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

Andrea Laretto[1], Fabio Gadducci[2], Davide Trotta[2]

1: Tallinn University of Technology, 2: University of Pisa

*ICGT 2023, Leicester*

July 19th, 2023

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

1. Temporal logics and counterpart semantics

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

1. Temporal logics and counterpart semantics
2. Positive normal form

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

1. Temporal logics and counterpart semantics
2. Positive normal form
3. Categorical perspective

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

1. Temporal logics and counterpart semantics
2. Positive normal form
3. Categorical perspective
4. Agda formalization

# Contribution and Outline

*In this work we present the classical and categorical semantics of a counterpart-based temporal logic, and formalize it using the proof assistant Agda along with results on its positive normal form.*

1. Temporal logics and counterpart semantics
2. Positive normal form
3. Categorical perspective
4. Agda formalization
5. Conclusion and future work

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

# Temporal logics

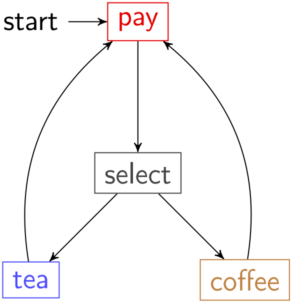
*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*

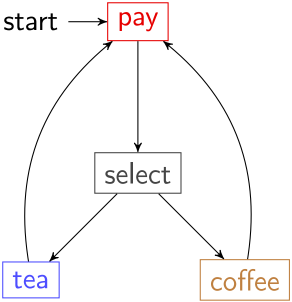



*Transition system for a simple vending machine*

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*



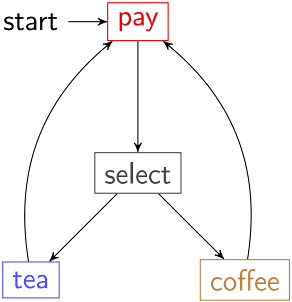

*Transition system for a simple vending machine*

2. Express desired properties as *formulas* in a **temporal logic**

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*




*Transition system for a simple vending machine*

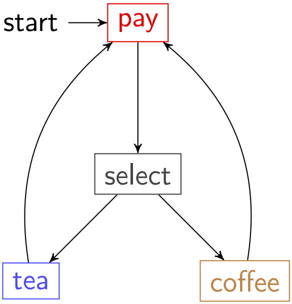
2. Express desired properties as *formulas* in a **temporal logic**

Always(Eventually(pay))

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*




*Transition system for a simple vending machine*

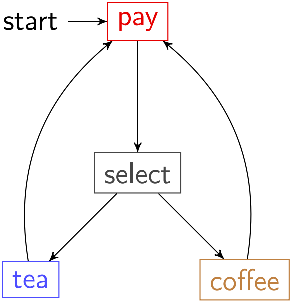
2. Express desired properties as *formulas* in a **temporal logic**

$$\text{Always}(\text{Eventually}(\text{pay})) \qquad \neg\,\text{Eventually}(\text{tea})$$

# Temporal logics

*Well-known formalism for specifying and verifying complex systems*

1. Represent the system as a **transition system**, called *model*




*Transition system for a simple vending machine*

2. Express desired properties as *formulas* in a **temporal logic**

$$\text{Always}(\text{Eventually}(\text{pay})) \qquad \neg\,\text{Eventually}(\text{tea})$$

3. Use a program to *check* that the *model* **satisfies** the formula

# Motivation: Multi-component models

- States are simply atomic points

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
    - Time evolution of *graph topologies*: merging nodes, deletion of edges

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
  - Time evolution of *graph topologies*: merging nodes, deletion of edges
  - Managing *processes* in memory: forking, allocation and deallocation

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
  - Time evolution of *graph topologies*: merging nodes, deletion of edges
  - Managing *processes* in memory: forking, allocation and deallocation
  - Dynamic behaviour of *election algorithms*: splitting and union of parties

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
  - Time evolution of *graph topologies*: merging nodes, deletion of edges
  - Managing *processes* in memory: forking, allocation and deallocation
  - Dynamic behaviour of *election algorithms*: splitting and union of parties
- Objectives:

*Can we enrich our models to express multi-component behaviour?*

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
  - Time evolution of *graph topologies*: merging nodes, deletion of edges
  - Managing *processes* in memory: forking, allocation and deallocation
  - Dynamic behaviour of *election algorithms*: splitting and union of parties
- Objectives:

*Can we enrich our models to express multi-component behaviour?*

*Can we define logics that can reason on the fate of individual elements?*

# Motivation: Multi-component models

- States are simply atomic points
- In practice, states often have structure that can change in time:
  - Time evolution of *graph topologies*: merging nodes, deletion of edges
  - Managing *processes* in memory: forking, allocation and deallocation
  - Dynamic behaviour of *election algorithms*: splitting and union of parties
- Objectives:

  *Can we enrich our models to express multi-component behaviour?*
  *Can we define logics that can reason on the fate of individual elements?*

- *Yes!* Using **counterpart models** and quantified temporal logics

# Counterpart paradigm

- Standard LTL traces: *sequences of states*

# Counterpart paradigm

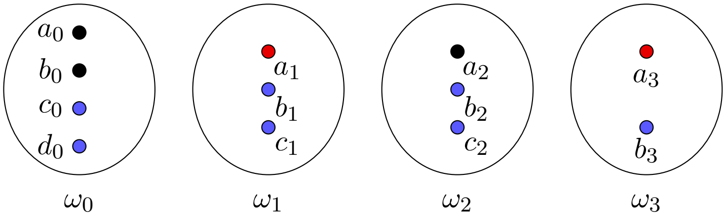
- Standard LTL traces: *sequences of states*


- Associate to each state a set of individuals, called **worlds**

# Counterpart paradigm

- Standard LTL traces: *sequences of states*

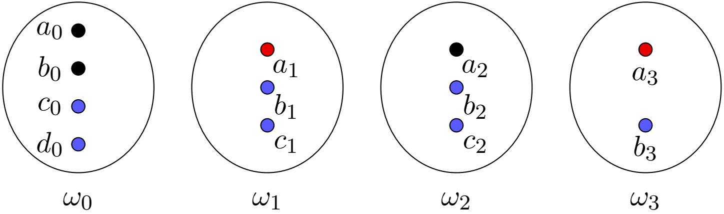

- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds*



# Counterpart paradigm

- Standard LTL traces: *sequences of states*


- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds*

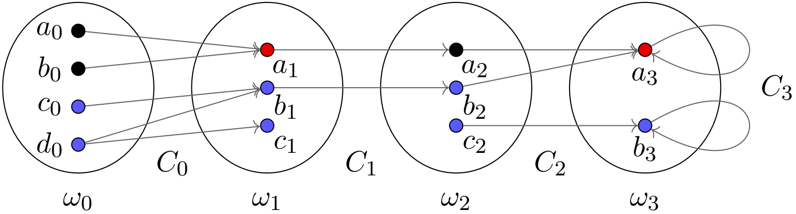



*How do we represent transitions?*

# Counterpart paradigm

- Standard LTL traces: *sequences of states*

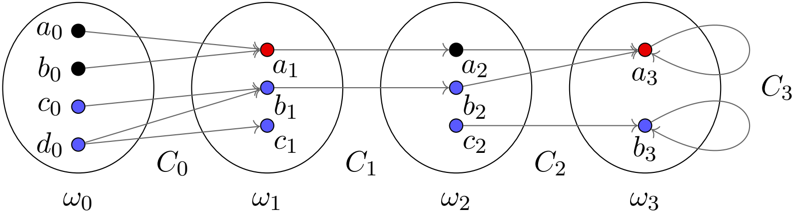

- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds, connected with **counterpart relations***



# Counterpart paradigm

- Standard LTL traces: *sequences of states*


- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds, connected with **counterpart relations***

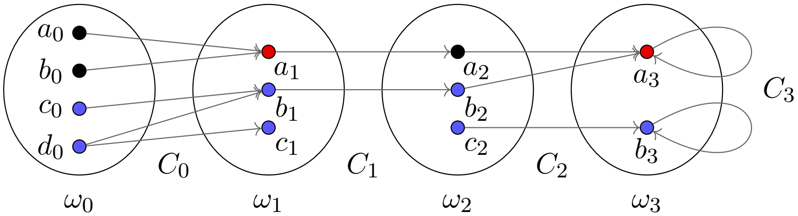



- Intuition: individuals connected by a relation *are the same after one step*

## Counterpart paradigm

- Standard LTL traces: *sequences of states*


- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds, connected with **counterpart relations***




- Intuition: individuals connected by a relation *are the same after one step*
- We call these sequences of worlds and relations **counterpart traces**

# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to$ **Set** and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$

# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to$ **Set** and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature* $\Sigma$

# Counterpart traces with algebras

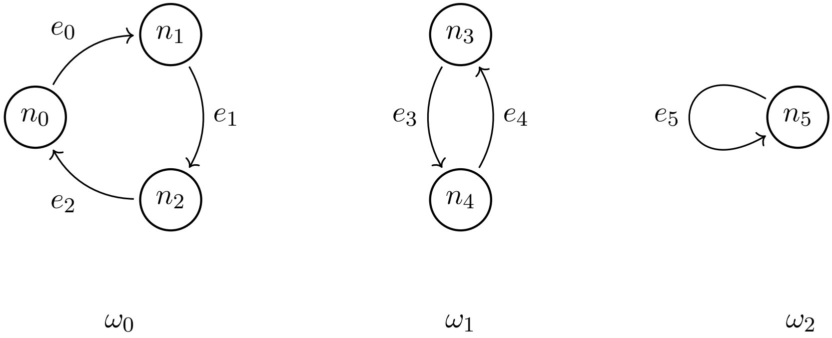
- Counterpart trace: function $\omega : \mathbb{N} \to$ **Set** and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature $\Sigma$*
- *Idea:* take $\Sigma$-algebras and structure-preserving relations between them

# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to \mathbf{Set}$ and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature* $\Sigma$
- *Idea:* take $\Sigma$-algebras and structure-preserving relations between them
- Examples: (multi)graphs, undirected graphs, trees, lists, etc.

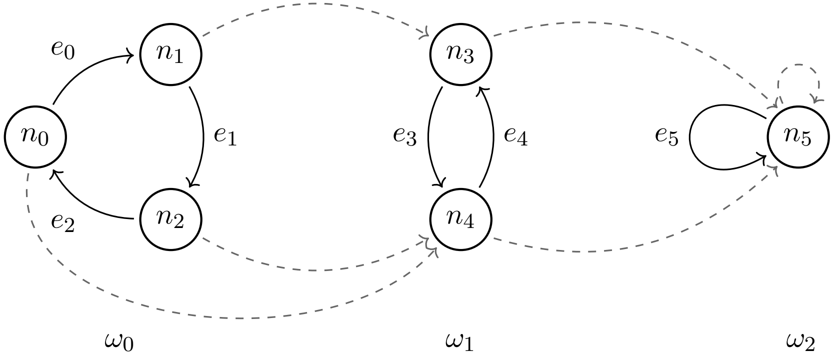
# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to \mathbf{Set}$ and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature $\Sigma$*
- *Idea:* take $\Sigma$-algebras and structure-preserving relations between them
- Examples: (multi)graphs, undirected graphs, trees, lists, etc.
- A counterpart trace on the signature of directed graphs:



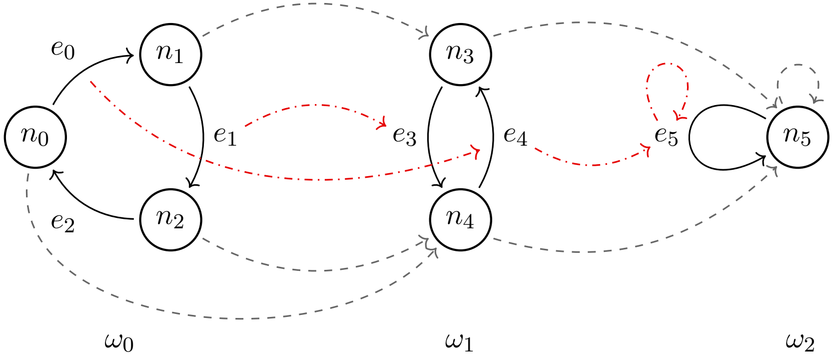
# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to$ **Set** and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature $\Sigma$*
- *Idea:* take $\Sigma$-algebras and structure-preserving relations between them
- Examples: (multi)graphs, undirected graphs, trees, lists, etc.
- A counterpart trace on the signature of directed graphs:



# Counterpart traces with algebras

- Counterpart trace: function $\omega : \mathbb{N} \to$ **Set** and $\{R_i \subseteq \omega(i) \times \omega(i+1)\}_{i \in \mathbb{N}}$
- *Worlds-as-algebras*: generalize sets to *algebras over a signature* $\Sigma$
- *Idea:* take $\Sigma$-algebras and structure-preserving relations between them
- Examples: (multi)graphs, undirected graphs, trees, lists, etc.
- A counterpart trace on the signature of directed graphs:



# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

*Counterpart model* $\approx$ *a category* $\mathcal{W}$

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

$$\textit{Counterpart model} \approx \textit{a category } \mathcal{W} + \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}}$$

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

$$\begin{aligned} \textit{Counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{a presheaf } D : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Relational presheaf}} \end{aligned}$$

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

$$\begin{aligned} \textit{Counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{a presheaf } D : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Relational presheaf}} \end{aligned}$$

- Objects of $\mathcal{W}$ are the states of the underlying *transition system*

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

$$\begin{aligned} \textit{Counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{a presheaf } D : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Relational presheaf}} \end{aligned}$$

- Objects of $\mathcal{W}$ are the states of the underlying *transition system*
- Morphisms of $\mathcal{W}$ represent *transitions* between states

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

$$\begin{aligned} \textit{Counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{a presheaf } D : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Relational presheaf}} \end{aligned}$$

- Objects of $\mathcal{W}$ are the states of the underlying *transition system*
- Morphisms of $\mathcal{W}$ represent *transitions* between states
- The *temporal structure* identifies the one-step transitions of the model

# Categorical semantics

- *Counterpart model*: a transition system enriched with worlds and counterpart relations between them
- Counterpart models can be understood within the unifying perspective of *category theory and categorical logic:*

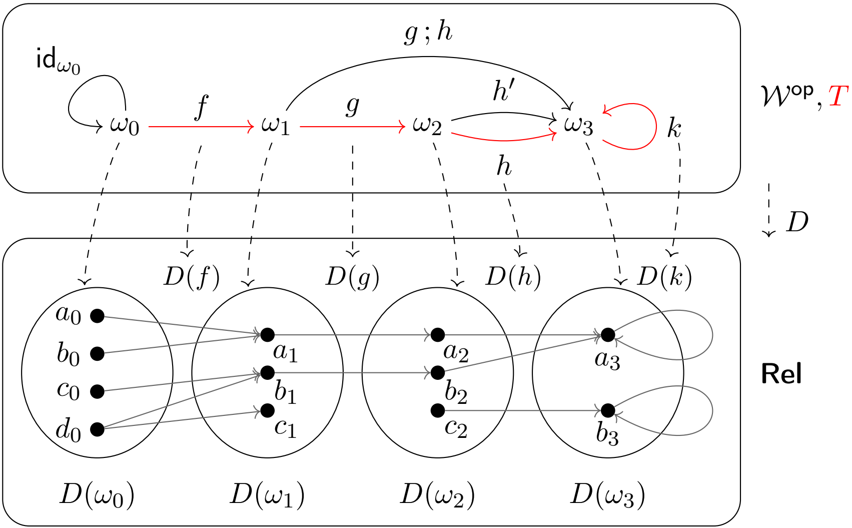
$$\begin{aligned} \textit{Counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{a presheaf } D : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Relational presheaf}} \end{aligned}$$

- Objects of $\mathcal{W}$ are the states of the underlying *transition system*
- Morphisms of $\mathcal{W}$ represent *transitions* between states
- The *temporal structure* identifies the one-step transitions of the model
- The *relational presheaf* assign worlds and counterpart relations to states

# Example – Counterpart model

$$
\begin{tikzcd}
\omega_0 \arrow[loop left, "\mathsf{id}_{\omega_0}"] \arrow[r, "f"] & \omega_1 \arrow[r, "g"] \arrow[rr, bend left=50, "g\,;h"] & \omega_2 \arrow[r, bend left, "h'"] \arrow[r, bend right, "h"'] & \omega_3 \arrow[loop right, "k"]
\end{tikzcd}
\qquad \mathcal{W}^{\mathsf{op}}, T
$$

# Example – Counterpart model



# Categorical semantics

- For the signature of directed graphs:

$$\begin{aligned} \textit{Graph counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \end{aligned}$$

# Categorical semantics

- For the signature of directed graphs:

$$\begin{aligned} \textit{Graph counterpart model} \approx\ & \textit{a category } \mathcal{W} \\ +\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\ +\ & \underbrace{\textit{relational presheaves } N, E : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Sorts of the signature}} \end{aligned}$$

# Categorical semantics

- For the signature of directed graphs:

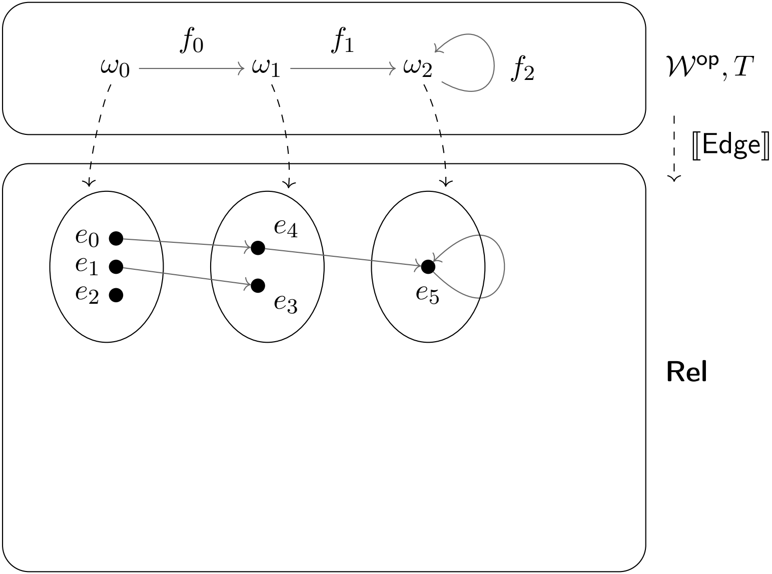
$$
\begin{aligned}
\textit{Graph counterpart model} \approx\ & \textit{a category } \mathcal{W} \\
+\ & \underbrace{\textit{a class } T \textit{ of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \\
+\ & \underbrace{\textit{relational presheaves } N, E : \mathcal{W}^{op} \to \textbf{\textit{Rel}}}_{\textit{Sorts of the signature}} \\
+\ & \underbrace{\textit{relational morphisms } s, t : E \Rightarrow N}_{\textit{Function symbols}}
\end{aligned}
$$

# Example – Graph counterpart model

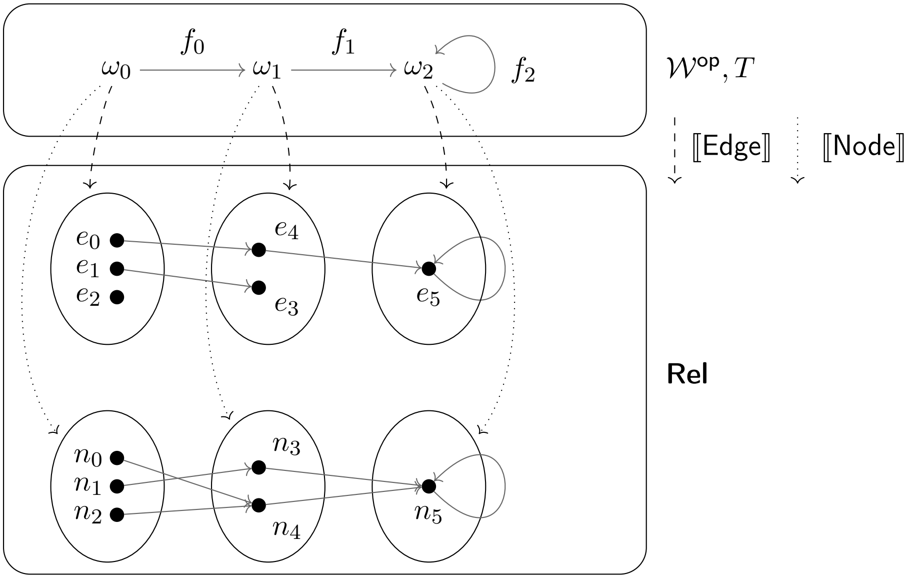
$$\begin{tikzcd} \omega_0 \arrow[r, "f_0"] & \omega_1 \arrow[r, "f_1"] & \omega_2 \arrow[loop right, "f_2"] \end{tikzcd}$$

$\mathcal{W}^{\mathsf{op}}, T$

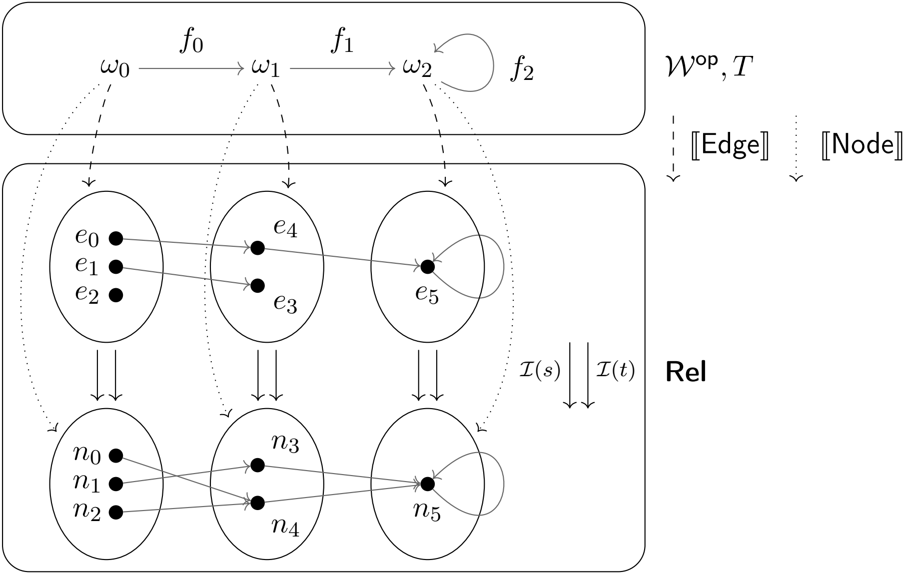
# Example – Graph counterpart model



# Example – Graph counterpart model



# Example – Graph counterpart model



# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi$$

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi \,\mathsf{Until}\, \phi$$

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x)$$

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi \,\mathsf{Until}\, \phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;
  - $\sigma, \mu \vDash \exists_{\mathsf{N}} x.\phi$ iff there is a node $n \in D(\omega_0)_N$ such that $\sigma, \mu[x \mapsto n] \vDash \phi$;

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi \,\mathsf{Until}\, \phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;
  - $\sigma, \mu \vDash \exists_{\mathsf{N}} x.\phi$ iff there is a node $n \in D(\omega_0)_N$ such that $\sigma, \mu[x \mapsto n] \vDash \phi$;
  - $\sigma, \mu \vDash \mathsf{O}\phi$ iff there is an assignment $\mu_1$ s.t. $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$;

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;
  - $\sigma, \mu \vDash \exists_{\mathsf{N}} x.\phi$ iff there is a node $n \in D(\omega_0)_N$ such that $\sigma, \mu[x \mapsto n] \vDash \phi$;
  - $\sigma, \mu \vDash \mathsf{O}\phi$ iff there is an assignment $\mu_1$ s.t. $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$;
  - $\sigma, \mu \vDash \phi_1 \mathsf{U} \phi_2$ iff there is an $\bar{n} \geq 0$ such that

# QLTL

- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;
  - $\sigma, \mu \vDash \exists_{\mathsf{N}} x.\phi$ iff there is a node $n \in D(\omega_0)_N$ such that $\sigma, \mu[x \mapsto n] \vDash \phi$;
  - $\sigma, \mu \vDash \mathsf{O}\phi$ iff there is an assignment $\mu_1$ s.t. $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$;
  - $\sigma, \mu \vDash \phi_1 \mathsf{U} \phi_2$ iff there is an $\bar{n} \geq 0$ such that
    1. for any $i < \bar{n}$, there is a $\mu_i$ such that $\langle \mu, \mu_i \rangle \in \sigma_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi_1$;

# QLTL

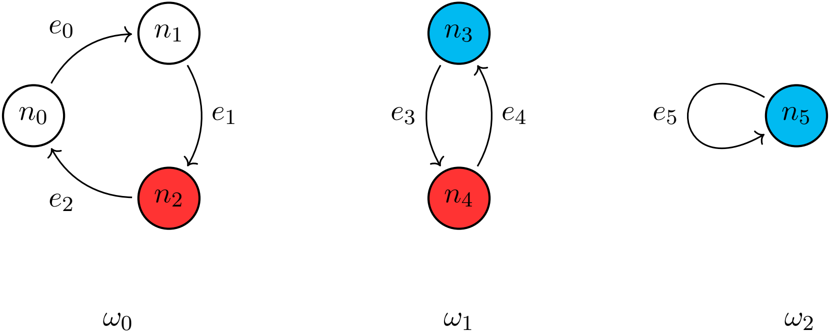
- QLTL: *(first-order) quantified linear temporal logic* using traces
- Syntax of QLTL formulas:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{Next}(\phi) \mid \phi\,\mathsf{Until}\,\phi \mid \exists_{\mathsf{N}} x.\phi \mid \exists_{\mathsf{E}} x.\phi \mid P(x) \mid \psi$$

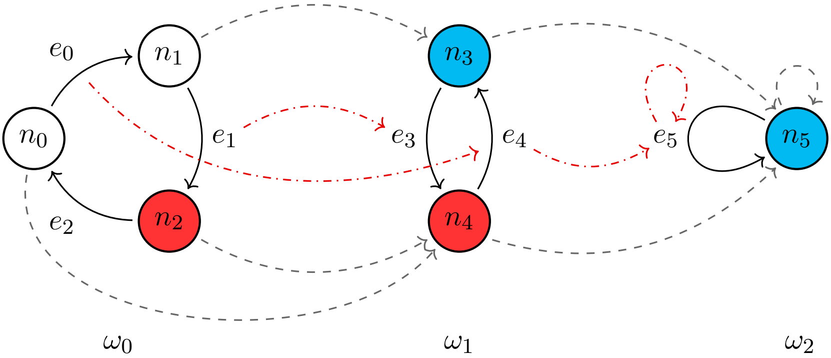
$$\psi := n =_{\mathsf{N}} n \mid e =_{\mathsf{E}} e, \quad \text{with} \quad n := x \mid s(e) \mid t(e), \quad \text{and} \quad e := x.$$

- Semantics: *given a trace* $\sigma$, define a satisfiability relation on (tuples of) nodes and edges satisfying $\phi$, i.e., assignments $\mu$ for the $\mathrm{fv}(\phi)$.
  - $\sigma, \mu \vDash \mathsf{true}$;
  - $\sigma, \mu \vDash \phi_1 \wedge \phi_2$ iff $\sigma, \mu \vDash \phi_1$ and $\sigma, \mu \vDash \phi_2$;
  - $\sigma, \mu \vDash e_1 =_{\mathsf{E}} e_2$ iff $\mu^*_{\mathsf{E}}(e_1) = \mu^*_{\mathsf{E}}(e_2)$;
  - $\sigma, \mu \vDash \exists_{\mathsf{N}} x.\phi$ iff there is a node $n \in D(\omega_0)_N$ such that $\sigma, \mu[x \mapsto n] \vDash \phi$;
  - $\sigma, \mu \vDash \mathsf{O}\phi$ iff there is an assignment $\mu_1$ s.t. $\langle \mu, \mu_1 \rangle \in C_0$ and $\sigma_1, \mu_1 \vDash \phi$;
  - $\sigma, \mu \vDash \phi_1 \mathsf{U} \phi_2$ iff there is an $\bar{n} \geq 0$ such that
    1. for any $i < \bar{n}$, there is a $\mu_i$ such that $\langle \mu, \mu_i \rangle \in \sigma_{\leq i}$ and $\sigma_i, \mu_i \vDash \phi_1$;
    2. there is a $\mu_{\bar{n}}$ such that $\langle \mu, \mu_{\bar{n}} \rangle \in \sigma_{\leq \bar{n}}$ and $\sigma_{\bar{n}}, \mu_{\bar{n}} \vDash \phi_2$;

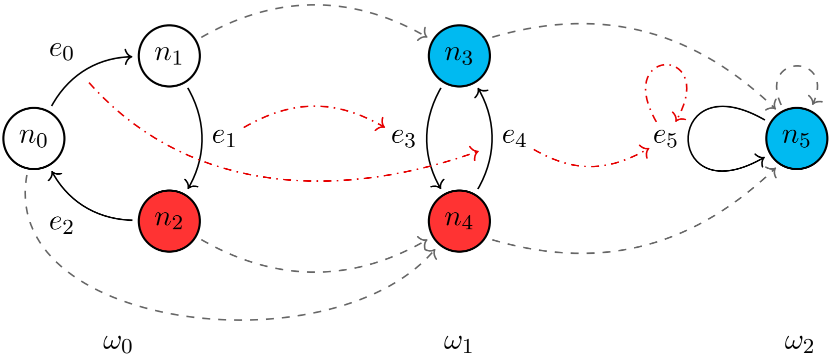
# Example – QLTL



# Example – QLTL

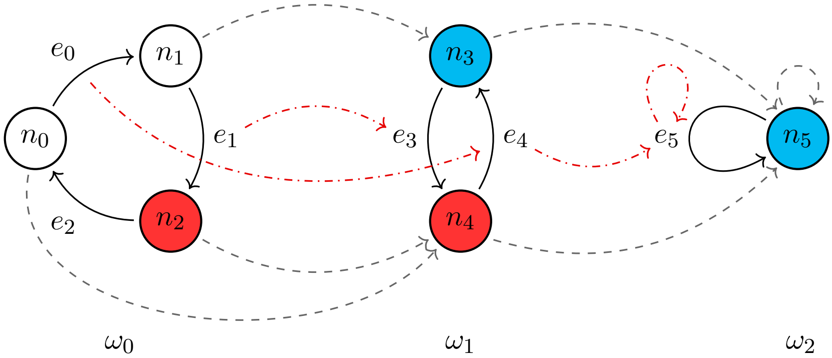


# Example – QLTL



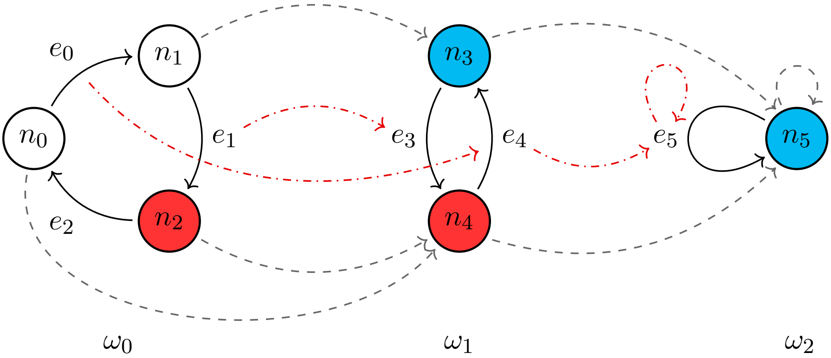

- $n_1 \vDash_{\omega_0} \mathsf{Next}(\mathsf{Blue}(x))$

# Example – QLTL



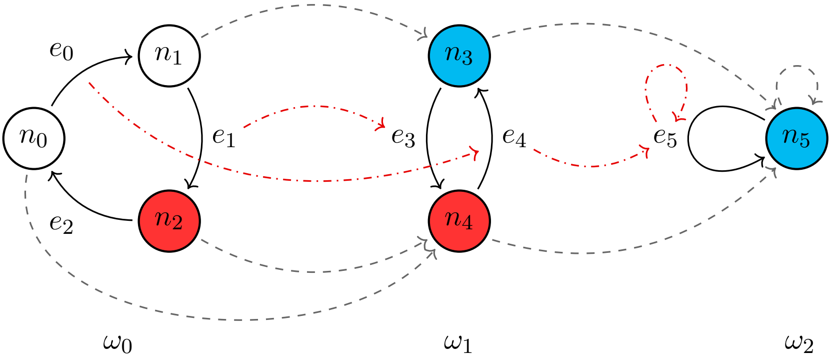

- $n_1 \vDash_{\omega_0}$ Next(Blue($x$))
- $n_0 \vDash_{\omega_0}$ ¬Next(Red($x$))

# Example – QLTL



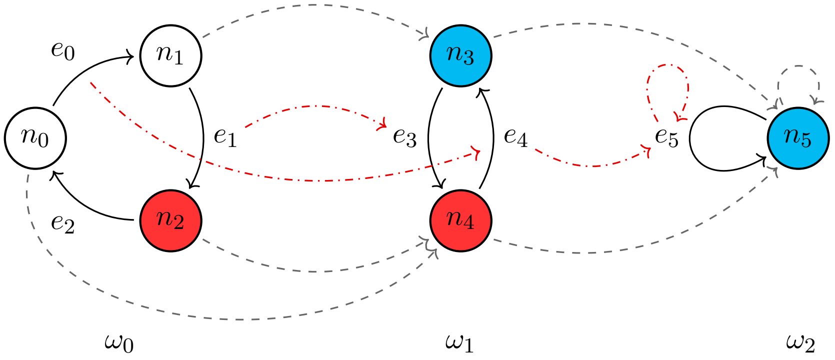

- $n_1 \vDash_{\omega_0}$ Next(Blue($x$))
- $n_0 \vDash_{\omega_0}$ ¬Next(Red($x$))
- $n_2 \vDash_{\omega_0}$ Red($x$) Until Blue($x$)

# Example – QLTL



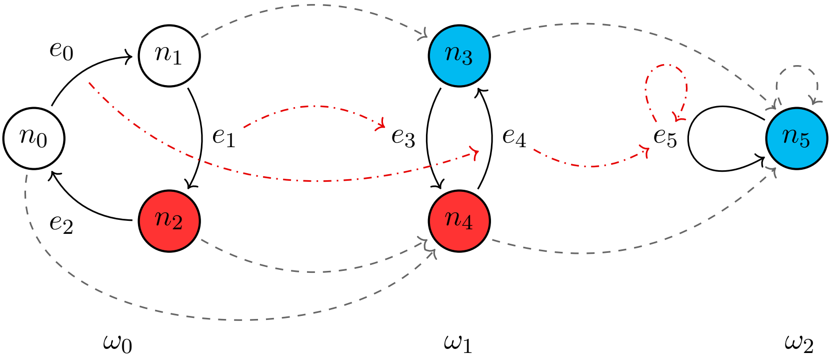

- $n_1 \vDash_{\omega_0}$ Next(Blue$(x)$)
- $n_0 \vDash_{\omega_0}$ $\neg$Next(Red$(x)$)
- $n_2 \vDash_{\omega_0}$ Red$(x)$ Until Blue$(x)$
- $(n_3, n_4) \vDash_{\omega_1}$ Next$(x = y)$

# Example – QLTL




- $n_1 \vDash_{\omega_0}$ Next(Blue($x$))
- $n_0 \vDash_{\omega_0}$ ¬Next(Red($x$))
- $n_2 \vDash_{\omega_0}$ Red($x$) Until Blue($x$)
- $(n_3, n_4) \vDash_{\omega_1}$ Next($x = y$)
- $() \vDash_{w_0}$ $\exists x$.Next(Blue($x$))

# Example – QLTL




- $n_1 \vDash_{\omega_0}$ Next(Blue($x$))
- $n_0 \vDash_{\omega_0}$ ¬Next(Red($x$))
- $n_2 \vDash_{\omega_0}$ Red($x$) Until Blue($x$)
- $(n_3, n_4) \vDash_{\omega_1}$ Next($x = y$)
- $() \vDash_{w_0}$ $\exists x$.Next(Blue($x$))
- $(n_1, n_2) \vDash_{\omega_0}$ $(\neg(x = y))$ Until $(x = y)$

# Graph formulae

- We can define formulae that capture structural aspects of the graph:

$$\textsf{\textbf{loop}}(e) \quad := \quad s(e) =_{\mathsf{N}} t(e),$$

# Graph formulae

- We can define formulae that capture structural aspects of the graph:

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

# Graph formulae

- We can define formulae that capture structural aspects of the graph:

$$
\begin{aligned}
\mathbf{loop}(e) &:= s(e) =_{\mathsf{N}} t(e), \\
\mathbf{hasLoop}(n) &:= \exists_{\mathsf{E}} e.s(e) =_{\mathsf{N}} n \wedge \mathbf{loop}(e), \\
\mathbf{composable}(x, y) &:= t(x) =_{\mathsf{N}} s(y)
\end{aligned}
$$

# Graph formulae

- We can define formulae that capture structural aspects of the graph:

$$
\begin{aligned}
\textbf{loop}(e) &:= s(e) =_{\mathsf{N}} t(e), \\
\textbf{hasLoop}(n) &:= \exists_{\mathsf{E}} e . s(e) =_{\mathsf{N}} n \wedge \textbf{loop}(e), \\
\textbf{composable}(x, y) &:= t(x) =_{\mathsf{N}} s(y) \\
\textbf{haveComposition}(x, y) &:= \textbf{composable}(x, y)
\end{aligned}
$$

# Graph formulae

- We can define formulae that capture structural aspects of the graph:

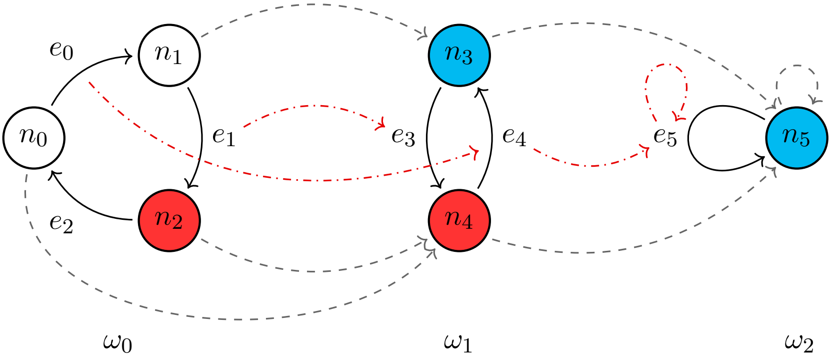
$$
\begin{aligned}
\textbf{loop}(e) \quad &:= \quad s(e) =_{\mathsf{N}} t(e), \\
\textbf{hasLoop}(n) \quad &:= \quad \exists_{\mathsf{E}} e.s(e) =_{\mathsf{N}} n \land \textbf{loop}(e), \\
\textbf{composable}(x, y) \quad &:= \quad t(x) =_{\mathsf{N}} s(y) \\
\textbf{haveComposition}(x, y) \quad &:= \quad \textbf{composable}(x, y) \\
&\;\land \quad \exists_{\mathsf{E}} e.(s(x) =_{\mathsf{N}} s(e) \land t(e) =_{\mathsf{N}} t(y))
\end{aligned}
$$

# Graph formulae

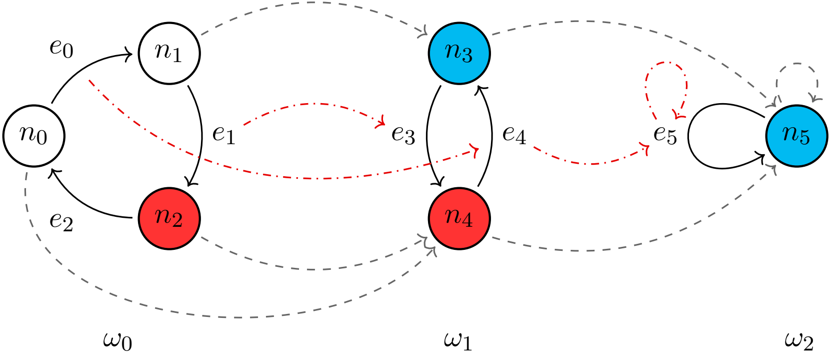
- We can define formulae that capture structural aspects of the graph:

$$
\begin{aligned}
\textbf{loop}(e) &:= s(e) =_{\mathsf{N}} t(e), \\
\textbf{hasLoop}(n) &:= \exists_{\mathsf{E}} e. s(e) =_{\mathsf{N}} n \wedge \textbf{loop}(e), \\
\textbf{composable}(x, y) &:= t(x) =_{\mathsf{N}} s(y) \\
\textbf{haveComposition}(x, y) &:= \textbf{composable}(x, y) \\
&\wedge \quad \exists_{\mathsf{E}} e.(s(x) =_{\mathsf{N}} s(e) \wedge t(e) =_{\mathsf{N}} t(y)) \\
\textbf{adjacent}(x, y) &:= \exists_{\mathsf{E}} e.((s(e) =_{\mathsf{N}} x \wedge t(e) =_{\mathsf{N}} y) \\
&\qquad \vee (t(e) =_{\mathsf{N}} x \wedge s(e) =_{\mathsf{N}} y))
\end{aligned}
$$

# Example – QLTL on graphs

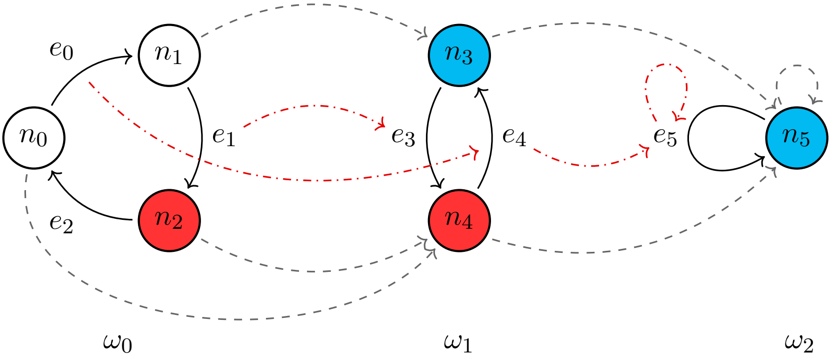


# Example – QLTL on graphs



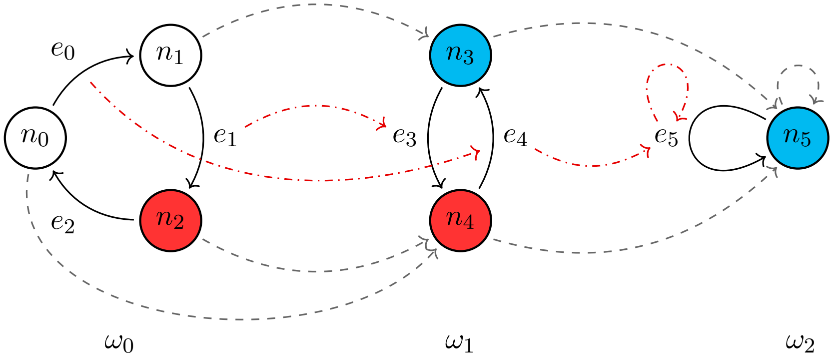

- $e_4 \vDash_{\omega_1}$ Next(**loop**$(x)$)

# Example – QLTL on graphs



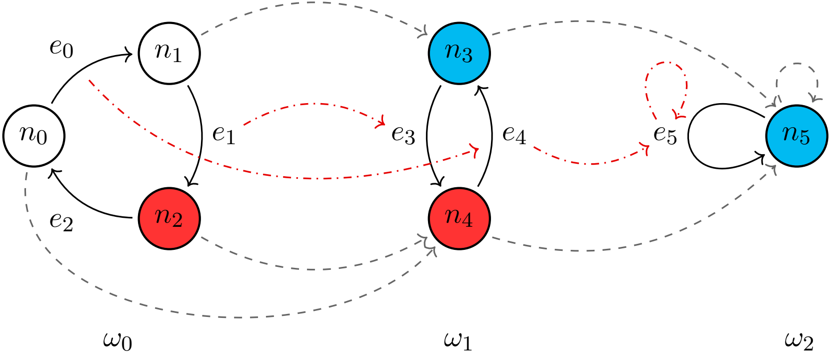

- $e_4 \vDash_{\omega_1}$ Next(**loop**$(x)$)
- $e_3 \vDash_{\omega_1}$ $\neg$Next(**loop**$(x)$)

# Example – QLTL on graphs

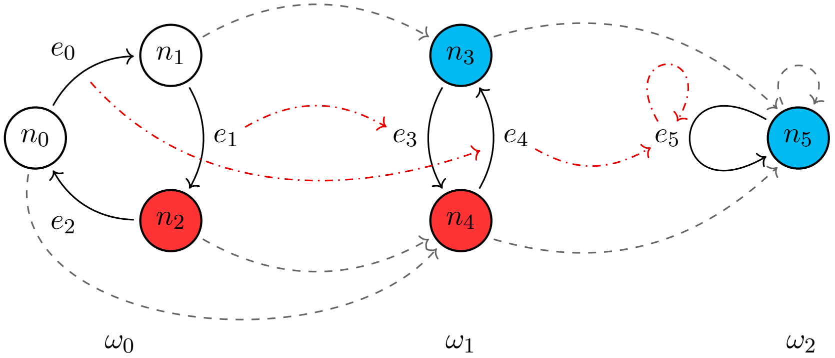



- $e_4 \vDash_{\omega_1}$ $\mathsf{Next}(\textbf{loop}(x))$
- $e_3 \vDash_{\omega_1}$ $\neg\mathsf{Next}(\textbf{loop}(x))$
- $(e_3, e_4) \vDash_{\omega_0}$ $\textbf{composable}(x, y)$

# Example – QLTL on graphs

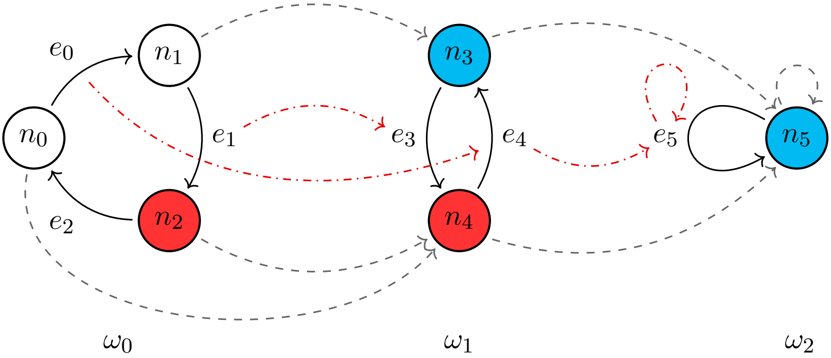



- $e_4 \vDash_{\omega_1}$ Next(**loop**$(x)$)
- $e_3 \vDash_{\omega_1}$ $\neg$Next(**loop**$(x)$)
- $(e_3, e_4) \vDash_{\omega_0}$ **composable**$(x, y)$
- $(n_0, n_2) \vDash_{\omega_0}$ **adjacent**$(x, y)$




- $e_4 \vDash_{\omega_1}$ Next(**loop**$(x)$)
- $e_3 \vDash_{\omega_1} \neg$Next(**loop**$(x)$)
- $(e_3, e_4) \vDash_{\omega_0}$ **composable**$(x, y)$
- $(n_0, n_2) \vDash_{\omega_0}$ **adjacent**$(x, y)$
- $(n_0, n_2) \nvDash_{\omega_0}$ O**adjacent**$(x, y)$

# Example – QLTL on graphs




- $e_4 \vDash_{\omega_1}$ Next(**loop**$(x)$)
- $e_3 \vDash_{\omega_1} \neg$Next(**loop**$(x)$)
- $(e_3, e_4) \vDash_{\omega_0}$ **composable**$(x, y)$
- $(n_0, n_2) \vDash_{\omega_0}$ **adjacent**$(x, y)$
- $(n_0, n_2) \nvDash_{\omega_0}$ O**adjacent**$(x, y)$
- $e_0 \vDash_{\omega_0} \Diamond$**loop**$(x)$

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant
- PNF for QLTL:

$$\phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists_{\mathsf{E}} x.\phi \mid \exists_{\mathsf{N}} x.\phi \mid \forall_{\mathsf{E}} x.\phi \mid \forall_{\mathsf{N}} x.\phi$$

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant
- PNF for QLTL:

$$\phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists_{\mathsf{E}} x.\phi \mid \exists_{\mathsf{N}} x.\phi \mid \forall_{\mathsf{E}} x.\phi \mid \forall_{\mathsf{N}} x.\phi$$

$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2$$

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant
- PNF for QLTL:

$$\phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists_{\mathsf{E}} x.\phi \mid \exists_{\mathsf{N}} x.\phi \mid \forall_{\mathsf{E}} x.\phi \mid \forall_{\mathsf{N}} x.\phi$$

$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2 \mid \underline{\mathsf{NextF}}(\phi) \mid \phi_1 \underline{\mathsf{UntilF}} \phi_2 \mid \phi_1 \underline{\mathsf{WUntilF}} \phi_2$$

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant
- PNF for QLTL:

$$\phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists_{\mathsf{E}} x.\phi \mid \exists_{\mathsf{N}} x.\phi \mid \forall_{\mathsf{E}} x.\phi \mid \forall_{\mathsf{N}} x.\phi$$

$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2 \mid \underline{\mathsf{NextF}}(\phi) \mid \phi_1 \underline{\mathsf{UntilF}} \phi_2 \mid \phi_1 \underline{\mathsf{WUntilF}} \phi_2$$

- Intuition: *universal counterparts* to the previous operators

$$\begin{aligned} \neg\mathsf{Next}(\phi) &\equiv \mathsf{NextF}(\neg\phi) \\ \neg(\phi_1 \mathsf{Until} \phi_2) &\equiv (\neg\phi_2)\mathsf{WUntilF}(\neg\phi_1 \wedge \neg\phi_2) \\ \neg(\phi_1 \mathsf{WUntil} \phi_2) &\equiv (\neg\phi_2)\mathsf{UntilF}(\neg\phi_1 \wedge \neg\phi_2) \end{aligned}$$

# Positive normal forms for QLTL

- PNF: a standard presentation for temporal logics
- Usually given to simplify model checking and for fixpoint semantics
- PNFs are essential to work in a constructive proof assistant
- PNF for QLTL:

$$\phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists_{\mathsf{E}} x.\phi \mid \exists_{\mathsf{N}} x.\phi \mid \forall_{\mathsf{E}} x.\phi \mid \forall_{\mathsf{N}} x.\phi$$

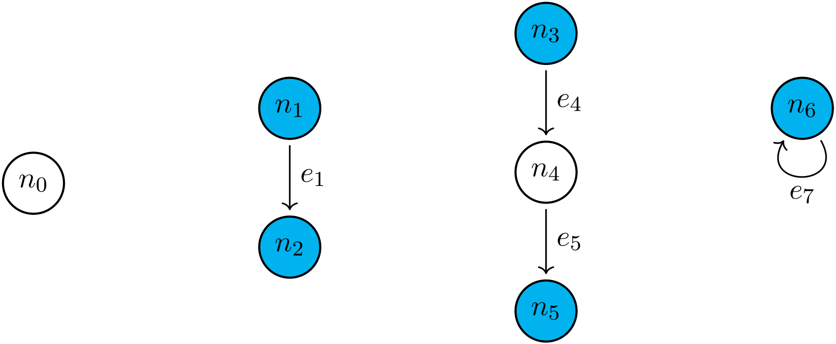
$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2 \mid \underline{\mathsf{NextF}}(\phi) \mid \phi_1 \underline{\mathsf{UntilF}} \phi_2 \mid \phi_1 \underline{\mathsf{WUntilF}} \phi_2$$

- Intuition: *universal counterparts* to the previous operators

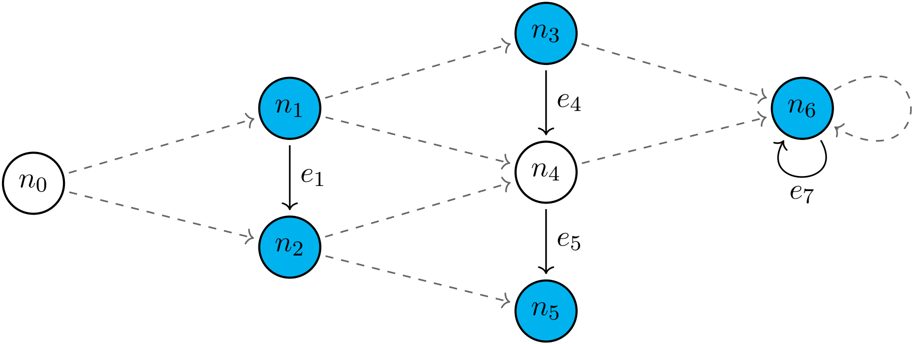
$$\begin{aligned} \neg\mathsf{Next}(\phi) &\equiv \mathsf{NextF}(\neg\phi) \\ \neg(\phi_1 \mathsf{Until} \phi_2) &\equiv (\neg\phi_2)\mathsf{WUntilF}(\neg\phi_1 \wedge \neg\phi_2) \\ \neg(\phi_1 \mathsf{WUntil} \phi_2) &\equiv (\neg\phi_2)\mathsf{UntilF}(\neg\phi_1 \wedge \neg\phi_2) \end{aligned}$$

- Become particularly useful to treat duplicating relations

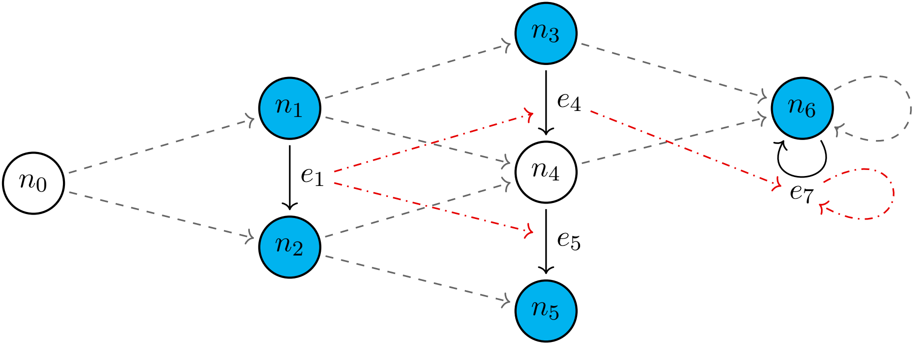
# Example – Duplicating relations



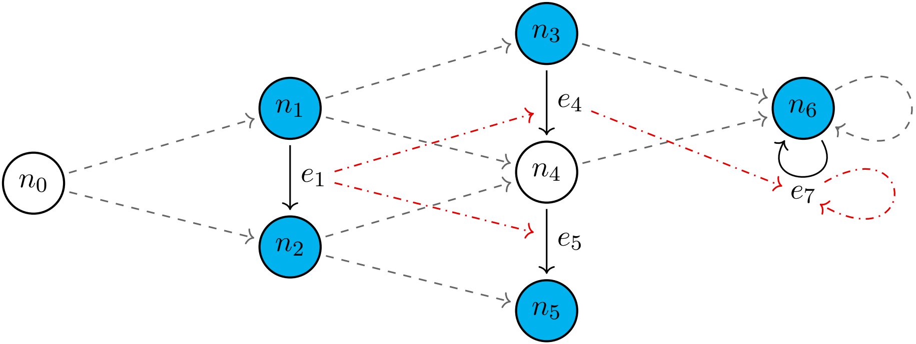
# Example – Duplicating relations



# Example – Duplicating relations

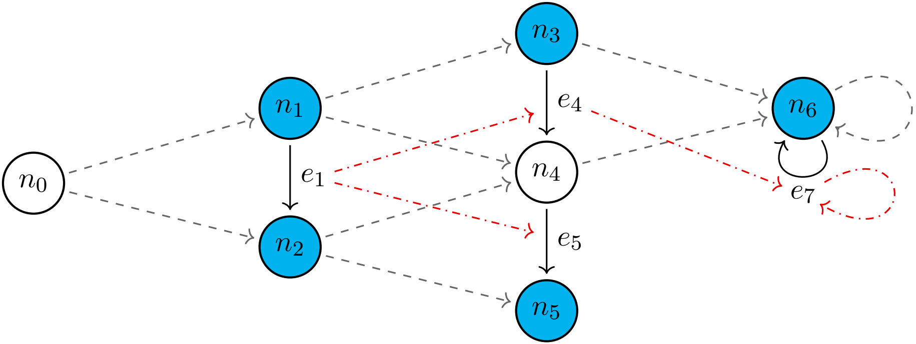


# Example – Duplicating relations



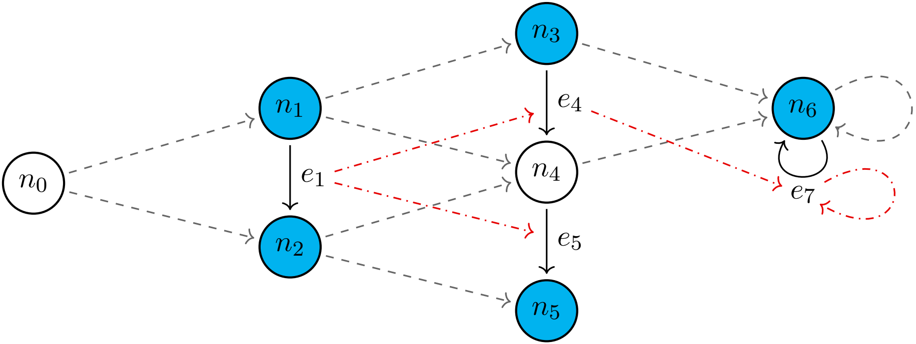

- $n_0 \vDash_{\omega_0} \mathsf{NextF}(\mathsf{Blue}(x))$

# Example – Duplicating relations



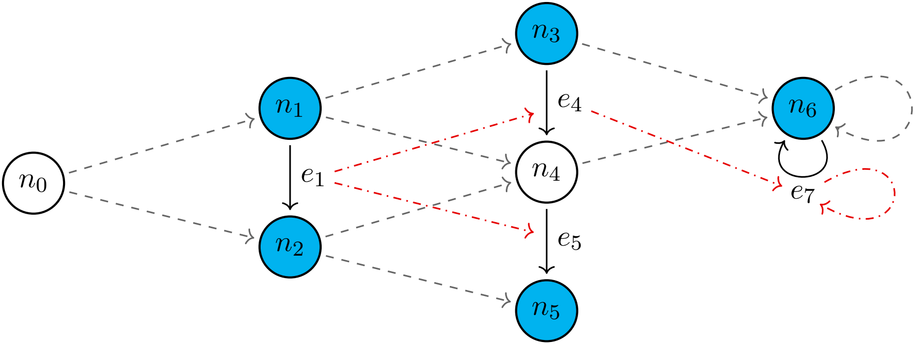

- $n_0 \vDash_{\omega_0}$ NextF(Blue($x$))
- $n_1 \nvDash_{\omega_1}$ NextF(Blue($x$))

# Example – Duplicating relations



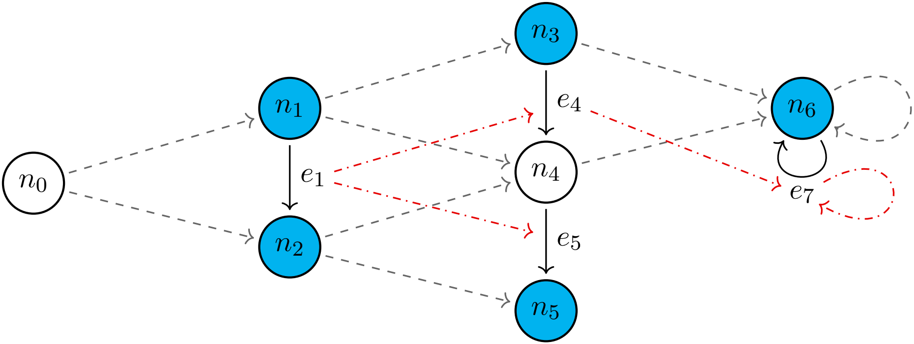

- $n_0 \vDash_{\omega_0}$ NextF(Blue($x$))
- $n_1 \nvDash_{\omega_1}$ NextF(Blue($x$))
- $e_5 \vDash_{\omega_1}$ NextF(**loop**($x$))

# Example – Duplicating relations



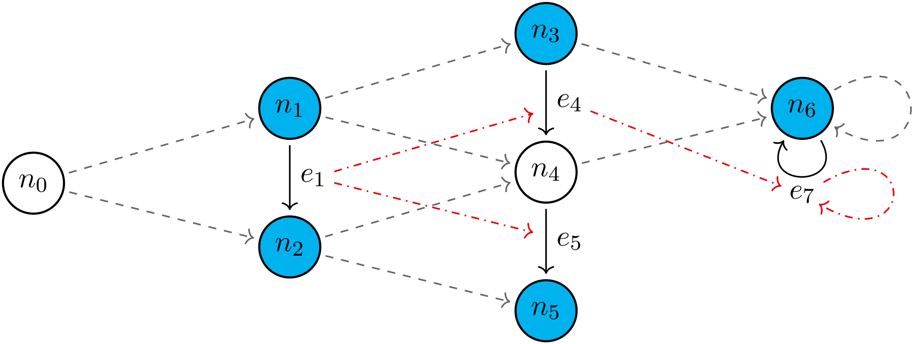

- $n_0 \vDash_{\omega_0}$ NextF(Blue($x$))
- $n_1 \nvDash_{\omega_1}$ NextF(Blue($x$))
- $e_5 \vDash_{\omega_1}$ NextF(**loop**($x$))
- $e_1 \vDash_{\omega_1}$ Blue($s(x)$) Until (**loop**($x$))

# Example – Duplicating relations




- $n_0 \vDash_{\omega_0}$ NextF(Blue($x$))
- $n_1 \nvDash_{\omega_1}$ NextF(Blue($x$))
- $e_5 \vDash_{\omega_1}$ NextF(**loop**($x$))
- $e_1 \vDash_{\omega_1}$ Blue($s(x)$) Until (**loop**($x$))
- $e_1 \nvDash_{\omega_1}$ Blue($s(x)$) UntilF (**loop**($x$))

# Example – Duplicating relations




- $n_0 \vDash_{\omega_0}$ NextF(Blue($x$))
- $n_1 \nvDash_{\omega_1}$ NextF(Blue($x$))
- $e_5 \vDash_{\omega_1}$ NextF(**loop**($x$))
- $e_1 \vDash_{\omega_1}$ Blue($s(x)$) Until (**loop**($x$))
- $e_1 \nvDash_{\omega_1}$ Blue($s(x)$) UntilF (**loop**($x$))
- $e_4 \vDash_{\omega_1}$ Blue($s(x)$) WUntilF (**false**)

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

1. A formalization of categorical QLTL and its models in Agda

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

1. A formalization of categorical QLTL and its models in Agda
2. Categorical semantics formalized using the agda-categories library

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

1. A formalization of categorical QLTL and its models in Agda
2. Categorical semantics formalized using the agda-categories library
3. Algebraic QLTL: worlds-as-algebras over any multi-sorted signature

# Agda formalization

Agda

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

1. A formalization of categorical QLTL and its models in Agda
2. Categorical semantics formalized using the agda-categories library
3. Algebraic QLTL: worlds-as-algebras over any multi-sorted signature
4. A classical set-based semantics without the use of categorical logic

# Agda formalization

- Agda: *dependently typed programming language and proof assistant*
- Can be used in practice to formalize mathematical constructions
- Mechanization work:

1. A formalization of categorical QLTL and its models in Agda
2. Categorical semantics formalized using the agda-categories library
3. Algebraic QLTL: worlds-as-algebras over any multi-sorted signature
4. A classical set-based semantics without the use of categorical logic
5. Presentation of the *positive normal forms* of QLTL, also in Agda

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*
- Formalizing constructions and semantics establishes their *correctness*

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*
- ☺ Formalizing constructions and semantics establishes their *correctness*
- ☺ Provides a formal setting to *test and experiment with* temporal logics

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*
- ☺ Formalizing constructions and semantics establishes their *correctness*
- ☺ Provides a formal setting to *test and experiment with* temporal logics
- ☺ Establishes a foundation to build *verified model checkers* for QTL

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*
- Formalizing constructions and semantics establishes their *correctness*
- Provides a formal setting to *test and experiment with* temporal logics
- Establishes a foundation to build *verified model checkers* for QTL
- Gives a program to *convert* standard models into categorical ones:

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*

☺ Formalizing constructions and semantics establishes their *correctness*

☺ Provides a formal setting to *test and experiment with* temporal logics

☺ Establishes a foundation to build *verified model checkers* for QTL

☺ Gives a program to *convert* standard models into categorical ones:

1. Define the semantics of the logic with *categorical* notions and models

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*

☺ Formalizing constructions and semantics establishes their *correctness*

☺ Provides a formal setting to *test and experiment with* temporal logics

☺ Establishes a foundation to build *verified model checkers* for QTL

☺ Gives a program to *convert* standard models into categorical ones:

1. Define the semantics of the logic with *categorical* notions and models
2. Provide a standard *non-categorical* transition system as model

# Agda formalization

- Categorical semantics: **1388 lines** of Agda code
- Positive normal form: **1740 lines** of Agda code
- *Why is a formal presentation of our logic useful?*
- Formalizing constructions and semantics establishes their *correctness*
- Provides a formal setting to *test and experiment with* temporal logics
- Establishes a foundation to build *verified model checkers* for QTL
- Gives a program to *convert* standard models into categorical ones:
  1. Define the semantics of the logic with *categorical* notions and models
  2. Provide a standard *non-categorical* transition system as model
  3. Use the procedure `ClassicalToCategorical` to construct the categorical model so that the logic can be applied

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices do not necessarily get in the way of practical applications

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices do not necessarily get in the way of practical applications
- Main definitions used:
  - Categories, functors, natural transformations
  - **Rel**: category of sets and relations
  - Free categories generated from a quiver (`PathCategory`)
  - Presheaves, the category of (relational) presheaves is complete
  - Relational presheaves and morphisms between them

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- 🙂 Extremely practical and flexible, no magic involved
- 🙂 Design choices do not necessarily get in the way of practical applications
- Main definitions used:
  - Categories, functors, natural transformations
  - **Rel**: category of sets and relations
  - Free categories generated from a quiver (`PathCategory`)
  - Presheaves, the category of (relational) presheaves is complete
  - Relational presheaves and morphisms between them
- 😐 Functoriality and setoid-equality preservation can be annoying to prove

# Experience with agda-categories

https://github.com/agda/agda-categories

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- Extremely practical and flexible, no magic involved
- Design choices do not necessarily get in the way of practical applications
- Main definitions used:
  - Categories, functors, natural transformations
  - **Rel**: category of sets and relations
  - Free categories generated from a quiver (`PathCategory`)
  - Presheaves, the category of (relational) presheaves is complete
  - Relational presheaves and morphisms between them
- Functoriality and setoid-equality preservation can be annoying to prove
- Relatively limited use of the theorems/properties given by the library

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:
    - PNF with *partial functions* as counterpart relations

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:
  - PNF with *partial functions* as counterpart relations
  - PNF with general relations (i.e. allow duplication of entities)

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:
  - PNF with *partial functions* as counterpart relations
  - PNF with general relations (i.e. allow duplication of entities)
- Expansion laws and equivalences in QLTL in both settings

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:
  - PNF with *partial functions* as counterpart relations
  - PNF with general relations (i.e. allow duplication of entities)
- Expansion laws and equivalences in QLTL in both settings

$\Rightarrow$ LTL-like expansion laws break down in the case of relations!

# PNF in Agda

- Formalized in Agda: PNF equivalence (using classical reasoning)
- Two cases, using non-categorical semantics:
  - PNF with *partial functions* as counterpart relations
  - PNF with general relations (i.e. allow duplication of entities)
- Expansion laws and equivalences in QLTL in both settings

$\Rightarrow$ LTL-like expansion laws break down in the case of relations!

$\Rightarrow$ (But they can be mostly recovered in the case of partial functions.)

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models
  - interfacing Agda with SMT solvers and model checkers for QLTL

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models
  - interfacing Agda with SMT solvers and model checkers for QLTL
  - formalize syntax and models of the logic with indexed categories and morphisms between them, as in categorical logic [Jacobs, 2001]

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models
  - interfacing Agda with SMT solvers and model checkers for QLTL
  - formalize syntax and models of the logic with indexed categories and morphisms between them, as in categorical logic [Jacobs, 2001]
- A study of formally-presented temporal logics is absent in the literature

# Conclusion and Future Work

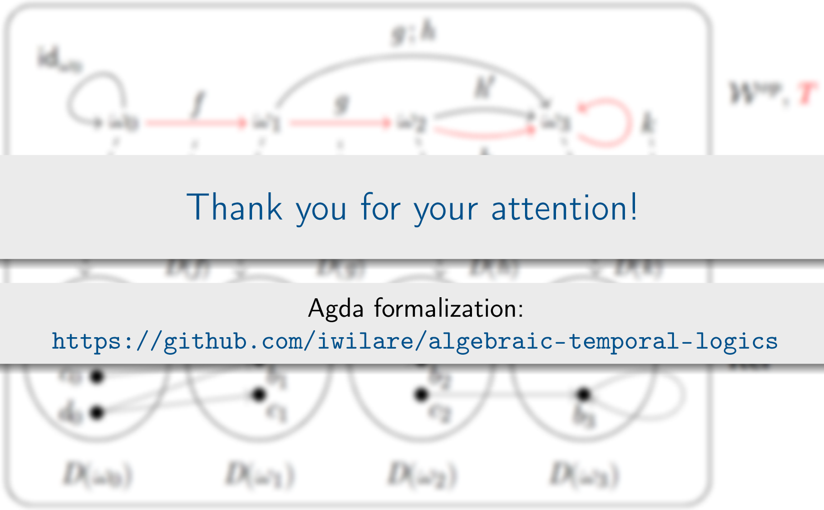
*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models
  - interfacing Agda with SMT solvers and model checkers for QLTL
  - formalize syntax and models of the logic with indexed categories and morphisms between them, as in categorical logic [Jacobs, 2001]
- A study of formally-presented temporal logics is absent in the literature
- Other verified model checkers: LTL in Isabelle [Nipkow, 2013]

# Conclusion and Future Work

*In this work we present a counterpart-based temporal logic that can reason on the temporal evolution of algebraic structures and formalize its semantics in Agda along with results on its PNF.*

- Many possible extensions of this work:
  - formalization of second-order QLTL to express set quantification
  - extending counterpart semantics to CTL, CTL$^*$ and their models
  - interfacing Agda with SMT solvers and model checkers for QLTL
  - formalize syntax and models of the logic with indexed categories and morphisms between them, as in categorical logic [Jacobs, 2001]
- A study of formally-presented temporal logics is absent in the literature
- Other verified model checkers: LTL in Isabelle [Nipkow, 2013]
- Proof searching using reflection in Agda for CTL [O'Connor, 2016]



# Thank you for your attention!

Agda formalization:
https://github.com/iwilare/algebraic-temporal-logics