbf{R}(x)$ nor $\sigma_1, \{x \mapsto b_1\} \models \mathbf{B}(x)\mathbf{U}\mathbf{R}(x)$.

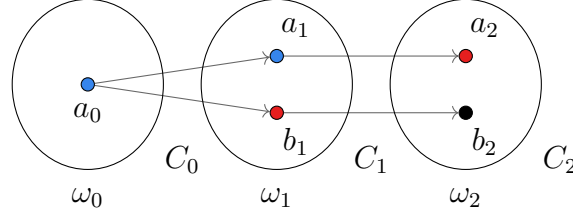



Figure 2.5: A counterexample model where, in the case of counterpart relations, we have that $\mathbf{B}(x)\mathbf{F}\mathbf{R}(x) \not\equiv \mathbf{R}(x) \vee (\mathbf{B}(x) \wedge \mathbf{A}(\mathbf{B}(x)\mathbf{F}\mathbf{R}(x)))$.

Consider the *until-forall* case with the formula $\mathbf{B}(x)\mathbf{F}\mathbf{R}(x)$. In the model shown in Figure 2.5 we have that a_0 satisfies the expanded formula, since the one-step counterparts a_1 and b_1 are such that both $\sigma, \{x \mapsto a_0\} \models \mathbf{B}(x)\mathbf{F}\mathbf{R}(x)$ and $\sigma, \{x \mapsto a_0\} \models \mathbf{B}(x)\mathbf{F}\mathbf{R}(x)$, with the world where all counterparts satisfy $\mathbf{R}(x)$ being reached after two and one steps, respectively. However, we have that $\sigma, \{x \mapsto a_0\} \not\models \mathbf{A}(\mathbf{B}(x)\mathbf{F}\mathbf{R}(x))$ since there is no single world ω_n where all counterparts after n steps satisfy $\mathbf{R}(x)$.

□

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

Theorem 2.7 (Expansion laws for partial functions). ( [FunctionalExpansionLaws](#))

The following equivalences hold in PNF whenever we restrict ourselves to counterpart traces $\sigma = (C_0, C_1, \dots)$ where each counterpart relation C_i is a partial function $C_i : D(w_i) \rightarrow D(w_{i+1})$:

$$\begin{aligned} \phi_1 \mathbf{U} \phi_2 &\equiv \phi_2 \vee (\phi_1 \wedge \mathbf{O}(\phi_1 \mathbf{U} \phi_2)) & \phi_1 \mathbf{F} \phi_2 &\equiv \phi_2 \vee (\phi_1 \wedge \mathbf{A}(\phi_1 \mathbf{F} \phi_2)) \\ \phi_1 \mathbf{W} \phi_2 &\equiv \phi_2 \vee (\phi_1 \wedge \mathbf{O}(\phi_1 \mathbf{W} \phi_2)) & \phi_1 \mathbf{T} \phi_2 &\equiv \phi_2 \vee (\phi_1 \wedge \mathbf{A}(\phi_1 \mathbf{T} \phi_2)). \end{aligned}$$

Remark 2.6 (Temporal operators as fixpoints). Consider the case of counterpart models where each counterpart relation is a partial function as in [Theorem 2.7](#): in a set-based semantics with fixpoints, we have that the *until* $\phi_1 \mathbf{U} \phi_2$ and *until-forall* $\phi_1 \mathbf{F} \phi_2$ operators correspond to least fixpoints of their respective expansion laws presented in [Theorem 2.7](#), and the *weak until* $\phi_1 \mathbf{F} \phi_2$ and *then* $\phi_1 \mathbf{T} \phi_2$ operators corresponding to greatest fixpoints.

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 $\mathbf{O}\phi \equiv \mathbf{A}\phi$, $\phi_1 \mathbf{U} \phi_2 \equiv \phi_1 \mathbf{F} \phi_2$, $\phi_1 \mathbf{W} \phi_2 \equiv \phi_1 \mathbf{T} \phi_2$, and this fact in turn allows us to obtain a similar notion of trace to that classically presented in LTL.

Chapter 3

Categorical semantics

In this chapter we provide a categorical presentation of the quantified logic introduced in [Chapter 2](#), by generalizing both the models and the semantics of our logic through the use of relational presheaves, counterpart \mathcal{W} -models, and classical attributes. In [Appendix A](#) we provide a self-contained exposition for the basic notions of category theory later assumed in the following sections.

3.1 Relational presheaves models

The crucial definition of Kripke frame presented in [Definition 2.1](#) admits a natural generalization in the categorical setting, namely the foundational concept of a *category*. Given a category \mathcal{W} , its objects A, B, C, \dots can be considered as worlds or instants of time, and the arrows $f : A \rightarrow B$ 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 taken to be a binary relation on the set W of worlds. This means that two worlds can only 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 generalize 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* $D : \mathcal{W}^{op} \rightarrow \mathbf{Set}$ 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 [Law69; Jac01].

Concretely, a presheaf assigns to each world $\omega \in \mathcal{C}$ a concrete set $D(\omega)$ of individuals, and to each time development $f : \omega \rightarrow \sigma$ a function $D(f) : D(\sigma) \rightarrow D(\omega)$ 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, thus disallowing the creation of new elements to be modeled. Considering a standard covariant functor $D : \mathcal{W} \rightarrow \mathbf{Set}$ would similarly allow for the creation of elements to be modeled, but not their deallocation, since the morphisms in \mathbf{Set} are taken to be total functions.

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

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

Given a relational presheaf D and a temporal development $f : \omega \rightarrow \sigma$, 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 [Figure 3.1](#) a pictorial example of relational presheaf on a category \mathcal{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;
- $D : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ is a relational presheaf on \mathcal{W} .

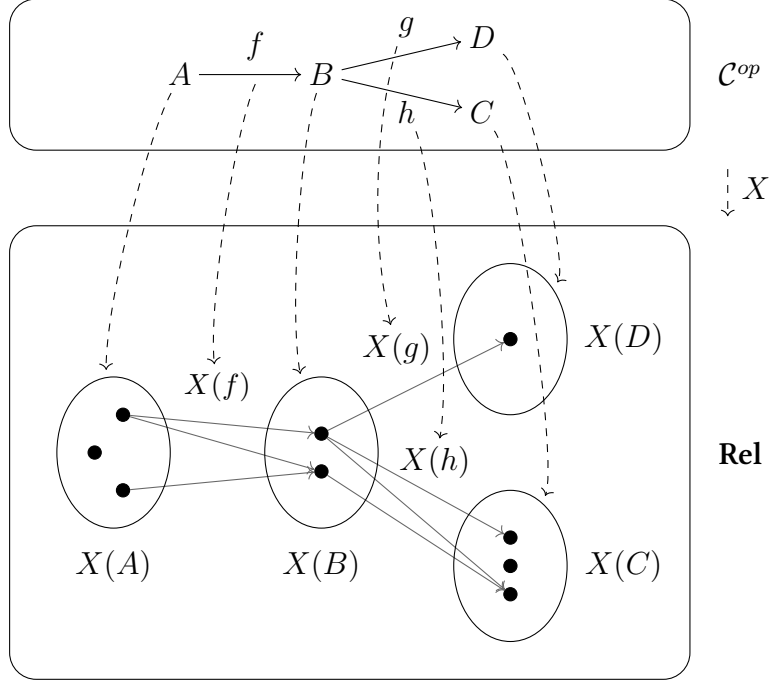


Figure 3.1: An example of a relational presheaf X on a category \mathcal{C} .

A crucial difference with classical counterpart models is that counterpart \mathcal{W} -models introduce considerably more morphisms than those that might be desirable in a model. 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 entities are neither created nor 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 tran-*

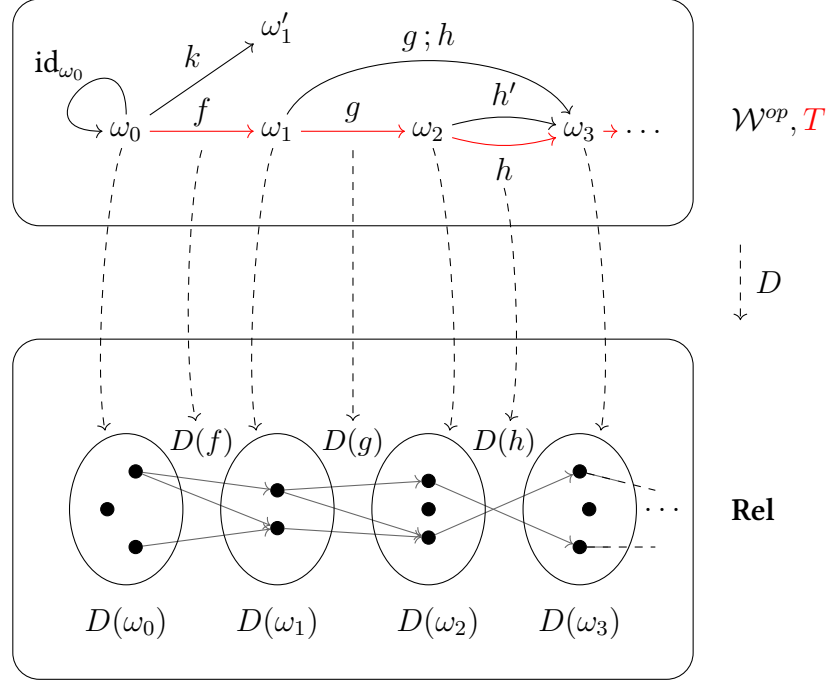


Figure 3.2: An example of temporal counterpart \mathcal{W} -model $\langle \mathcal{W}, D, T \rangle$, where the temporal structure T is linear in the sense of Definition 3.5.

sitions, 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*.

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 Figure 3.2.

In particular, by restricting ourselves to only certain kinds of temporal structures, we obtain as particular instances the classical models of LTL and CTL. For example, the classical traces of LTL can be retrieved by considering as temporal structure the notion of *flow of time*.

Definition 3.5. A **flow of time** is a pair $(F, <)$ where F is a set whose elements are called *time points*, and $<$ is an irreflexive and transitive binary relation on

F . Given two time points $a, b \in F$, $a < b$ means that a is *earlier* than b , and the irreflexivity constraint represents the fact that arrows are time irreversible processes, i.e., worlds cannot be in the past of future of themselves. A flow of time is *linear* if given $a, b \in F$ with $a \neq b$ then either $a < b$ or $b < a$.

By equipping presheaves with temporal structures, we have that the notion of QLTL trace given in [Definition 2.5](#) is equivalent to a temporal counterpart \mathcal{W} -model $\langle \mathcal{W}, D, T \rangle$ where the temporal structure T is given by a linear flow of time $(F, <)$ in the intuitive way. In general, we can consider a general notion of *paths* given by a temporal structure, which we later employ in [Definition 3.12](#) to present the semantics of our logic.

Definition 3.6 (Paths). Notationally, given a temporal structure T we denote by $\text{path}(T, \omega)$ the class of possible sequences of arrows $t = (t_0, t_1, t_2, \dots)$ such that $t_i \in T$ and $\text{cod}(t_i) = \text{dom}(t_{i+1})$ for any $i \geq 0$, with the sequence starting with $\text{dom}(t_0) = \omega$. Whenever the path $t \in \text{path}(T, \omega)$ is clear from the context, we indicate with $\omega_i = \text{cod}(t_i)$ the i -th world of the path.

Furthermore, equipping counterpart \mathcal{W} -models with temporal structures allows us to formally link classical counterpart models with their categorical version.

Theorem 3.1. There is a bijective correspondence between counterpart models $\langle W, d, \mathcal{C} \rangle$ and temporal counterpart \mathcal{W} -models $\langle \mathcal{W}, D, T \rangle$.

Proof. We provide the constructions explicitly in the two directions, and it is easy to check they are inverses.

- (\Rightarrow) Given a classical counterpart model $\langle W, d, \mathcal{C} \rangle$, we construct a temporal counterpart \mathcal{W} -model $\langle \mathcal{W}, D, T \rangle$ as follows:
 - \mathcal{W} is the category with $\text{Obj}(\mathcal{W}) := W$ as objects and whose arrows are freely generated (i.e.: adding identities and compositions) by introducing a morphism $r : \omega_1 \rightarrow \omega_2$ for each relation $R \in \mathcal{C}\langle \omega_1, \omega_2 \rangle$.
 - D is the relational presheaf $D : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ defined as $D(\omega) := d(\omega)$, and assigning to each arrow $r : \omega_1 \rightarrow \omega_2$ its generating relation $R \in \mathcal{C}\langle \omega_1, \omega_2 \rangle$ 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 $r : \omega_1 \rightarrow \omega_2$ given by the one-step relations $R \in \mathcal{C}\langle \omega_1, \omega_2 \rangle$ of the model.
- (\Leftarrow) Given a temporal counterpart \mathcal{W} -model $\langle \mathcal{W}, D, T \rangle$, we construct a classical counterpart model $\langle W, d, \mathcal{C} \rangle$ as follows:

- $W := \text{Obj}(\mathcal{W})$ 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 ;
- \mathcal{C} is the function assigning to each tuple $\langle \omega_1, \omega_2 \rangle$ the set of relations $\mathcal{C}\langle \omega_1, \omega_2 \rangle := \{D(r) \mid r : \omega_1 \rightarrow \omega_2 \wedge r \in T\}$, where each morphism r of \mathcal{W} must be selected by the temporal structure T .

□

3.3 Presheaf semantics for QLTL

After having introduced the categorical perspective on the counterpart models of our logic, we now introduce 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*: the intuition is that a classical attribute is simply a family of sets indexed on worlds, associating to each world ω only the subset of individuals in ω that satisfy a given property.

Definition 3.7 (Classical attributes). Let $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ 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 $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ 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. Therefore, given a relational presheaf $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ 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 $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$, 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 $\perp = \{\emptyset\}_{\omega \in \mathcal{W}}$.

Remark 3.1 (Classical attributes are presheaves). A classical attribute can alternatively be considered as a (relational) presheaf $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ 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 attributes 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 $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$, we can define the temporal operators of our logic as operators $O-$, $-U-$, $-W-$ that combine classical attributes and return other classical attributes. This mirrors the intuition that the set of individuals satisfying a composite formula $\phi_1 U \phi_2$ can be obtained by knowing which elements satisfy the subformulae ϕ_1, ϕ_2 composing it.

Definition 3.8 (Temporal operators on classical attributes). Let $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ be a relational presheaf and T a fixed 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**:

- $s \in (OA)_\omega$ iff for every arrow $r : \omega \rightarrow \sigma$ of T there exists an element $z \in X(\sigma)$ such that $\langle z, s \rangle \in X(r)$ and $z \in A_\sigma$;
- $s \in (AA)_\omega$ iff for every arrow $r : \omega \rightarrow \sigma$ of T and any element $z \in X(\sigma)$ such that $\langle z, s \rangle \in X(r)$ we have that $z \in A_\sigma$;
- $s \in (AUB)_\omega$ iff for every $t \in \text{path}(T, \omega)$ there exists an $\bar{n} \geq 0$ such that:
 1. for any $i < \bar{n}$, there exists $z_i \in X(\omega_i)$ such that $\langle z_i, s \rangle \in X(t_{\leq i})$ and $z_i \in A_{\omega_i}$;
 2. there exists $z_{\bar{n}} \in X(\omega_{\bar{n}})$ such that $\langle z_{\bar{n}}, s \rangle \in X(t_{\leq \bar{n}})$ and $z_{\bar{n}} \in B_{\omega_{\bar{n}}}$.
- $s \in (AFB)_\omega$ iff for every $t \in \text{path}(T, \omega)$ there exists an $\bar{n} \geq 0$ such that:
 1. for every $i < \bar{n}$ and any $z_i \in X(\omega_i)$ 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 $z_{\bar{n}} \in B_{\omega_{\bar{n}}}$.

- $s \in (AWB)_\omega$ iff one of the following holds:
 - the same conditions for $s \in (AUB)_\omega$ apply;
 - for any i there exists $z_i \in X(\omega_i)$ such that $\langle z_i, s \rangle \in X(t_{\leq i})$ and $z_i \in A_{\omega_i}$.
- $s \in (ATB)_\omega$ iff one of the following holds:
 - the same conditions for $s \in (AFB)_\omega$ apply;
 - for any i and any $z_i \in X(\omega_i)$ with $\langle z_i, s \rangle \in X(t_{\leq i})$ we have that $z_i \in A_{\omega_i}$.

Remark 3.2 (Eventually and always). Given a relational presheaf $X : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$, 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 every $t \in \text{path}(T, \omega)$ there exists 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 every $t \in \text{path}(T, \omega)$ there exists 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 every $t \in \text{path}(T, \omega)$ and $i \geq 0$ there exists 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 every $t \in \text{path}(T, \omega)$ 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 $t \in \text{path}(T, \omega)$ provided by the model: notice, however, that the positive normal form equivalences for QLTL and PNF expressed in [Section 2.3.2](#) clear only hold in the case where the temporal structure T is a linear flow of time, with the time developments forming a linear a QLTL trace in th sense of [Definition 3.5](#).

We will use these temporal operators on classical attributes when providing the full semantics of the logic in [Definition 3.12](#).

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 the world ω the set of individuals in ω that satisfy the formula. In fact, *classical attributes are the categorical generalization 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. To make this intuition precise, we introduce products and a terminal object for relational presheaves:

Definition 3.9 (Product of relational presheaves). Given two relational presheaves $X, Y : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$, the **product of relational presheaves** $X \times Y : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ is computed pointwise using 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 $r : \omega_1 \rightarrow \omega_2$ 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 3.10 (Terminal relational presheaf). The **terminal relational presheaf** $\perp : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ is defined as the relational presheaf $\perp(\omega) = \{*\}$ assigning the singleton set $\{*\}$ to all worlds $\omega \in \mathcal{W}$, and assigning the identity relation on $\{*\}$ to every morphism.

Definition 3.11 (Relational presheaf of a context). For any context $\Gamma = [x_1, \dots, x_n]$ as presented in [Definition 2.7](#), we define the **context relational presheaf of Γ** as the presheaf $\llbracket \Gamma \rrbracket$ defined by

$$\llbracket \Gamma \rrbracket = \underbrace{D \times \dots \times D}_{n \text{ times}}$$

where \times denotes the product of relational presheaves given in [Definition 3.9](#). If the context is empty, we define $\llbracket \emptyset \rrbracket := \perp$. Given a variable $x \in \Gamma$, we indicate with $\pi_x : \llbracket \Gamma \rrbracket \rightarrow D$ 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 3.3 (Classical attribute on a singleton). Consider the case where a classical attribute A is given on the terminal relational presheaf $\perp : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$. Then each classical attribute A on \perp 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 not *for the entire world*. In light of [Definition 3.11](#), classical attributes on \perp correspond with the semantics of closed formulae.

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

Definition 3.12 (Satisfiability of a formula). Given a formula-in-context $[I]\phi$, the classical attribute $\llbracket [I]\phi \rrbracket$ on $\llbracket I \rrbracket$ is a function defined by induction on the formula $[I]\phi$ as follows:

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

In the case of temporal operators, the classical attribute $\llbracket [I]\phi \rrbracket$ is given directly by the operators defined in [Definition 3.8](#):

- $\llbracket [I]O\phi \rrbracket := O\llbracket [I]\phi \rrbracket$;
- $\llbracket [I]\phi_1 U \phi_2 \rrbracket := \llbracket [I]\phi_1 \rrbracket U \llbracket [I]\phi_2 \rrbracket$;
- $\llbracket [I]\phi_1 W \phi_2 \rrbracket := \llbracket [I]\phi_1 \rrbracket W \llbracket [I]\phi_2 \rrbracket$;

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 $\llbracket I \rrbracket$.

3.4 Multi-sorted algebra models

In this section we consider a specialization 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 [Cou90; Cou00; DGG07].

We briefly recall in this section the fundamentals of multi-sorted algebras and signatures.

Definition 3.13 (Signature). A **many-sorted signature** Σ is a pair $\langle \Sigma_S, \Sigma_F \rangle$ where:

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

Definition 3.14 (Algebra). A **many-sorted algebra** S with signature Σ , i.e. a Σ -algebra, is a pair $\langle S, F \rangle$ where:

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

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

- $\text{Gr}_S = \{\text{Node}, \text{Edge}\}$;
- $\text{Gr}_F = \{s : \text{Edge} \rightarrow \text{Node}, t : \text{Edge} \rightarrow \text{Node}\}$, representing the source and target functions on edges.

Example 3.4 (Example of graph). In concrete, 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 Figure 3.3. 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.

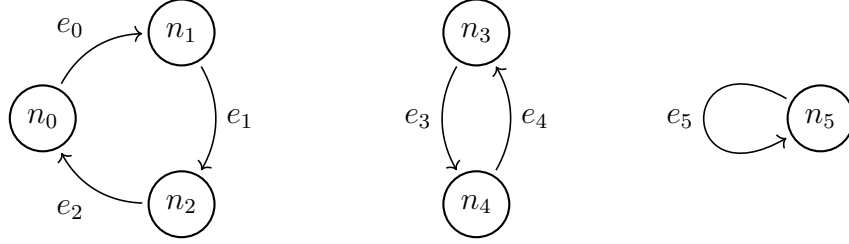


Figure 3.3: Examples of three Gr-algebras G_0 (left), G_1 (middle), G_2 (right).

Definition 3.15 (Relational homomorphism of algebras). Given two algebras A and B , a **relational homomorphism of algebras** ρ is a family of relations $\rho := \{\rho_\tau \subseteq A_\tau \times B_\tau \mid \tau \in \Sigma_S\}$ typed over Σ_S such that, for every function symbol $f : \tau_1 \times \cdots \times \tau_n \rightarrow \tau$ and every list of elements $(a_1, \dots, a_n) \in A_{\tau_1} \times \cdots \times A_{\tau_n}$ and $(b_1, \dots, b_n) \in B_{\tau_1} \times \cdots \times B_{\tau_n}$ we have that

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

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.

In order 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 3.16 (Typed context). Given a denumerable set of variables X , a **typed context** Γ over a signature Σ is a finite list $[x_1 : \tau_1, \dots, x_n : \tau_n]$ of pairs $(x_i, \tau_i) \in X \times \Sigma_S$ such that x_1, \dots, x_n are distinct.

Definition 3.17 (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

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

where $f : \tau_1 \times \cdots \times \tau_n \rightarrow \tau$ 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 group 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 3.18 (Relational morphisms). A **relational morphism** between two relational presheaves $X, Y : \mathcal{W}^{op} \rightarrow \mathbf{Rel}$ is a family of set functions $\eta = \{\eta_\omega : X(\omega) \rightarrow Y(\omega)\}_{\omega \in \mathcal{W}}$ such that for every morphism $f : A \rightarrow B$ 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 3.19 (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} ;
- $\mathcal{S} = \{\llbracket \tau \rrbracket : \mathcal{W}^{op} \rightarrow \mathbf{Rel}\}_{\tau \in \Sigma_S}$ is a set of relational presheaves on \mathcal{W} , assigning a relational presheaf to each sort in Σ_S ;
- $\mathcal{F} = \{\mathcal{I}(f) : \llbracket \tau_1 \rrbracket \times \cdots \times \llbracket \tau_n \rrbracket \rightarrow \llbracket \tau \rrbracket\}_{f \in \Sigma_F}$ is a set of relational morphisms, assigning a relational morphism to each function symbol $f : \tau_1 \times \cdots \times \tau_n \rightarrow \tau$ 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 [Figure 3.4](#) 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.

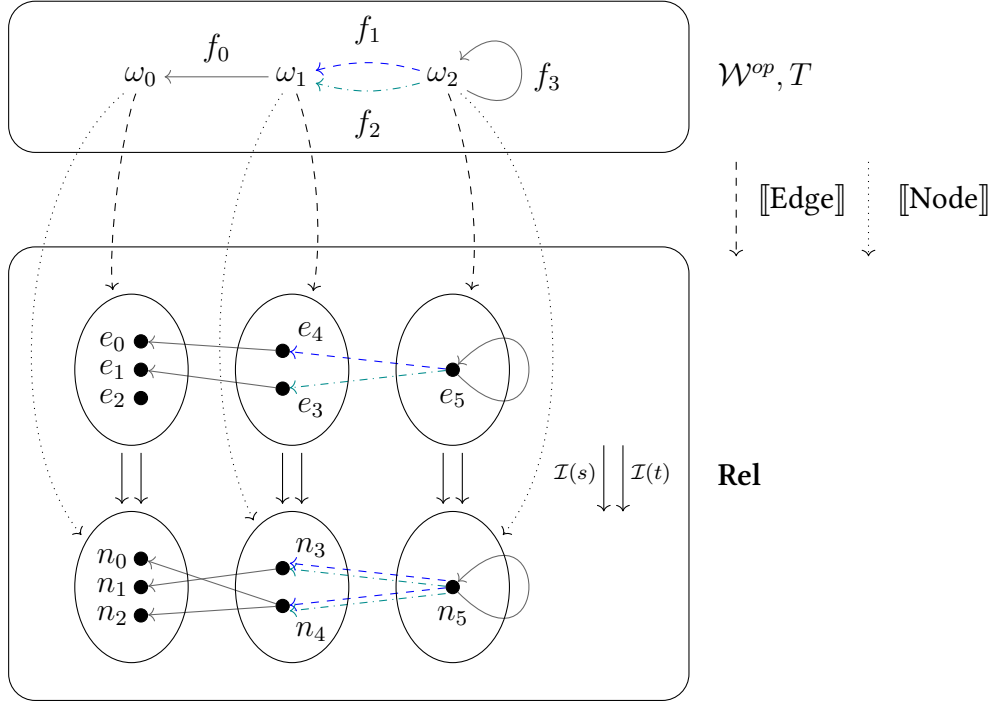


Figure 3.5: Explicit categorical data given by [Example 3.5](#).

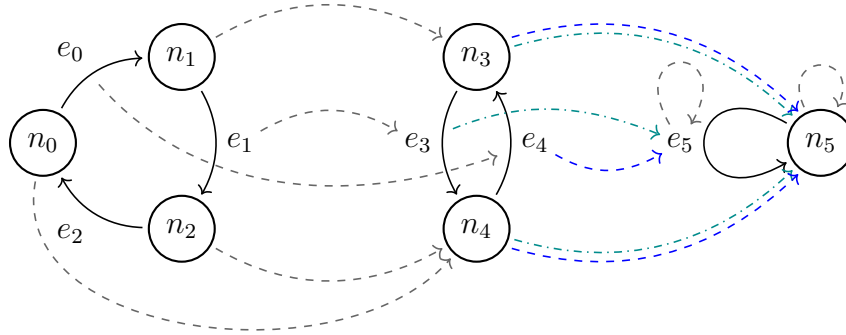
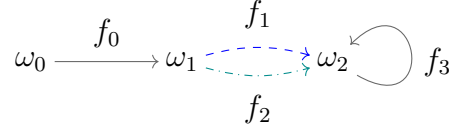


Figure 3.4: Graphical representation of an algebraic counterpart \mathcal{W} -model.

We provide the categorical data given by the model by describing explicitly each component. A concrete perspective of our model is shown in [Figure 3.5](#), where relational presheaves are represented explicitly as functors from \mathcal{W}^{op} to \mathbf{Rel} .

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



- The temporal structure T is a family of morphism 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 $\{\text{Edge}, \text{Node}\}$ are given by the following data. We first consider the action on objects:

$$\begin{aligned} \llbracket \text{Edge} \rrbracket(\omega_0) &= \{e_0, e_1, e_2\}, & \llbracket \text{Node} \rrbracket(\omega_0) &= \{n_0, n_1, n_2\}; \\ \llbracket \text{Edge} \rrbracket(\omega_1) &= \{e_3, e_4\}, & \llbracket \text{Node} \rrbracket(\omega_1) &= \{n_3, n_4\}; \\ \llbracket \text{Edge} \rrbracket(\omega_2) &= \{e_5\}, & \llbracket \text{Node} \rrbracket(\omega_2) &= \{n_5\}; \end{aligned}$$

By considering the action on morphisms, we define the following assignments:

$$\begin{aligned} \llbracket \text{Edge} \rrbracket(f_0) &= \{(e_4, e_0), (e_3, e_1)\}, & \llbracket \text{Node} \rrbracket(f_0) &= \{(n_4, n_0), (n_3, n_1), (n_4, n_2)\}; \\ \llbracket \text{Edge} \rrbracket(f_1) &= \{(e_5, e_4)\}, & \llbracket \text{Node} \rrbracket(f_1) &= \{(n_5, n_3), (n_5, n_4)\}; \\ \llbracket \text{Edge} \rrbracket(f_2) &= \{(e_5, e_3)\}, & \llbracket \text{Node} \rrbracket(f_2) &= \{(n_5, n_3), (n_5, n_4)\}; \\ \llbracket \text{Edge} \rrbracket(f_3) &= \{(e_5, e_5)\}, & \llbracket \text{Node} \rrbracket(f_3) &= \{(n_5, n_5)\}; \end{aligned}$$

- 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:

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

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 summarize 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, existence of individuals $\exists x.\phi$ is now 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 generalizing [Definition 3.11](#) and similarly provide the interpretation of typed contexts as relational presheaves.

Definition 3.20 (Typed contexts as relational presheaves). Given a typed context $\Gamma = [x_1 : \tau_1, \dots, x_n : \tau_n]$, we define the **typed context relational presheaf** as the relational presheaf $\llbracket \Gamma \rrbracket$ given by

$$\llbracket \Gamma \rrbracket := \llbracket \tau_1 \rrbracket \times \dots \times \llbracket \tau_n \rrbracket$$

where \times denotes the product of relational presheaves defined in [Definition 3.9](#).

Definition 3.21 (Terms as relational morphisms). Given a typed context Γ and a term $[\Gamma] t : \tau$, we define the **term relational morphism** $\llbracket t \rrbracket$ 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 $\llbracket t \rrbracket$ is given by the relational morphism

$$\llbracket \Gamma \rrbracket \xrightarrow{\pi_i} \llbracket \tau \rrbracket$$

where π_i is the i -th projection out of the product of relational presheaves.

- if $t = f(t_1, \dots, t_n)$, then $\llbracket t \rrbracket$ is given by the composition of relational morphisms

$$\llbracket \Gamma \rrbracket \xrightarrow{\langle \llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket \rangle} \llbracket \Gamma' \rrbracket \xrightarrow{\mathcal{I}(f)} \llbracket \tau \rrbracket$$

where $\langle \llbracket t_1 \rrbracket, \dots, \llbracket t_n \rrbracket \rangle$ denotes the universal property of the n -ary product of relational presheaves, intuitively mapping the relational morphisms $\llbracket t_i \rrbracket$ to each component of the product.

Finally, we extend the interpretation of QLTL in the algebraic setting by generalizing formulae-in-context to typed contexts, and then providing its semantics.

Definition 3.22 (Algebraic QLTL). Given a set of denumerable variables \mathcal{X} with $x \in \mathcal{X}$, the syntax of **algebraic QLTL** formulae-in-context is given by:

$$\psi := \text{true} \mid s =_{\tau} t \mid P(s), \quad \phi := \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \exists_{\tau} x. \phi \mid \text{O}\phi \mid \phi_1 \text{U}\phi_2 \mid \phi_1 \text{W}\phi_2,$$

where $[I] s : \tau$ and $[I] t : \tau$ are terms defined in the context I 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 3.23 (Semantics of algebraic QLTL). Given the semantic interpretation of QLTL formulae in [Definition 3.12](#), we consider the following additional definitions:

- $\llbracket [I] P(s) \rrbracket_{\omega} := \{a \in \llbracket [I] \rrbracket(\omega) \mid P(\llbracket s \rrbracket_{\omega}(a))\};$
- $\llbracket [I] s = t \rrbracket_{\omega} := \{a \in \llbracket [I] \rrbracket(\omega) \mid \llbracket s \rrbracket_{\omega}(a) = \llbracket t \rrbracket_{\omega}(a)\};$
- $\llbracket [I] \exists_{\tau} x. \phi \rrbracket_{\omega} := \{a \in \llbracket [I] \rrbracket(\omega) \mid \exists b \in \llbracket \tau \rrbracket(\omega). \langle a, b \rangle \in \llbracket [I, x : \tau] \phi \rrbracket_{\omega}\}.$

The intuitive semantics for the first rule is that it identifies the set of assignments on the free variables of the term s such that $P(s)$ holds.

Notice how the definitions of temporal operators given in [Definition 3.8](#) to deal with the temporal operators $\text{O}-$, $- \text{U}-$, $- \text{W}-$ and their universally quantified counterparts also remain unchanged, since the base relational presheaf X is simply the relational presheaf associated to a typed context $\llbracket [I] \rrbracket(\omega)$.

3.7 Examples

We provide some examples of satisfiability for simple algebraic QLTL formulae on the running example in [Figure 3.4](#). Taking for example the formulae

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

we can obtain the set of edges of each graph G_1, G_2, G_3 that survive at the next steps:

$$\begin{aligned} \llbracket [x : \text{Edge}] \text{nextStepPreserved}(x) \rrbracket_{w_0} &= \{e_0, e_1\} \\ \llbracket [x : \text{Edge}] \text{nextStepPreserved}(x) \rrbracket_{w_1} &= \{\} \\ \llbracket [x : \text{Edge}] \text{nextStepPreserved}(x) \rrbracket_{w_2} &= \{e_5\} \end{aligned}$$

Notice that no edge is **nextStepPreserved** in the second case, since the development $\llbracket \text{Edge} \rrbracket(f_1)$ deallocates the edge e_4 and, similarly, $\llbracket \text{Edge} \rrbracket(f_2)$ deallocates the edge e_3 : following the semantics presented in [Definition 3.8](#), we require that a given property has to hold for *every* time development of length one.

By considering deallocations we have the following:

$$\begin{aligned}\llbracket [x : \text{Edge}] \text{ nextStepDeallocated}(x) \rrbracket_{w_0} &= \{e_2\} \\ \llbracket [x : \text{Edge}] \text{ nextStepDeallocated}(x) \rrbracket_{w_1} &= \{\} \\ \llbracket [x : \text{Edge}] \text{ nextStepDeallocated}(x) \rrbracket_{w_2} &= \{\}\end{aligned}$$

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:

$$\begin{aligned}\llbracket [x : \text{Node}] \text{ nextStepPreserved}(x) \rrbracket_{w_0} &= \{n_1, n_2\} \\ \llbracket [x : \text{Node}] \text{ nextStepPreserved}(x) \rrbracket_{w_1} &= \{n_3\} \\ \llbracket [x : \text{Node}] \text{ nextStepPreserved}(x) \rrbracket_{w_2} &= \{n_5\}\end{aligned}$$

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

$$\begin{aligned}\text{loop}(e) &:= s(e) =_{\text{Node}} t(e), \\ \text{nodeHasLoop}(n) &:= \exists_{\text{Edge}} e. s(e) =_{\text{Node}} n \wedge \text{loop}(e), \\ \text{hasLoop} &:= \exists_{\text{Edge}} e. \text{loop}(e), \\ \text{willBecomeLoop}(e) &:= \neg \text{loop}(e) \wedge \Diamond \text{loop}(e)\end{aligned}$$

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

$$\begin{aligned}\llbracket [x : \text{Node}] \text{ nodeHasLoop}(x) \rrbracket_{w_0} &= \{\} \\ \llbracket [x : \text{Node}] \text{ nodeHasLoop}(x) \rrbracket_{w_1} &= \{\} \\ \llbracket [x : \text{Node}] \text{ nodeHasLoop}(x) \rrbracket_{w_2} &= \{n_5\}\end{aligned}$$

Alternatively, we can express this by stating that the loop belongs to the entire world:

$$\begin{aligned}\llbracket \emptyset \text{ hasLoop} \rrbracket_{w_0} &= \{\} \\ \llbracket \emptyset \text{ hasLoop} \rrbracket_{w_1} &= \{\} \\ \llbracket \emptyset \text{ hasLoop} \rrbracket_{w_2} &= \{*\}\end{aligned}$$

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 3.3](#).

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:

$$\begin{aligned}\llbracket [x : \text{Node}] \text{ willBecomeLoop}(x) \rrbracket_{w_0} &= \{\} \\ \llbracket [x : \text{Node}] \text{ willBecomeLoop}(x) \rrbracket_{w_1} &= \{\} \\ \llbracket [x : \text{Node}] \text{ willBecomeLoop}(x) \rrbracket_{w_2} &= \{\}\end{aligned}$$

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.

$$\begin{aligned}\llbracket [x : \text{Node}] \neg \text{nodeHasLoop}(x) \wedge \Diamond \text{nodeHasLoop}(x) \rrbracket_{w_0} &= \{n_0, n_1, n_2\} \\ \llbracket [x : \text{Node}] \neg \text{nodeHasLoop}(x) \wedge \Diamond \text{nodeHasLoop}(x) \rrbracket_{w_1} &= \{n_3, n_4\} \\ \llbracket [x : \text{Node}] \neg \text{nodeHasLoop}(x) \wedge \Diamond \text{nodeHasLoop}(x) \rrbracket_{w_2} &= \{\}\end{aligned}$$

Chapter 4

Agda formalization

In this chapter we present an overview of one the main contributions of this thesis, which is a complete formalization of the categorical semantics presented in [Chapter 3](#) using the dependently typed programming language and proof assistant Agda [Nor09]. We introduce here just a part of the formalized codebase, and we focus on the most important structures and definitions by providing the formal semantics of our logic directly with algebraic counterpart \mathcal{W} -models. The full code of the categorical formalization is available at:

<https://github.com/iwilare/categorical-ctl>

We assume in this chapter that the reader is already 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 Agda and its syntax. For a complete introduction to Agda we refer to [WKS22].

4.1 Formalization aspects

We start by highlighting some general ideas of the formalization presented in this thesis. In our work we embed a first-order fragment of QLTL in Agda, defining both a classical and a categorical semantics. Embedding a temporal logic in a proof assistant essentially amounts to the following points:

1. being able to define the *class of models* on which the logic is defined,
2. being able to formalize *syntax and semantics* of the logic,

3. being able to prove *metatheorems* about such semantics and construct *algorithms* that can operate on the logic and its models,
4. being able to show that formulae of the logic are *satisfied for a specific model*.

We identify two main actors interplaying with the formalization aspects mentioned above:

- the **implementer** of the logic, which primarily formalizes aspects (1.) and (2.) of the logic by establishing the definitions and constructions required to give a mechanized presentation. Optionally, (3.) can be implemented by proving metatheorems about the logic for both theoretical (e.g. positive normal form results) and practical convenience, for example by providing model checking algorithms or mechanisms for proof automation.
- the **user** of the logic, which, being concerned with practicality and expressiveness of logic, uses the infrastructure provided by the implementer in order to show properties on their model of interest, thus focusing their attention on the usability aspects of (4.).

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* [GLT89]. This paradigm gives a *constructive* 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* and *double negation elimination*, are *not provable* in the system. Consequently, both implementer and user would not be able to prove that in our logic QLTL the formula $\neg\neg\phi \implies \phi$ always holds for any choice of models and formula ϕ , since it is also not provable in the metalanguage provided by the proof assistant. Both (3.) and (4.) directly concern themselves with the *metacapabilities* of the proof assistant in which we work, which means that, without explicitly assuming any other logical principle, the embedding of our temporal logic is actually restricted to the *intuitionistic* fragment of QLTL.

The negation of logical connectives can also be cumbersome to handle practically. In constructive mathematics, negation is defined as the implication $\neg\phi := \phi \implies \perp$, where \perp 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 that presented in [Section 2.3](#).

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

$$\sigma, \mu \models_{\text{QLTL}} \neg(\neg\phi_1 \mathbf{U} \neg\phi_2).$$

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

$$\sigma, \mu \models_{\text{QLTL}} \phi_2 \mathbf{W}(\phi_1 \wedge \phi_2),$$

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

Even working with the negation of simple connectives such as $\neg(\phi_1 \vee \phi_2)$ can 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.

In order to tackle these usability issues and the treatment of negation in the constructive setting, we take the following approach: the formulae of the logic are expressed in Agda directly using a full *positive normal form* similar to that 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 just shown. On the other hand, the positive normal form is supported by the equiexpressivity results shown in [Theorem 2.4](#), which in turn *do* require for classical reasoning and principles to be postulated. This effectively shifts the burden of dealing (classically) with negation from the user to the implementer, while also providing a theoretical guarantee that no expressive power is either gained or lost in the alternative presentation of the logic.

4.1.1 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 relatively limited, especially when compared to proof assistants where automation and the use of tactics is a core aspect of the software environment, such as Coq [Coq16], Lean [MU21], and Isabelle [NPW02]. The Agda synthesizer Apsy [LB06] is the main helper tool in Agda implementing a form of automated proof search. Unfortunately, since Apsy only provides general-purpose searching procedures, its theorem proving capabilities are nowhere near those of a specialized model checking algorithm. However, 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.

4.1.2 Category theory

The `agda-categories` library [HC21] 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 enter too much in detail in the definitions and structures provided by the library.

4.2 Formalization

In the following sections we present our formalization work, starting with the formalization of algebras. We then proceed by describing algebraic counterpart \mathcal{W} -models along with the categorical definitions behind them. Finally, we introduce the syntax and semantics of our quantified temporal logic, and formalize 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.

4.3 Signatures and algebras

We start by introducing the notion of signatures, algebras, and terms, along the techniques needed to formalize them. In order 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 { $\ell$ } where

record FunctionSignature ( $\mathcal{S} : \text{Set } \ell$ ) :  $\text{Set } \ell$  where
  constructor  $\_ \mapsto \_$ 
  field
    {arity} :  $\mathbb{N}$ 
     $\tau^* : \text{Vec } \mathcal{S} \text{ arity}$ 
     $\tau : \mathcal{S}$ 
```

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

```
record Signature :  $\text{Set } (\text{suc } \ell)$  where
  field
     $\mathcal{S} : \text{Set } \ell$ 
     $\mathcal{F} : \text{Set } \ell$ 
     $\text{sign}\mathcal{F} : \mathcal{F} \rightarrow \text{FunctionSignature } \mathcal{S}$ 

  args =  $\tau^* \circ \text{sign}\mathcal{F}$ 
  ret =  $\tau \circ \text{sign}\mathcal{F}$ 
```

An algebra for a given signature Σ amounts to providing a function \mathbf{S} from symbols to sets and a function \mathbf{F} from function symbols to actual functions with type given by the signature.

```
record  $\Sigma$ -Algebra ( $\Sigma : \text{Signature}$ ) :  $\text{Set } (\text{suc } \ell)$  where
  field
     $\mathbf{S} : \mathcal{S} \rightarrow \text{Set } \ell$ 

  argType :  $\mathcal{F} \rightarrow \text{Set } \ell$ 
  argType f =  $\text{mapT } \mathbf{S} (\text{args } f)$ 

  retType :  $\mathcal{F} \rightarrow \text{Set } \ell$ 
```

$retType\ f = \mathbf{S}\ (\mathbf{ret}\ f)$

field

$\mathbf{F} : \forall (f : \mathcal{F}) \rightarrow argType\ f \rightarrow retType\ f$

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 will refer to lemmas and properties of types obtained this way as the module **VecT**. The unit type **T** and its single element ***** is used in the case of the empty vector, and in the inductive case we combine the set fx given by the function using the cartesian product \times .

$mapT : (A \rightarrow \mathbf{Set}\ \ell) \rightarrow \mathbf{Vec}\ A\ n \rightarrow \mathbf{Set}\ \ell$

$mapT\ f\ [] = \mathbf{T}$

$mapT\ f\ (x :: v) = fx \times mapT\ f\ v$

Given some (possibly heterogeneous) relation R , we can relate two **Vecs** obtained with **mapT** if they are pointwise related with R :

$zip : \forall \{v : \mathbf{Vec}\ A\ n\} \{fg : A \rightarrow \mathbf{Set}\ \ell'\}$

$\rightarrow (\forall \{x\} \rightarrow fx \rightarrow gx \rightarrow \mathbf{Set}\ \ell)$

$\rightarrow mapT\ f\ v \rightarrow mapT\ g\ v \rightarrow \mathbf{Set}\ \ell$

$zip\ \{v = []\}\ R\ \mathbf{*}\ \mathbf{*} = \mathbf{T}$

$zip\ \{v = _ :: _\}\ R\ (x, xs)\ (y, ys) = R\ x\ y \times zip\ R\ xs\ ys$

We can define the type **Σ -Rel** of relational homomorphisms between algebras given in [Definition 3.15](#). In order to specify the homomorphism property **ρ -homo**, we use **zip** to relate pointwise the function arguments given by the two algebras:

record **Σ -Rel** $\{\Sigma\}$ $(A : \Sigma\text{-Algebra}\ \Sigma)\ (B : \Sigma\text{-Algebra}\ \Sigma) : \mathbf{Set}\ (\mathbf{suc}\ \ell)\ \ell$ **where**

open **Signature** Σ

private

module **A** = $\Sigma\text{-Algebra}\ A$

module **B** = $\Sigma\text{-Algebra}\ B$

field

$\rho : \forall \{\tau\} \rightarrow \mathbf{REL}\ (A.S\ \tau)\ (B.S\ \tau)\ \ell$

```

ρ-homo :
  ∀ (f:  $\mathcal{F}$ )
  → {as : A.argType f}
  → {bs : B.argType f}
  → zip ρ as bs
  → ρ (A.F f as) (B.F f bs)

```

4.4 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  $\mathcal{S}$ 

```

The type of terms-in-context `_⊢_⟨_⟩` is given inductively, and is parameterized 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 formalization of our temporal logics.

```

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

  fun : ∀ {s}
    → (f:  $\mathcal{F}$ )
    → 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)
```

4.5 Relational presheaves

We can now introduce the notion of relational presheaf on a given category C , which we give by parameterizing the `RelPresheaves` module.

```
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 $C [\omega_1, \omega_2]$ 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
  eta-equality
  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 the 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

```

}
}

4.6 Counterpart models

We now provide the definition of counterpart model from the non-categorical perspective. First, we define a standard Lewis-style 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 the two worlds whenever the worlds 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. This will be the principal notion of non-categorical model used in our examples in [Section 4.12](#).

```
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.

4.7 Algebraic counterpart \mathcal{W} -model

We now provide the categorical 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 [F₀](#) and [F₁](#).

module [ContextPresheaf](#) $\{\ell\} \{W : \text{Category } \ell \ell \ell\} \{\mathcal{S} : \text{Set } \ell\}$
 $(\llbracket _ \rrbracket : \mathcal{S} \rightarrow \text{RelPresheaf } W)$ **where**

$\llbracket _ \rrbracket^* : \forall \{n\} \rightarrow \text{Vec } \mathcal{S} \ n \rightarrow \text{RelPresheaf } W$
 $\llbracket \Gamma \rrbracket^* =$
record
 $\{ \text{F}_0 = \lambda \omega \rightarrow \text{mapT } (\lambda \Sigma \rightarrow \text{F}_0 (\llbracket \Sigma \rrbracket) \omega) \Gamma$
 $; \text{F}_1 = \lambda f \rightarrow \text{zip } (\lambda \{\Sigma\} \rightarrow \text{F}_1 (\llbracket \Sigma \rrbracket) f)$
 $\}$

An algebraic counterpart \mathcal{W} -model on a given signature is simply the collection of the three fields given in [Definition 3.19](#): a category [W](#), a presheaf $\llbracket \tau \rrbracket$ 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](#) $\{\ell\} (\Sigma : \text{Signature } \{\ell\}) : \text{Set } (\text{suc } \ell \ell)$ **where**
field
 $\text{W} : \text{Category } \ell \ell \ell$
 $\llbracket _ \rrbracket : \forall (\tau : \mathcal{S}) \rightarrow \text{RelPresheaf } \text{W}$
 $\text{I} : \forall (f : \mathcal{F}) \rightarrow \text{RelPresheaf} \Rightarrow \llbracket \text{args } f \rrbracket^* \llbracket \text{ret } f \rrbracket$

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:

$$\begin{aligned}
\pi_i &: \forall \{n\} \{\Gamma : \text{Ctx } n\} \\
&\rightarrow (i : \text{Fin } n) \\
&\rightarrow \text{RelPresheaf} \Rightarrow ([\Gamma]^*) [\text{Vec.lookup } \Gamma \ i] \\
\pi_i \ i &= \text{record} \{ \eta = \text{lookup } i \\
&\quad ; \text{imply} = \text{lookup-zip } i \\
&\quad \}
\end{aligned}$$

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.

$$\begin{aligned}
\langle _ \rangle^* &: \forall \{n \ m\} \{\Gamma : \text{Ctx } n\} \{\Gamma' : \text{Ctx } m\} \\
&\rightarrow \text{mapT } (\lambda \tau \rightarrow \text{RelPresheaf} \Rightarrow ([\Gamma]^*) [\tau]) \Gamma' \\
&\rightarrow \text{RelPresheaf} \Rightarrow ([\Gamma]^*) ([\Gamma']^*) \\
\langle _ \rangle^* \{\Gamma' = []\}^* &= \\
&\quad \text{record} \{ \eta = \lambda _ \rightarrow * \\
&\quad ; \text{imply} = \lambda _ \rightarrow * \\
&\quad \} \\
\langle _ \rangle^* \{\Gamma' = _ :: _ \} (x, xs) &= \\
&\quad \text{let module } x = \text{RelPresheaf} \Rightarrow x \\
&\quad \quad \text{module } xs = \text{RelPresheaf} \Rightarrow (\langle xs \rangle^*) \\
&\quad \text{in record} \{ \eta = \langle x.\eta, xs.\eta \rangle \\
&\quad ; \text{imply} = \langle x.\text{imply}, xs.\text{imply} \rangle \\
&\quad \}
\end{aligned}$$

Finally, following [Definition 3.21](#) we have the relational morphism associated to a term, by induction on the term structure.

$$\begin{aligned}
[_]^\dagger &: \forall \{i \ n \ \tau\} \{\Gamma : \text{Ctx } n\} \\
&\rightarrow \Gamma \vdash \tau \langle i \rangle \\
&\rightarrow \text{RelPresheaf} \Rightarrow ([\Gamma]^*) [\tau] \\
[\text{var } i]^\dagger &= \pi_i \ i
\end{aligned}$$

$$\llbracket \text{fun } fx \rrbracket^t = \text{! } f \circ \langle \text{map } \llbracket _ \rrbracket^t x \rangle^*$$

4.8 Temporal structure

The last piece of data for our models is the notion of temporal structure on a category \mathcal{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 \mathbf{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 $arr \in \mathbf{T}$ indicating that arr is selected by the temporal structure \mathbf{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 can 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 $\text{compose} \leq p$ obtained by composing the first i arrows together. Note 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

```

4.9 From classical to categorical models

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

```

record TemporalCounterpartWModel {ℓ} (Σ : Signature {ℓ}) : Set (suc 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 [Theorem 3.1](#). Since our logic LTL is defined using the categorical presentation with presheaf semantics, this procedure will be used in

Section 4.12 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 a similar idea to Theorem 3.1, and we elucidate the four fields \mathbf{W} , $\llbracket _ \rrbracket$, \mathbf{I} , and \mathbf{T} provided by the construction.

```

ClassicalToCategorical : CounterpartModel Σ
  → TemporalCounterpartWModel Σ
ClassicalToCategorical M =

```

Given a classical model, the category \mathbf{W} is given by the free category (indicated here by `PathCategory`) induced by the set of worlds \mathbf{W} of the model with the accessibility relation $_ \rightsquigarrow _$ on it.

```

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

```

For any sort τ , its corresponding presheaf $\llbracket \tau \rrbracket$ takes each world ω to the set of objects of the algebra $\mathbf{d} \omega$. Similarly, for any function symbol \mathcal{F} , the relational morphism \mathbf{I} takes each world ω to the corresponding function given by the algebra $\mathbf{d} \omega$ on the symbol \mathcal{F} . In order to show that this is a proper relational morphism, an additional lemma `star-impl` 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 \mathbf{T} associated to this model is a simple predicate that returns the unit type \mathbf{T} for all morphisms of the free category with length *exactly* one. The empty type \perp representing falsity is given in all other cases.

```

; T = TStructure
  λ { ε → ⊥
    ; (⊥ < ε) → T
    ; (⊥ < (⊥ < ⊥)) → ⊥
    }
}

```

The action on arrows of the relational presheaf $[\tau]$ is given by the function $\mathbf{StarRel}$, 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

```

$\text{StarRel } \varepsilon = _ \equiv _$
 $\text{StarRel } (B \rightsquigarrow C \triangleleft C \rightsquigarrow^* A) = \text{StarRel } C \rightsquigarrow^* A ; \text{flip } (\rho (f B \rightsquigarrow C))$

where the base case is the identity relation $_ \equiv _$ 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 \mathbf{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.

$\text{star-homomorphism} : \forall \{ \tau \ X \ Y \ Z \} \{ g : \text{Star } _ \rightsquigarrow _ \ Y \ X \} \{ f : \text{Star } _ \rightsquigarrow _ \ Z \ Y \}$
 $\rightarrow \text{StarRel } \{ \tau \} (g \triangleright \triangleright f) \approx \text{StarRel } \{ \tau \} f \circ \text{StarRel } \{ \tau \} g$

$\text{star-imply} : \forall \{ \mathcal{F} \ \sigma \ \tau \ t \ s \} f$
 $\rightarrow \text{zip } (\text{StarRel } f) \ t \ s$
 $\rightarrow \text{StarRel } f (\mathbf{F} (\mathbf{d} \ \tau) \ \mathcal{F} \ t) (\mathbf{F} (\mathbf{d} \ \sigma) \ \mathcal{F} \ s)$

$\text{star-resp-}\approx^* : \forall \{ \tau \} \{ A \ B \} \{ f \ g : \text{Star } _ \rightsquigarrow _ \ B \ A \}$
 $\rightarrow f \approx^* g$
 $\rightarrow \text{Rels } \ell \ \ell \ [\text{StarRel } \{ \tau \} f \approx \text{StarRel } \{ \tau \} g]$

In these last lemmas the function $_ \triangleright \triangleright _$ and the relation $_ \approx^* _$ indicate composition and morphism equality in the `PathCategory`, respectively.

4.10 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 3.8](#). 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 $\exists$  :  $\forall \{\omega\} \rightarrow \text{Path } \omega \infty \rightarrow X_0 \omega \rightarrow \text{ClassicalAttribute } X \rightarrow \mathbb{N} \rightarrow \text{Set } _$ 
  at $\exists$   $p \ s \ A \ i = \exists [z] X_1 (\text{compose} \leq p \ i) \ z \ s \times z \in A$ 

  – Shorthand for:
  – ”All counterparts of  $s$  in the path  $p$ 
  – after  $i$  steps satisfy  $A$ ”
  at $\forall$  :  $\forall \{\omega\} \rightarrow \text{Path } \omega \infty \rightarrow X_0 \omega \rightarrow \text{ClassicalAttribute } X \rightarrow \mathbb{N} \rightarrow \text{Set } _$ 
  at $\forall$   $p \ s \ A \ i = \forall z \rightarrow X_1 (\text{compose} \leq p \ i) \ z \ s \rightarrow z \in A$ 

```

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

```

XO : ClassicalAttribute X  $\rightarrow$  ClassicalAttribute X
XO  $A \ s = \forall \{\sigma\}$ 
   $\rightarrow (\rho : _ \Rightarrow \sigma)$ 
   $\rightarrow \{\rho \in \mathbf{T}\}$ 
   $\rightarrow \exists [z] X_1 \rho \ z \ s \times s \in A$ 

XA : ClassicalAttribute X  $\rightarrow$  ClassicalAttribute X
XA  $A \ s = \forall \{\sigma\}$ 
   $\rightarrow (\rho : _ \Rightarrow \sigma)$ 
   $\rightarrow \{\rho \in \mathbf{T}\}$ 
   $\rightarrow \forall z \rightarrow X_1 \rho \ z \ s \rightarrow s \in A$ 

```

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**
 $_before_ : \forall \{\ell\} (A : \text{Pred } \mathbb{N} \ell) \rightarrow \text{Pred } \mathbb{N} \ell$
 $A \text{ before } n = \forall i \rightarrow i < n \rightarrow i \in A$
- **A holds until B is satisfied**
 $_until_ : \forall \{\ell\} (A B : \text{Pred } \mathbb{N} \ell) \rightarrow \text{Set } \ell$
 $A \text{ until } B = \exists [n] (A \text{ before } n \times n \in B)$
- **A is always satisfied at each step**
 $\text{always} : \forall \{\ell\} (A : \text{Pred } \mathbb{N} \ell) \rightarrow \text{Set } \ell$
 $\text{always } A = \forall i \rightarrow i \in A$
- **Either until or always hold**
 $_weakUntil_ : \forall \{\ell\} (A B : \text{Pred } \mathbb{N} \ell) \rightarrow \text{Set } \ell$
 $A \text{ weakUntil } B = A \text{ until } B \uplus \text{always } A$

Finally, we define the classical attributes associated to each operator by combining the previous shorthands to provide each possible operator. The predicates $\text{at}\exists p s A$ and $\text{at}\forall p s A$ are curried over the number of steps i , and are thus viewed as predicates on integers \mathbb{N} .

- $XU : \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X$
 $XU A B \{\omega\} s = \forall (p : \text{Path } \omega \infty) \rightarrow (\text{at}\exists p s A) \text{ until } (\text{at}\exists p s B)$
- $XF : \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X$
 $XF A B \{\omega\} s = \forall (p : \text{Path } \omega \infty) \rightarrow (\text{at}\forall p s A) \text{ until } (\text{at}\forall p s B)$
- $XW : \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X$
 $XW A B \{\omega\} s = \forall (p : \text{Path } \omega \infty) \rightarrow (\text{at}\exists p s A) \text{ weakUntil } (\text{at}\exists p s B)$
- $XT : \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X \rightarrow \text{ClassicalAttribute } X$
 $XT A B \{\omega\} s = \forall (p : \text{Path } \omega \infty) \rightarrow (\text{at}\forall p s A) \text{ weakUntil } (\text{at}\forall p s B)$

4.11 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 QTLTL by explicitly providing the entire set of operators as well as negation.

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

The type of QTLTL formulae `QTLTL` 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 QTLTL {n} (Γ : Ctx n) : Set ℓ where
  true  : QTLTL Γ
  false : QTLTL Γ
  !_    : QTLTL Γ → QTLTL Γ
  _^_   : QTLTL Γ → QTLTL Γ → QTLTL Γ
  _v_   : QTLTL Γ → QTLTL Γ → QTLTL Γ
  O_    : QTLTL Γ → QTLTL Γ
  A_    : QTLTL Γ → QTLTL Γ
  _F_   : QTLTL Γ → QTLTL Γ → QTLTL Γ
  _U_   : QTLTL Γ → QTLTL Γ → QTLTL Γ
  _W_   : QTLTL Γ → QTLTL Γ → QTLTL Γ
  _T_   : QTLTL Γ → QTLTL Γ → QTLTL 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 subformula to refer to it.

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

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

```
_≡t_ : ∀ {i τ}
```


$$\begin{aligned}
& \rightarrow \Gamma \vdash \tau \langle i \rangle \\
& \rightarrow \Gamma \vdash \tau \langle i \rangle \\
& \rightarrow \text{QLTL } \Gamma \\
\text{\textcolor{blue}{_}\#^t_} & : \forall \{i \tau\} \\
& \rightarrow \Gamma \vdash \tau \langle i \rangle \\
& \rightarrow \Gamma \vdash \tau \langle i \rangle \\
& \rightarrow \text{QLTL } \Gamma
\end{aligned}$$

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

$$\begin{aligned}
\Diamond_ & : \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{QLTL } \Gamma \\
\Diamond \phi & = \text{true } \mathbf{U} \phi \\
\Box_ & : \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{QLTL } \Gamma \\
\Box \phi & = \phi \mathbf{W} \text{false} \\
\Diamond^*_ & : \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{QLTL } \Gamma \\
\Diamond^* \phi & = \text{true } \mathbf{F} \phi \\
\Box^*_ & : \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{QLTL } \Gamma \\
\Box^* \phi & = \phi \mathbf{T} \text{false}
\end{aligned}$$

The semantics of LTL 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 $\llbracket \Gamma \rrbracket^*$.

$$\begin{aligned}
\langle _ \rangle & : \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{ClassicalAttribute } (\llbracket \Gamma \rrbracket^*) \\
\langle \text{true} \rangle & a = \top \\
\langle \text{false} \rangle & a = \perp \\
\langle ! \phi \rangle & a = \neg \langle \phi \rangle a \\
\langle \phi_1 \wedge \phi_2 \rangle & a = \langle \phi_1 \rangle a \times \langle \phi_2 \rangle a \\
\langle \phi_1 \vee \phi_2 \rangle & a = \langle \phi_1 \rangle a \uplus \langle \phi_2 \rangle a \\
\langle \exists < \tau > \phi \rangle & a = \exists [b] \langle \phi \rangle (b, a) \\
\langle \forall < \tau > \phi \rangle & a = \forall b \rightarrow \langle \phi \rangle (b, a) \\
\langle t_1 \equiv^t t_2 \rangle & a = \eta (\llbracket t_1 \rrbracket^t) a = \eta (\llbracket t_2 \rrbracket^t) a \\
\langle t_1 \neq^t t_2 \rangle & a = \eta (\llbracket t_1 \rrbracket^t) a \neq \eta (\llbracket t_2 \rrbracket^t) a
\end{aligned}$$

$$\begin{aligned}
\langle \text{O } \phi \rangle &= \text{XO } (\llbracket _ \rrbracket^*) \langle \phi \rangle \\
\langle \text{A } \phi \rangle &= \text{XA } (\llbracket _ \rrbracket^*) \langle \phi \rangle \\
\langle \phi_1 \text{ U } \phi_2 \rangle &= \text{XU } (\llbracket _ \rrbracket^*) \langle \phi_1 \rangle \langle \phi_2 \rangle \\
\langle \phi_1 \text{ F } \phi_2 \rangle &= \text{XF } (\llbracket _ \rrbracket^*) \langle \phi_1 \rangle \langle \phi_2 \rangle \\
\langle \phi_1 \text{ W } \phi_2 \rangle &= \text{XW } (\llbracket _ \rrbracket^*) \langle \phi_1 \rangle \langle \phi_2 \rangle \\
\langle \phi_1 \text{ T } \phi_2 \rangle &= \text{XT } (\llbracket _ \rrbracket^*) \langle \phi_1 \rangle \langle \phi_2 \rangle
\end{aligned}$$

In order to provide examples of satisfiability of a formula, it is useful to decide its validity in all possible worlds and choice of individuals:

$$\begin{aligned}
\text{DecidableFormula} &: \forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } \Gamma \rightarrow \text{Set } \ell \\
\text{DecidableFormula } \{_ \} \{\Gamma\} \phi &= \forall \omega (a : \text{F}_0 (\llbracket \Gamma \rrbracket^*) \omega) \rightarrow \text{Dec } (a \in \langle \phi \rangle)
\end{aligned}$$

4.12 Examples

We show how the running example presented in Figure 3.4 can be codified in Agda and how we can prove the validity of formulae on such a model. First, we define the signature of graphs.

```

module Example where

data Gr-Sorts : Set where
  Edge : Gr-Sorts
  Node : Gr-Sorts

data Gr-Functions : Set where
  s : Gr-Functions
  t : Gr-Functions

Gr : Signature
Gr = record {  $\mathcal{S}$  = Gr-Sorts
             ;  $\mathcal{F}$  = Gr-Functions
             ; sign $\mathcal{F}$  =  $\lambda \{ s \rightarrow [ \text{Edge} ] \mapsto \text{Node}$ 
                       ;  $t \rightarrow [ \text{Edge} ] \mapsto \text{Node}$ 
                       }
             } where open import Data.Vec using ([_])

```

For convenience, we report here in Figure 4.1 a copy of the running example presented in Figure 3.4.

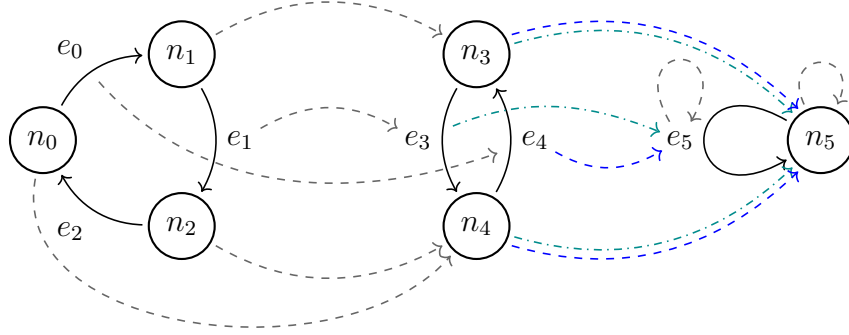


Figure 4.1: Running example of counterpart \mathcal{W} -model formalized in Agda.

We show a simple example of algebra by formalizing the first graph G_0 , omitting the rest of graphs G_1 , G_2 . First, we define the two sets of edges and nodes, and then specify the concrete s and t functions between them. Note that $*$ denotes the singleton type, since an algebra function folds over the signature of a function with the singleton as base case with `mapT`.

```
data G0-Edges : Set where e0 e1 e2 : G0-Edges
data G0-Nodes : Set where n0 n1 n2 : G0-Nodes

G0 : Σ-Algebra Gr
G0 = record { S = λ { Edge → G0-Edges ; Node → G0-Nodes }
              ; F = λ { s → λ { (e0 , *) → n0
                                ; (e1 , *) → n1
                                ; (e2 , *) → n2
                                }
                        ; t → λ { (e0 , *) → n1
                                ; (e1 , *) → n2
                                ; (e2 , *) → n0
                                }
                        }
              }
```

A homomorphism F_0 between graphs is specified by providing two relations between edges F_0 -Edges and nodes F_0 -Nodes, along with a proof ρ -homo that these functions are structure-preserving. This latter proof must be done by case analysis on all possible arguments of the function. We again omit the remaining

relational homomorphisms F_1, F_2 .

```

data F0-Edges : G0-Edges → G1-Edges → Set where
  e0e4 : F0-Edges e0 e4
  e1e3 : F0-Edges e1 e3

data F0-Nodes : G0-Nodes → G1-Nodes → Set where
  n0n4 : F0-Nodes n0 n4
  n1n3 : F0-Nodes n1 n3
  n2n4 : F0-Nodes n2 n4

F0 : Σ-Rel G0 G1
F0 = record { ρ = λ { {Edge} → F0-Edges
                      ; {Node} → F0-Nodes
                      }
              ; ρ-homo = λ { s (e0e4 , *) → n0n4
                             ; s (e1e3 , *) → n1n3
                             ; t (e0e4 , *) → n1n3
                             ; t (e1e3 , *) → n2n4
                             }
              }

```

After having defined the algebras that will inhabit the classical model, we define the set of worlds W and the accessibility relation \rightsquigarrow between them. Each world is then associated to its corresponding algebra with the function d , and each inhabitant of the transition relation is paired with a relational homomorphism with f .

```

data W : Set where
  ω0 ω1 ω2 : W

data _rightsquigarrow_ : W → W → Set where
  f0 : ω0 rightsquigarrow ω1
  f1 f2 : ω1 rightsquigarrow ω2
  f3 : ω2 rightsquigarrow ω2

d : W → Σ-Algebra Gr
d ω0 = G0
d ω1 = G1

```

$d \omega_2 = G_2$

$f : \forall \{A B\} \rightarrow A \rightsquigarrow B \rightarrow \Sigma\text{-Rel } (d A) (d B)$

$f f_0 = F_0$

$f f_1 = F_1$

$f f_2 = F_2$

$f f_3 = F_3$

$M : \text{CounterpartModel Gr}$

$M = \text{record } \{ W = W ; d = d ; _ \rightsquigarrow _ = _ \rightsquigarrow _ ; f = f \}$

We now obtain the categorical model of the classical setting constructed so far, so that it can be used with the semantics of our logic based on classical attributes.

$\text{TWM} : \text{TemporalCounterpartWModel Gr}$

$\text{TWM} = \text{ClassicalToCategorical } M$

open QLTL TWM

open Terms Gr

The formulae presented in [Section 3.7](#) are formalized using de Bruijn indices to refer to the variables in the context of the formula. In order to make terms more readable we introduce some shorthands, such as $_ \$ _$ to indicate function application in terms, and $v0$ and $v1$ to refer to the first and second de Bruijn variables available in the context.

$\text{module ExampleFormulae where}$

$\text{open Data.Fin hiding } (\# _)$

$\text{infix 27 } _ \$ _$

$_ \$ _ : \forall \{s n\} \{ \Gamma : \text{Ctx } n \} f \rightarrow _ \rightarrow \Gamma \vdash _ \langle \uparrow s \rangle$

$_ \$ _ = \text{fun}$

$v0 : \forall \{n\} \{ \Gamma : \text{Ctx } (1 + n) \} \rightarrow \Gamma \vdash _ \langle \infty \rangle$

$v0 = \text{var zero}$

$v1 : \forall \{n\} \{ \Gamma : \text{Ctx } (2 + n) \} \rightarrow \Gamma \vdash _ \langle \infty \rangle$

$v1 = \text{var } (\text{suc zero})$

In the case of the formula `present`, for example, `v0` refers to the innermost existential binder and `v1` to the implicit free variable of the formula.

```

present :  $\forall \{\tau\} \rightarrow \text{QLTL } (\tau :: [])$ 
present  $\{\tau\} = \exists < \tau > v1 \equiv^t v0$ 

notPresent :  $\forall \{\tau\} \rightarrow \text{QLTL } (\tau :: [])$ 
notPresent  $\{\tau\} = \forall < \tau > v1 \not\equiv^t v0$ 

nextStepPreserved :  $\forall \{\tau\} \rightarrow \text{QLTL } (\tau :: [])$ 
nextStepPreserved = present  $\wedge$  O present

nextStepDeallocated :  $\forall \{\tau\} \rightarrow \text{QLTL } (\tau :: [])$ 
nextStepDeallocated = present  $\wedge$  A notPresent

loop :  $\forall \{n\} \{\Gamma : \text{Ctx } n\} \rightarrow \text{QLTL } (\text{Edge} :: \Gamma)$ 
loop = s $ (v0 , *)  $\equiv^t$  t $ (v0 , *)

hasLoop :  $\text{QLTL } []$ 
hasLoop =  $\exists < \text{Edge} >$  loop

nodeHasLoop :  $\text{QLTL } (\text{Node} :: [])$ 
nodeHasLoop =  $\exists < \text{Edge} > (s \$ (v0 , *) \equiv^t v1 \wedge \text{loop})$ 

willBecomeLoop :  $\text{QLTL } (\text{Edge} :: [])$ 
willBecomeLoop = ! loop  $\wedge$   $\diamond$  loop

eventuallyNodeHasLoop :  $\text{QLTL } (\text{Node} :: [])$ 
eventuallyNodeHasLoop =  $\diamond$  nodeHasLoop

```

Finally, we show that each formula identifies for each world a decidable subset of elements, i.e.: a decidable classical attribute. We can decide for each world and choice of individual if it satisfies the formula or not, and provide a concrete proof in each case. We use the shorthands `_ \Rightarrow` and `step` to construct and pattern match on one-step morphisms of the free category, respectively.

```

_ $\Rightarrow$  :  $\forall \{\ell \ell'\} \{A : \text{Set } \ell\} \{i j : A\} \{R : \text{Rel } A \ell'\} \rightarrow R i j \rightarrow \text{Star } R i j$ 
a  $\Rightarrow$  = a  $\triangleleft \varepsilon$ 

pattern step a = a  $\triangleleft \varepsilon$ 

```

We start by considering the deallocation of edges. We select for each world and possible assignment whether the formula can be satisfied or not with the **yes** and **no** constructors, and then provide the corresponding proof.

Consider the first case, where the formula is satisfied since the edge **e2** will be deallocated. To satisfy the formula we provide the following data: to the left of the conjunction constructor **,** we prove that **present** holds by giving a concrete witness of the existential and a proof **refl** stating that the two witness is definitionally equal to the individual we provided. On the right side of the conjunction, we prove that every counterpart satisfies **notPresent** since the only temporal development **f₀** provides no possible counterpart for **e2**.

The remaining cases follow the structure used to prove negations: we assume that the formula holds and derive a contradiction. We can derive a contradiction by providing a (one-step) temporal development and a counterpart for the element being considered to the assumption $A\neg p$ that $A\neg\text{present}(x)$ holds.

```
exampleNextStepDeallocated : DecidableFormula (nextStepDeallocated {Edge})
exampleNextStepDeallocated ω0 (e2 , *) =
  yes ((e2 , refl) , λ { (step f0) _ () _ _ })
exampleNextStepDeallocated ω0 (e0 , *) =
  no λ { ((e0 , refl) , A¬p)
    → A¬p (f0 ⇒) (e4 , *) ((e4 , refl , e0e4) , *) e0 refl }
exampleNextStepDeallocated ω0 (e1 , *) =
  no λ { ((e1 , refl) , A¬p)
    → A¬p (f0 ⇒) (e3 , *) ((e3 , refl , e1e3) , *) e1 refl }
exampleNextStepDeallocated ω1 (e3 , *) =
  no λ { ((e3 , refl) , A¬p)
    → A¬p (f1 ⇒) (e5 , *) ((e5 , refl , e3e51) , *) e3 refl }
exampleNextStepDeallocated ω1 (e4 , *) =
  no λ { ((e4 , refl) , A¬p)
    → A¬p (f2 ⇒) (e5 , *) ((e5 , refl , e4e52) , *) e4 refl }
exampleNextStepDeallocated ω2 (e5 , *) =
  no λ { ((e5 , refl) , A¬p)
    → A¬p (f3 ⇒) (e5 , *) ((e5 , refl , e5e5) , *) e5 refl }
```

In order to show that an edge is preserved by all one-step temporal developments, we again first show that the element is **present** and then give a function that shows that **O present** for each development. In the cases where we need to show that the formula is not satisfied, we derive a contradiction by providing the

non-preserving temporal development to the assumption Op that there exists a counterpart for each development.

In these examples we only need to pattern match on the one-step developments provided by the categorical model, since Agda recognizes that the (implicit) proof that the arrow is selected by the temporal structure can only be supplied when the arrow is a one-step morphism, and is the empty type in all other cases.

```

exampleNextStepPreserved : DecidableFormula (nextStepPreserved {Edge})
exampleNextStepPreserved  $\omega_0$  (e0 , *) =
  yes ((e0 , refl)
    ,  $\lambda$  { (step f0)  $\rightarrow$  (e4 , *)
      , ((e4 , refl , e0e4) , *)
      , e0 , refl })
exampleNextStepPreserved  $\omega_0$  (e1 , *) =
  yes ((e1 , refl)
    ,  $\lambda$  { (step f0)  $\rightarrow$  (e3 , *)
      , ((e3 , refl , e1e3) , *)
      , e1 , refl })
exampleNextStepPreserved  $\omega_0$  (e2 , *) =
  no  $\lambda$  { ((e2 , refl) , Op)  $\rightarrow$  absurd (Op (f0  $\Rightarrow$ )) }
  where absurd :  $\_ \rightarrow \_;$  absurd ()
exampleNextStepPreserved  $\omega_1$  (e3 , *) =
  no  $\lambda$  { ((e3 , refl) , Op)  $\rightarrow$  absurd (Op (f2  $\Rightarrow$ )) }
  where absurd :  $\_ \rightarrow \_;$  absurd ()
exampleNextStepPreserved  $\omega_1$  (e4 , *) =
  no  $\lambda$  { ((e4 , refl) , Op)  $\rightarrow$  absurd (Op (f1  $\Rightarrow$ )) }
  where absurd :  $\_ \rightarrow \_;$  absurd ()
exampleNextStepPreserved  $\omega_2$  (e5 , *) =
  yes ((e5 , refl)
    ,  $\lambda$  { (step f3)  $\rightarrow$  (e5 , *)
      , ((e5 , refl , e5e5) , *)
      , e5 , refl })

```

The simple formulae on loops are straightforward, and we note how in the case of the closed formula `hasLoop` we only need to consider empty assignments for each world.

```

exampleLoop : DecidableFormula (loop { $\Gamma$  = []})
exampleLoop  $\omega_0$  (e0 , *) = no ( $\lambda$  ())

```



```

exampleLoop  $\omega_0$  (e1 , *) = no ( $\lambda$  ())
exampleLoop  $\omega_0$  (e2 , *) = no ( $\lambda$  ())
exampleLoop  $\omega_1$  (e3 , *) = no ( $\lambda$  ())
exampleLoop  $\omega_1$  (e4 , *) = no ( $\lambda$  ())
exampleLoop  $\omega_2$  (e5 , *) = yes refl

exampleNodeHasLoop : DecidableFormula nodeHasLoop
exampleNodeHasLoop  $\omega_0$  (n0 , *) = no  $\lambda$  { (e0 , ()) ; (e1 , ()) ; (e2 , ()) }
exampleNodeHasLoop  $\omega_0$  (n1 , *) = no  $\lambda$  { (e0 , ()) ; (e1 , ()) ; (e2 , ()) }
exampleNodeHasLoop  $\omega_0$  (n2 , *) = no  $\lambda$  { (e0 , ()) ; (e1 , ()) ; (e2 , ()) }
exampleNodeHasLoop  $\omega_1$  (n3 , *) = no  $\lambda$  { (e3 , ()) ; (e4 , ()) }
exampleNodeHasLoop  $\omega_1$  (n4 , *) = no  $\lambda$  { (e3 , ()) ; (e4 , ()) }
exampleNodeHasLoop  $\omega_2$  (n5 , *) = yes (e5 , refl , refl)

exampleHasLoop : DecidableFormula hasLoop
exampleHasLoop  $\omega_0$  * = no  $\lambda$  { (e0 , ()) ; (e1 , ()) ; (e2 , ()) }
exampleHasLoop  $\omega_1$  * = no  $\lambda$  { (e3 , ()) ; (e4 , ()) }
exampleHasLoop  $\omega_2$  * = yes (e5 , refl)

```

Finally, we provide some examples on temporal formulae quantifying over paths. We start by defining the two paths of our model `path1` and `path2` depending on which morphism is selected as second step. Both paths end with the self-looping path `self2` in the last world.

```

self2 :  $\forall \{i\} \rightarrow \text{Path } \omega_2 i$ 
self2 = (f3  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  self2 }

path1 :  $\forall \{\omega i\} \rightarrow \text{Path } \omega i$ 
path1 { $\omega_0$ } = (f0  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  (f1  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  self2 } }
path1 { $\omega_1$ } = (f1  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  self2 }
path1 { $\omega_2$ } = self2

path2 :  $\forall \{\omega i\} \rightarrow \text{Path } \omega i$ 
path2 { $\omega_0$ } = (f0  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  (f2  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  self2 } }
path2 { $\omega_1$ } = (f2  $\Rightarrow$ )  $\rightarrow \lambda$  { .force  $\rightarrow$  self2 }
path2 { $\omega_2$ } = self2

```

As stated in [Section 3.7](#), we have that no edge will become a loop since it is either not preserved by a temporal development or it was already a loop in the case of `e5`.

The proofs all proceed by assuming that there exists an n for each time development such that the edge will become a loop and then deriving a contradiction. This is done by considering all possible cases of n that could be provided by the *eventually* operator, with the last case of **e5** disproving the formula since it already was a loop.

exampleWillBecomeLoop : DecidableFormula willBecomeLoop

exampleWillBecomeLoop =

```

λ { ω0 (e0 , *) → no ex0
  ; ω0 (e1 , *) → no ex1
  ; ω0 (e2 , *) → no ex2
  ; ω1 (e3 , *) → no ex3
  ; ω1 (e4 , *) → no ex4
  ; ω2 (e5 , *) → no ex5
}

```

where

ex0 : $\langle ! \text{willBecomeLoop} \rangle \{\omega_0\} (e_0 , *)$

ex0 ($\neg \text{loop}$, $\diamond \text{loop}$) with $\diamond \text{loop}$ path1

```

... | 0 , b , (e0 , *) , ()
... | 0 , b , (e1 , *) , ()
... | 0 , b , (e2 , *) , ()
... | 1 , b , (e3 , *) , ()
... | 1 , b , (e4 , *) , ()
... | suc (suc a) , b , _ , ((e3 , ()), *) , eq
... | suc (suc a) , b , _ , ((e4 , ()), *) , eq

```

ex1 : $\langle ! \text{willBecomeLoop} \rangle \{\omega_0\} (e_1 , *)$

ex1 ($\neg \text{loop}$, $\diamond \text{loop}$) with $\diamond \text{loop}$ path2

```

... | 0 , b , (e0 , *) , ()
... | 0 , b , (e1 , *) , ()
... | 0 , b , (e2 , *) , ()
... | 1 , b , (e3 , *) , ()
... | 1 , b , (e4 , *) , ()
... | suc (suc a) , b , _ , ((e3 , ()), *) , eq
... | suc (suc a) , b , _ , ((e4 , ()), *) , eq

```

ex2 : $\langle ! \text{willBecomeLoop} \rangle \{\omega_0\} (e_2 , *)$

ex2 ($\neg \text{loop}$, $\diamond \text{loop}$) with $\diamond \text{loop}$ path1

```

... | 0 , p , (e2 , *) , a , n2≡n0 = ⊥-elim (¬loop n2≡n0)

```

```

ex3 : ⟨ ! willBecomeLoop ⟩ {ω1} (e3 , *)
ex3 (¬loop , ◇loop) with ◇loop path2
... | 0 , p , (e4 , *) , a , n4≡n3 = ⊥-elim (¬loop (sym n4≡n3))
... | 0 , p , (e3 , *) , a , n3≡n4 = ⊥-elim (¬loop n3≡n4)

ex4 : ⟨ ! willBecomeLoop ⟩ {ω1} (e4 , *)
ex4 (¬loop , ◇loop) with ◇loop path1
... | 0 , p , (e3 , *) , a , n3≡n4 = ⊥-elim (¬loop (sym n3≡n4))
... | 0 , p , (e4 , *) , a , n4≡n3 = ⊥-elim (¬loop n4≡n3)

ex5 : ⟨ ! willBecomeLoop ⟩ {ω2} (e5 , *)
ex5 (¬loop , _) = ¬loop refl

```

Finally, we show how the *eventually* operator can be validated by providing a concrete number of steps n after which the formula is satisfied and by considering all intermediate counterparts for the path provided. For example, in the case of **n4** we pattern match on the one-step morphism given by the path and then provide the necessary data to show that a counterpart exists for each step of the path, as well as in the n -th world.

```

exampleEventuallyHasLoop : DecidableFormula eventuallyNodeHasLoop
exampleEventuallyHasLoop =
  λ { ω0 (n0 , *) → yes ex0
    ; ω0 (n1 , *) → yes ex1
    ; ω0 (n2 , *) → yes ex2
    ; ω1 (n3 , *) → yes ex3
    ; ω1 (n4 , *) → yes ex4
    ; ω2 (n5 , *) → yes ex5
  }
where
  ex5 : ⟨ eventuallyNodeHasLoop ⟩ {ω2} (n5 , *)
  ex5 (step f3 → p) =
    0 , (λ { (suc i) () } , (n5 , *) , ((refl , *) , e5 , refl , refl))

  ex4 : ⟨ eventuallyNodeHasLoop ⟩ {ω1} (n4 , *)
  ex4 (step f1 → p) with p.force
  ... | step f3 → p =
    1 , (λ { 0 (s ≤ s z ≤ n) → ((n4 , *) , (refl , *) , *) })

```

```

      , (n5 , *), (((n5 , (refl , n4n51)) , *), (e5 , (refl , refl)))
ex4 (step f2 → p) with p.force
... | step f3 → p =
  1 , (λ { 0 (s ≤ s z ≤ n) → ((n4 , *), (refl , *), *) })
    , (n5 , *), (((n5 , (refl , n4n52)) , *), (e5 , (refl , refl)))

ex3 : ⟨ eventuallyNodeHasLoop ⟩ {ω1} (n3 , *)
ex3 (step f1 → p) with p.force
... | step f3 → p =
  1 , (λ { 0 (s ≤ s z ≤ n) → ((n3 , *), (refl , *), *) })
    , (n5 , *), (((n5 , (refl , n3n51)) , *), (e5 , (refl , refl)))
ex3 (step f2 → p) with p.force
... | step f3 → p =
  1 , (λ { 0 (s ≤ s z ≤ n) → ((n3 , *), (refl , *), *) })
    , (n5 , *), (((n5 , (refl , n3n52)) , *), (e5 , (refl , refl)))

```

Notice how we can even reuse the proofs given for **n3** and **n4** in the cases of nodes preceding them temporally. This is useful in order to avoid having to consider a quadratic number of steps in the proof that a counterpart exists at each intermediate point, which now only needs to consider the base **0** case with the successor case being left to the other proof.

```

ex2 : ⟨ eventuallyNodeHasLoop ⟩ {ω0} (n2 , *)
ex2 (step f0 → p) with ex4 (p.force)
... | n , u , p , (k , *), m =
  ℕ.suc n
  , (λ { 0 (s ≤ s z ≤ n) → (n2 , *), (refl , *), *
    ; (suc i) (s ≤ s x) →
      let u1 , ((up , _) , _) = u i x
      in u1 , ((n4 , up , n2n4) , *), * })
    , p
    , ((n4 , k , n2n4) , *), m

ex1 : ⟨ eventuallyNodeHasLoop ⟩ {ω0} (n1 , *)
ex1 (step f0 → p) with ex3 (p.force)
... | n , u , p , (k , *), m =
  ℕ.suc n
  , (λ { 0 (s ≤ s z ≤ n) → (n1 , *), (refl , *), *
    ; (suc i) (s ≤ s x) →

```

```

      let u1, ((up, _) , _) = u i x
      in u1, ((n3, up, n1n3) , * ) , * })
    , p
    , ((n3, k, n1n3) , * ) , m

ex0 : ⟨ eventuallyNodeHasLoop ⟩ {ω0} (n0 , *)
ex0 (step f0 → p) with ex4 (p.force)
... | n , u , p , (k , * ) , m =
  ℕ.suc n
  , (λ { 0 (s ≤ s z ≤ n) → (n0 , * ) , (refl , * ) , *
    ; (suc i) (s ≤ s x) →
      let u1, ((up, _) , _) = u i x
      in u1, ((n4, up, n0n4) , * ) , * })
  , p
  , ((n4, k, n0n4) , * ) , m

```

Chapter 5

Conclusion

We have seen how a classic set-based semantics and a categorical semantics for a first-order linear temporal logic QLTL 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 formalized and practically experimented with in a proof assistant based on dependent type theory such as Agda.

5.1 Related work

The counterpart semantics of counterpart-based temporal logics is explored in the context of a μ -calculus with fixpoints in [GLV12]. The categorical semantics of quantified temporal logics and their models based is introduced in [GT21], exploiting the counterpart paradigm.

The formalization of temporal logics in (constructive) proof assistants has a long history, see for example [Cou03; Spr98; ZLS12]. A practical application and comparison with modern model checkers is given in [Esp+14], where a fully verified LTL model checker is implemented in the Isabelle theorem prover. In [O’C16], a verified proof-search program is formalized in Agda for the case of standard CTL, where they develop a toolbox to implement well-typed proof-searching procedures; a similar embedding of constructive LTL in Agda is given in [Jef12] in the context of the verification of functional reactive programs.

5.2 Future work

We identify a variety of possible expansions and future works for this thesis.

- **Second-order.** Our theoretical presentation and formalization work focuses on the first-order aspect of QLTL. The semantics given in [GT21; GLV12], however, allows for the quantification over sets of elements to be expressed. This notion is impractical to define in Agda due to the typical formalization 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 of this work 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 [Eme90], by considering branching models and building more complex temporal structures on the notion of category. We believe that extending our logic to the case of CTL and more complex models would be a straightforward task, which might however cause a combinatorial explosion in the case of possible temporal operators required to obtain 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 in order 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 [O’C16; Esp+14; KS15]. 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 specialized program. A possible extension of this work would be the implementation of either of these mechanisms to the setting of counterpart models and their semantics.

- **Category theory.** The category theoretical notions formalized in our work constitute a small part of the mechanization, 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 formalize our notions of models and assignments in terms of morphisms between suitable indexed categories [Jac01].

Bibliography

- [Lew68] D. K. Lewis. “Counterpart theory and quantified modal logic”. *The Journal of Philosophy* 65.5 (1968), pp. 113–126.
- [Law69] F. W. Lawvere. “Adjointness in foundations”. *Dialectica* 23 (1969), pp. 281–296.
- [Pnu77] A. Pnueli. “The temporal logic of programs”. *Foundations of Computer Science*. IEEE Computer Society, 1977, pp. 46–57.
- [Haz79] A. Hazen. “Counterpart-theoretic semantics for modal logic”. *The Journal of Philosophy* 76.6 (1979), pp. 319–338.
- [RS85] J. H. Reif and A. P. Sistla. “A multiprocess network logic with temporal and spatial modalities”. *Journal of Computer and System Science* 30.1 (1985), pp. 41–53.
- [GM88] S. Ghilardi and G. Meloni. “Modal and tense predicate logic: Models in presheaves and categorical conceptualization”. *Categorical Algebra and its Applications*. Ed. by F. Borceux. Vol. 1348. Lecture Notes in Mathematics. Springer, 1988, pp. 130–142.
- [GLT89] J. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1989.
- [Cou90] B. Courcelle. “The monadic second-order logic of graphs. I. Recognizable sets of finite graphs”. *Information and Computation* 85.1 (1990), pp. 12–75.
- [Eme90] E. A. Emerson. “Temporal and Modal Logic”. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Ed. by Jan van Leeuwen. Elsevier and MIT Press, 1990, pp. 995–1072.
- [GM96] S. Ghilardi and G. Meloni. “Relational and partial variable sets and basic predicate logic”. *Journal of Symbolic Logic* 61.3 (1996), pp. 843–872.

- [Spr98] C. Sprenger. “A Verified Model Checker for the Modal μ -calculus in Coq”. *Tools and Algorithms for Construction and Analysis of Systems*. Vol. 1384. Lecture Notes in Computer Science. Springer, 1998, pp. 167–183.
- [Cou00] B. Courcelle. “The monadic second-order logic of graphs. XII. Planar graphs and planar maps”. *Theoretical Computer Science* 237.1 (2000), pp. 1–32.
- [HWZ01] I. M. Hodkinson, F. Wolter, and M. Zakharyashev. “Monodic fragments of first-order temporal logics: 2000-2001 A.D”. *Logic for Programming, Artificial Intelligence and Reasoning*. Ed. by R. Nieuwenhuis and A. Voronkov. Vol. 2250. LNCS. Springer, 2001, pp. 1–23.
- [Jac01] B. P. F. Jacobs. *Categorical Logic and Type Theory*. Vol. 141. Studies in logic and the foundations of mathematics. North-Holland, 2001.
- [CGG02] L. Cardelli, P. Gardner, and G. Ghelli. “A spatial logic for querying graphs”. *International Colloquium on Automata, Languages and Programming*. Ed. by P. Widmayer et al. Springer, 2002, pp. 597–610.
- [NPW02] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media, 2002.
- [Cou03] S. Coupet-Grimal. “An Axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions”. *Journal of Logic and Computation* 13.6 (2003), pp. 801–813.
- [FT03] E. Franconi and D. Toman. “Fixpoint extensions of temporal description logics”. *DL 2003*. Vol. 81. CEUR Workshop Proceedings. 2003.
- [Bel04] F. Belardinelli. “Quantified Modal Logic and the Ontology of Physical Objects”. PhD thesis. Scuola Normale Superiore of Pisa, 2004-2005.
- [Bus+05] D. Bustan et al. “Regular Vacuity”. *Correct Hardware Design and Verification Methods*. Ed. by D. Borriore and W. J. Paul. Vol. 3725. LNCS. Springer, 2005, pp. 191–206.
- [LB06] F. Lindblad and M. Benke. “A Tool for Automated Theorem Proving in Agda”. *Types for Proofs and Programs*. 2006, pp. 154–169.
- [BBW07] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter, eds. *Handbook of Modal Logic*. Vol. 3. North-Holland, 2007.

- [DGG07] A. Dawar, P. Gardner, and G. Ghelli. “Expressiveness and complexity of graph logic”. *Information and Computation* 205.3 (2007), pp. 263–310.
- [Nor09] U. Norell. “Dependently typed programming in Agda”. *Types in Language Design and Implementation*. Ed. by A. Kennedy and A. Ahmed. ACM, 2009, pp. 1–2.
- [Awo10] S. Awodey. *Category Theory*. USA: Oxford University Press, Inc., 2010.
- [GLV12] F. Gadducci, A. Lluch-Lafuente, and A. Vandin. “Counterpart semantics for a second-order μ -calculus”. *Fundamenta Informaticae* 118.1-2 (2012), pp. 177–205.
- [Jef12] A. Jeffrey. “LTL types FRP: linear-time temporal logic propositions as types, proofs as functional reactive programs”. *Programming Languages meets Program Verification*. ACM, 2012, pp. 49–60.
- [ZLS12] D. Zanarini, C. Luna, and L. Sierra. “Alternating-Time Temporal Logic in the Calculus of (Co)Inductive Constructions”. *Formal Methods: Foundations and Applications - 15th Brazilian Symposium*. Vol. 7498. Lecture Notes in Computer Science. Springer, 2012, pp. 210–225.
- [Esp+14] J. Esparza et al. “A Fully Verified Executable LTL Model Checker”. *Archive of Formal Proofs* (2014).
- [Lei14] Tom Leinster. *Basic Category Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2014.
- [KS15] W. Kokke and W. Swierstra. “Auto in Agda - Programming Proof Search Using Reflection”. *Mathematics of Program Construction*. Vol. 9129. Lecture Notes in Computer Science. Springer, 2015, pp. 276–301.
- [Coq16] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. 2016.
- [O’C16] L. O’Connor. “Applications of applicative proof search”. *International Workshop on Type-Driven Development*. ACM, 2016, pp. 43–55.
- [GT21] F. Gadducci and D. Trotta. “A Presheaf Semantics for Quantified Temporal Logics”. *CoRR* abs/2111.03855 (2021). arXiv: [2111.03855](https://arxiv.org/abs/2111.03855).
- [HC21] J. Z. S. Hu and J. Carette. “Formalizing category theory in Agda”. *International Conference on Certified Programs and Proofs*. ACM, 2021, pp. 327–342.

- [MU21] L. de Moura and S. Ullrich. “The Lean 4 Theorem Prover and Programming Language”. *International Conference on Automated Deduction*. Vol. 12699. Lecture Notes in Computer Science. Springer, 2021, pp. 625–635.
- [HC22] S. Huang and R. Cleaveland. “A tableau construction for finite linear-time temporal logic”. *Journal of Logic and Algebraic Methods in Programming* 125 (2022), p. 100743.
- [WKS22] P. Wadler, W. Kokke, and J. G. Siek. *Programming Language Foundations in Agda*. Aug. 2022. URL: <https://plfa.inf.ed.ac.uk/20.08/>.

Appendix A

Category Theory

In this chapter we introduce some basic notions of category theory required in this thesis. For a more complete reference on category theory, consult for example [Awo10; Lei14].

Definition A.1 (Category). A **category** \mathcal{C} consists of the following data:

- a collection $\text{Obj}(\mathcal{C})$ of *objects*;
- for any two objects A, B , a collection $\mathcal{C}(A, B)$ of *morphisms* from A to B ;
- for every object $A \in \text{Obj}(\mathcal{C})$, there is a morphism $\text{id}_A \in \mathcal{C}(A, A)$ called the *identity morphism* on A ;
- for every three objects $A, B, C \in \text{Obj}(\mathcal{C})$ and morphisms $f \in \mathcal{C}(A, B)$ and $g \in \mathcal{C}(B, C)$, there is a morphism $f; g \in \mathcal{C}(A, C)$ called the *composite of f and g* .

We will shorten $A \in \text{Obj}(\mathcal{C})$ to simply $A \in \mathcal{C}$, and similarly $f \in \mathcal{C}(A, B)$ will be indicated as $f : A \rightarrow B$.

These constituents of a category are required to satisfy two conditions:

1. for any morphism $f : A \rightarrow B$, we have that:

$$\text{id}_A; f = f \quad f; \text{id}_B = f$$

2. for any morphisms $f : A \rightarrow B, g : B \rightarrow C, h : B \rightarrow C$, we have that:

$$(f; g); h = f; (g; h)$$

which allows us to write $f; g; h$ unambiguously.

A category can be graphically represented by showing the objects and the morphisms between them, with the composition of morphisms and the identity morphisms often not displayed. We show an example for a simple category \mathcal{C} in Figure A.1.

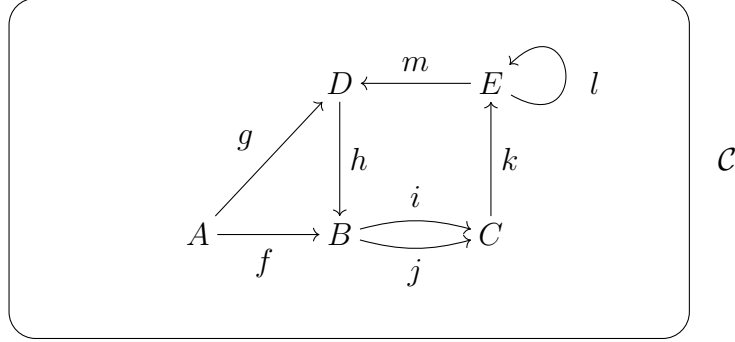


Figure A.1: An example of a category \mathcal{C} with $\text{Obj}(\mathcal{C}) = \{A, B, C, D, E\}$ and some morphisms.

Example A.1. We provide here some standard examples of categories, some of which will be used later:

- The category **Set** where objects are sets and morphisms are functions;
- The category **Rel** where objects are sets and morphisms are relations;
- The category **Grp** where objects are groups and morphisms are group homomorphisms;
- The category **Grph** where objects are graphs and morphisms are graph homomorphisms;
- The category **Alg** where objects are algebras and morphisms are algebra homomorphisms;

Definition A.2 (Opposite category). Given a category \mathcal{C} , the **opposite category** \mathcal{C}^{op} is obtained by taking $\text{Obj}(\mathcal{C}^{op}) := \text{Obj}(\mathcal{C})$ and for any two objects $A, B \in \text{Obj}(\mathcal{C})$ one has $\mathcal{C}^{op}(A, B) := \mathcal{C}(B, A)$.

Definition A.3 (Functor). Given two categories \mathcal{C} and \mathcal{D} , a **functor from \mathcal{C} to \mathcal{D}** , denoted with $F : \mathcal{C} \rightarrow \mathcal{D}$, consists of the following data:

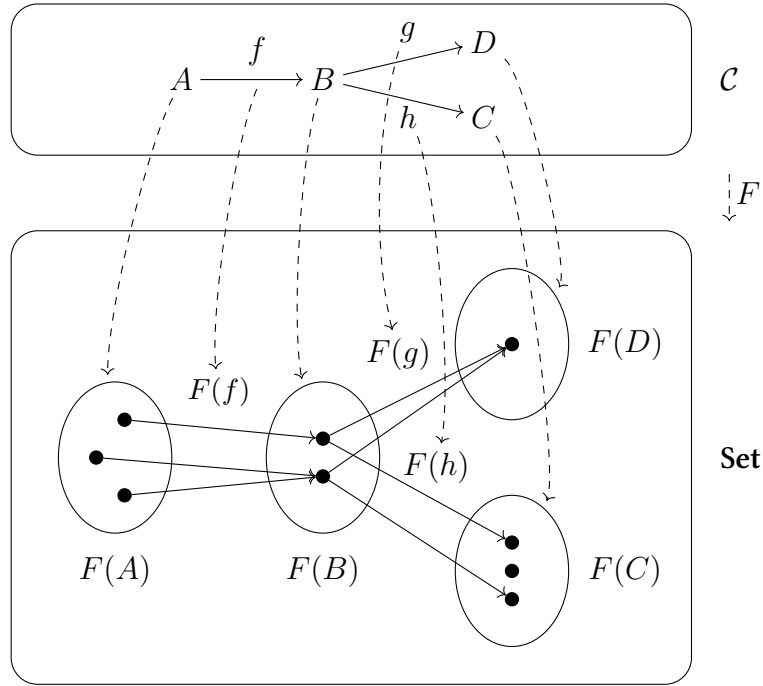


Figure A.2: An example of a functor F from a category \mathcal{C} to **Set**.

- for every object $A \in \text{Obj}(\mathcal{C})$, one specifies an object $F(A) \in \text{Obj}(\mathcal{D})$;
- for every morphism $f : A \rightarrow B$, one specifies a morphism $F(f) : F(A) \rightarrow F(B)$ in \mathcal{D} .

These constituents of a functor are required to satisfy two conditions:

1. for every object $A \in \text{Obj}(\mathcal{C})$ we have that:

$$F(\text{id}_A) = \text{id}_{F(A)}$$

2. for any morphisms $f : A \rightarrow B, g : B \rightarrow C$ we have the following equation in \mathcal{D} :

$$F(f; g) = F(f); F(g)$$

Example A.2. We show in [Figure A.2](#) a simple example of a functor \mathcal{F} from some category \mathcal{C} to the category **Set**.

Definition A.4 (Presheaf). Given a category \mathcal{C} , a **presheaf** is simply a functor $F : \mathcal{C}^{op} \rightarrow \mathbf{Set}$.

Definition A.5 (Natural transformation). Given two functors $F, G : \mathcal{C} \rightarrow \mathcal{D}$ defined on the same categories, a **natural transformation** α **from** F **to** G , denoted $\alpha : F \Rightarrow G$, is given by the following data:

- for each object $A \in \mathcal{D}$, a morphism $\alpha_A : F(A) \rightarrow G(A)$;
- for every morphism $f : A \rightarrow B$, we have that:

$$F(f); \alpha_B = \alpha_A; G(f)$$

Graphically, we can represent this equation as stating that the following diagram *commutes*:

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \alpha_A \downarrow & & \downarrow \alpha_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

Definition A.6 (Vertical composition). Let $F, G, H : \mathcal{C} \rightarrow \mathcal{D}$ be functors. Given two natural transformations $\alpha : F \Rightarrow G$ and $\beta : G \Rightarrow H$, their **vertical composition** $(\alpha; \beta) : F \Rightarrow H$ is a natural transformation defined by $(\alpha; \beta)_A := \alpha_A; \beta_A$ using composition in \mathcal{D} .

Example A.3 (Functor category). Given two categories \mathcal{C} and \mathcal{D} , the **functor category** $[\mathcal{C}, \mathcal{D}]$ is defined as follows:

- as objects, take the collection of all functors $F : \mathcal{C} \rightarrow \mathcal{D}$;
- morphisms between two functors F, G are given by the natural transformations $\alpha : F \Rightarrow G$;
- composition of morphisms is given by vertical composition.

Definition A.7 (Isomorphism). Let \mathcal{C} be a category. Two objects $A, B \in \mathcal{C}$ are said to be **isomorphic** if there exists two morphisms $f : A \rightarrow B$ and $g : B \rightarrow A$ such that $f; g = \text{id}_A$ and $g; f = \text{id}_B$.

Example A.4 (Product). Let \mathcal{C} be a category. Given two objects $A, B \in \mathcal{C}$, an object $A \times B \in \mathcal{C}$ is said to be a **product of A and B** if the following holds:

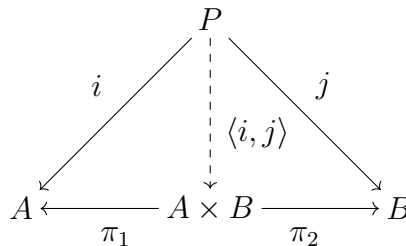
- two morphisms $\pi_1 : A \times B \rightarrow A$ and $\pi_2 : A \times B \rightarrow B$ are given;
- given any other object P with $i : P \rightarrow A$ and $j : P \rightarrow B$, there exists a unique morphism $\langle i, j \rangle$ such that:

$$\langle i, j \rangle; \pi_1 = i \qquad \langle i, j \rangle; \pi_2 = j$$

Explicitly, the uniqueness property is equivalent to stating that:

$$\forall p \in \mathcal{C}(P, A \times B). \quad p; \pi_1 = i \quad \wedge \quad p; \pi_2 = j \quad \implies \quad \langle i, j \rangle = p.$$

Graphically, this equates to making the following diagram commutes:



where the dotted line indicates that $\langle i, j \rangle$ is unique.

Proposition A.1 (Products and terminal objects are isomorphic). Given any two objects A, B , two products of A and B are unique up to isomorphism. This allows us to essentially speak of *the* terminal object and *the* product of two objects, whenever they exist.

Example A.5 (Terminal object). Let \mathcal{C} be a category. An object $\top \in \mathcal{C}$ is said to be a **terminal object** whenever for every object A there is a unique morphism $! : A \rightarrow \top$ with $\forall p \in \mathcal{C}(A, \top). p = !$. Similarly, terminal objects are unique up to isomorphism.

Definition A.8 (Cartesian category). A category \mathcal{C} is said to be **cartesian** if it has a terminal object \top and a product for any two objects $A, B \in \mathcal{C}$.