Categorical Semantics for Counterpart-based Temporal Logics in Agda

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

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

33rd Nordic Workshop on Programming Theory, Bergen

November 2nd, 2022

Contribution and Outline

In this work we present the 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 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 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. Categorical perspective

Contribution and Outline

In this work we present the 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. Categorical perspective
3. Agda formalization

Contribution and Outline

In this work we present the 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. Categorical perspective
3. Agda formalization
4. Positive normal form

Contribution and Outline

In this work we present the 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. Categorical perspective
3. Agda formalization
4. Positive normal form
5. Conclusion and future work

Temporal logics

Well-known formalism for specifying and verifying complex systems

Temporal logics

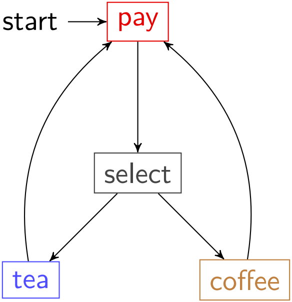
Well-known formalism for specifying and verifying complex systems

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*

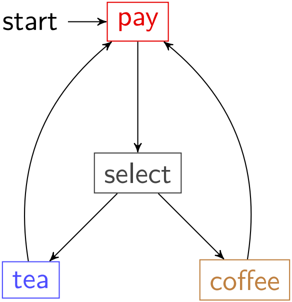


Transition system for a simple vending machine

Temporal logics

Well-known formalism for specifying and verifying complex systems

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



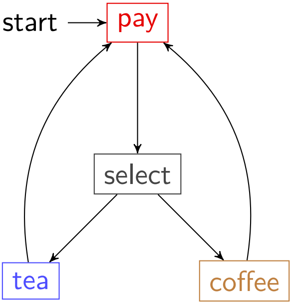
Transition system for a simple vending machine

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

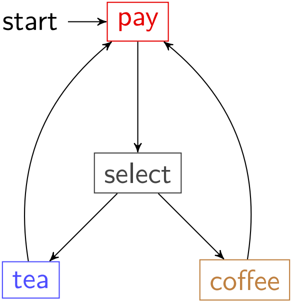
2. Express desired properties as *formulas* in a **temporal logic**

$$\text{Always}(\text{Eventually}(\text{pay}))$$

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

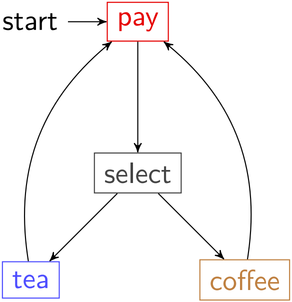
2. Express desired properties as *formulas* in a **temporal logic**

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

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

Multi-component scenarios

- States are simply atomic points

Multi-component scenarios

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

Multi-component scenarios

- 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

Multi-component scenarios

- 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

Multi-component scenarios

- 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

Multi-component scenarios

- 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?

Multi-component scenarios

- 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?

Multi-component scenarios

- 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

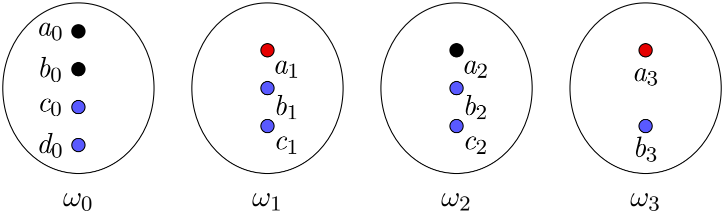
- Standard LTL traces: *sequences of states*

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

Counterpart paradigm

- Standard LTL traces: *sequences of states*

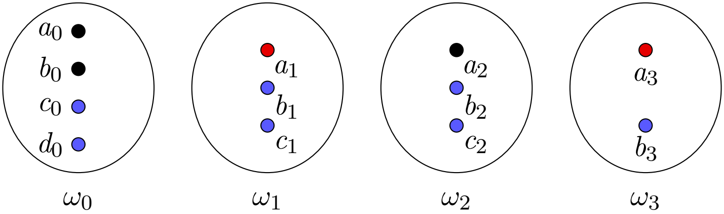
- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds*



Counterpart paradigm

- Standard LTL traces: *sequences of states*

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

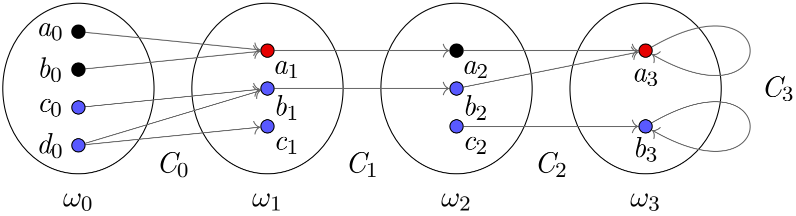


How do we represent transitions?

Counterpart paradigm

- Standard LTL traces: *sequences of states*

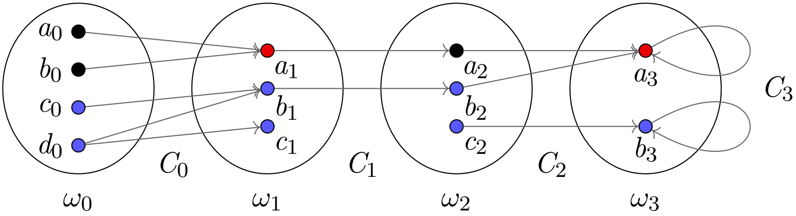
- Associate to each state a set of individuals, called **worlds**
- Our traces: *sequences of worlds, connected with **counterpart relations***



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***

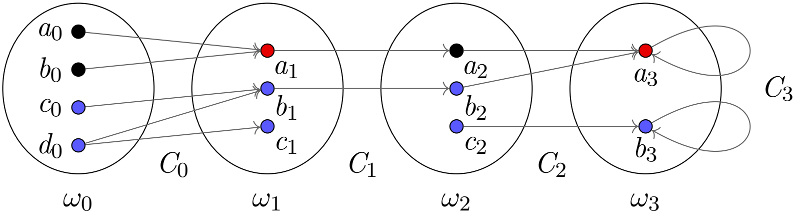


- Intuition: individuals connected by a relation *are the same after one step*

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**

QLTL

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

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2$$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2$$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

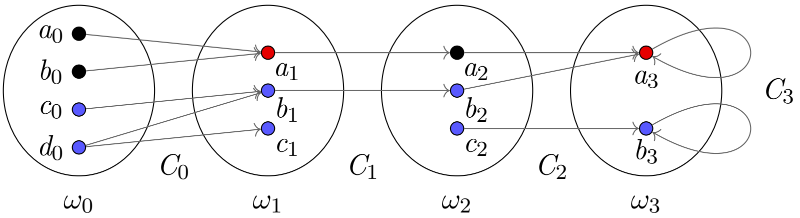
- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ

QLTL

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

$$\phi := \text{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \text{Next}(\phi) \mid \phi_1 \,\text{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ

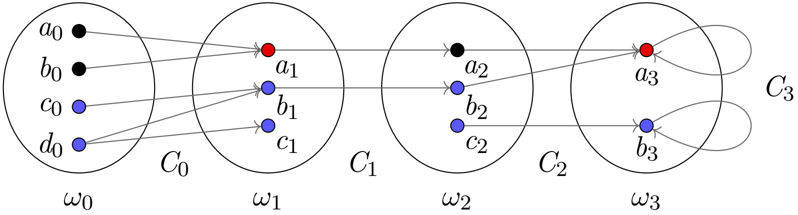


QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1\,\mathsf{Until}\,\phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ

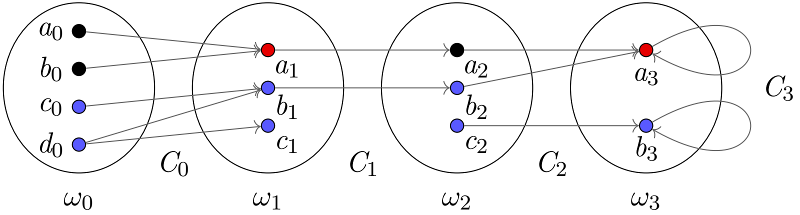


- $a_0 \vDash_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1\,\mathsf{Until}\,\phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ



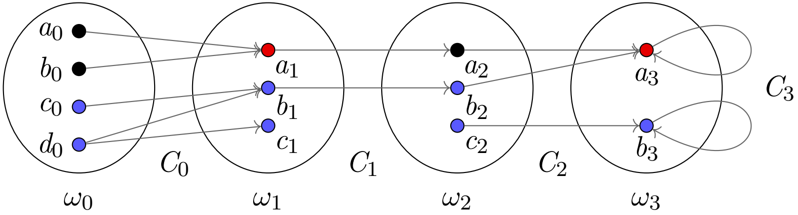
- $a_0 \models_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$
- $c_1 \models_{\omega_1} \neg\mathsf{Next}(\mathsf{Red}(x))$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1\,\mathsf{Until}\,\phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ



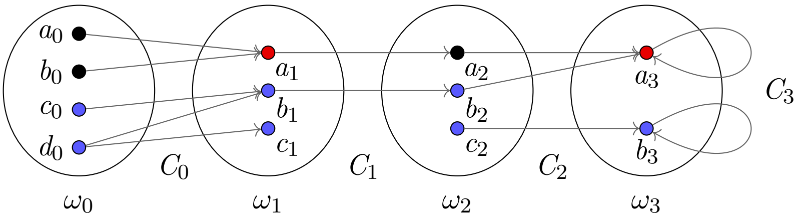
- $a_0 \models_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$
- $c_1 \models_{\omega_1} \neg\mathsf{Next}(\mathsf{Red}(x))$
- $c_0 \models_{\omega_0} \mathsf{Blue}(x)\,\mathsf{Until}\,\mathsf{Red}(x)$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ



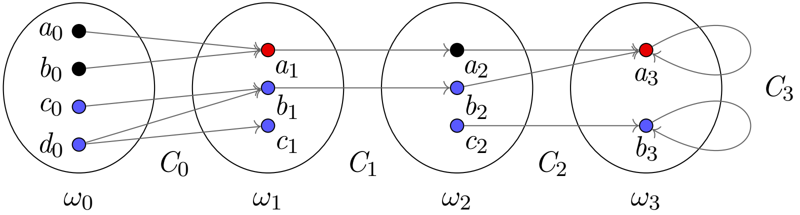
- $a_0 \vDash_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$
- $c_1 \vDash_{\omega_1} \neg\mathsf{Next}(\mathsf{Red}(x))$
- $c_0 \vDash_{\omega_0} \mathsf{Blue}(x)\,\mathsf{Until}\,\mathsf{Red}(x)$
- $(a_0, b_0) \vDash_{\omega_0} \mathsf{Next}(x = y)$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1\, \mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ



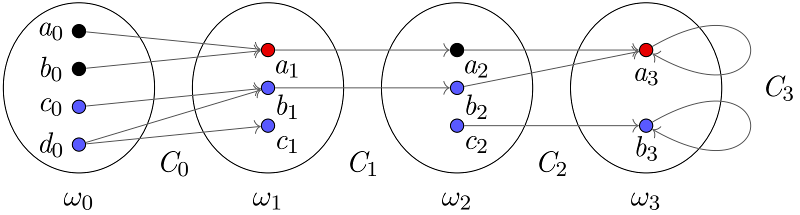
- $a_0 \vDash_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$
- $c_1 \vDash_{\omega_1} \neg\mathsf{Next}(\mathsf{Red}(x))$
- $c_0 \vDash_{\omega_0} \mathsf{Blue}(x)\, \mathsf{Until}\, \mathsf{Red}(x)$
- $(a_0, b_0) \vDash_{\omega_0} \mathsf{Next}(x = y)$
- $() \vDash_{w_2} \exists x.\mathsf{Next}(\mathsf{Blue}(x))$

QLTL

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

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- Semantics: *for each world*, define the (tuples of) individuals satisfying ϕ



- $a_0 \vDash_{\omega_0} \mathsf{Next}(\mathsf{Red}(x))$
- $c_1 \vDash_{\omega_1} \neg\mathsf{Next}(\mathsf{Red}(x))$
- $c_0 \vDash_{\omega_0} \mathsf{Blue}(x) \,\mathsf{Until}\, \mathsf{Red}(x)$
- $(a_0, b_0) \vDash_{\omega_0} \mathsf{Next}(x = y)$
- $() \vDash_{w_2} \exists x.\mathsf{Next}(\mathsf{Blue}(x))$
- $(a_0, c_0) \vDash_{\omega_0} (\neg(x = y)) \,\mathsf{Until}\, (x = y)$

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*

$$\textit{Counterpart model} \approx \underbrace{\textit{a } \mathbf{Rel}\textit{-enriched presheaf on a category } \mathcal{W}}_{\textit{Relational presheaf}}$$

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*

$$\textit{Counterpart model} \approx \underbrace{\textit{a } \mathbf{Rel}\textit{-enriched presheaf on a category } \mathcal{W}}_{\textit{Relational presheaf}}$$
$$+ \underbrace{\textit{a class of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}}$$

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*

$$
\begin{aligned}
\textit{Counterpart model} \approx\ & \underbrace{\textit{a } \mathbf{Rel}\textit{-enriched presheaf on a category } \mathcal{W}}_{\textit{Relational presheaf}} \\
+\ & \underbrace{\textit{a class of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}}
\end{aligned}
$$

- The *relational presheaf* assigns worlds and counterpart relations to 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*

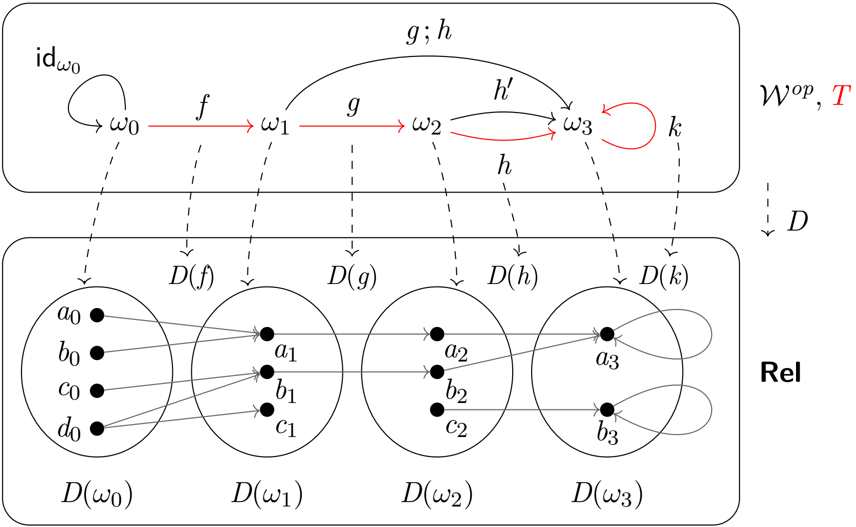
$$\begin{aligned} \textit{Counterpart model} \approx\ & \underbrace{\textit{a } \mathbf{Rel}\textit{-enriched presheaf on a category } \mathcal{W}}_{\textit{Relational presheaf}} \\ +\ & \underbrace{\textit{a class of selected morphisms of } \mathcal{W}}_{\textit{Temporal structure}} \end{aligned}$$

- The *relational presheaf* assigns worlds and counterpart relations to states
- The *temporal structure* identifies the one-step transitions of the model

Relational presheaves – Example

$$
\begin{tikzcd}
\omega_0 \arrow[loop left, "\mathsf{id}_{\omega_0}"] \arrow[r, "f"] & \omega_1 \arrow[r, "g"] \arrow[rr, bend left=40, "g\,;h"] & \omega_2 \arrow[r, bend left, "h'"] \arrow[r, bend right, "h"'] & \omega_3 \arrow[loop right, "k"]
\end{tikzcd}
\qquad \mathcal{W}^{op},\ T
$$

Relational presheaves – Example



Categorical and classical semantics

- *How is the semantics of our logic defined?*

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*
- Semantics, the categorical way via *classical attributes*: *for each world*, define the *set* of (tuples of) individuals satisfying ϕ

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*
- Semantics, the categorical way via *classical attributes*: *for each world*, define the *set* of (tuples of) individuals satisfying ϕ
- *How are the two semantics and their models related?*

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*
- Semantics, the categorical way via *classical attributes*: *for each world*, define the *set* of (tuples of) individuals satisfying ϕ
- *How are the two semantics and their models related?*

Theorem (Classical models $\leftrightarrow$ categorical models)

($\rightarrow$) For any classical counterpart model M, there exists a categorical counterpart model W which satisfies exactly the same QLTL formulae ϕ.

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*
- Semantics, the categorical way via *classical attributes*: *for each world*, define the *set* of (tuples of) individuals satisfying ϕ
- *How are the two semantics and their models related?*

Theorem (Classical models $\leftrightarrow$ categorical models)

($\rightarrow$) For any classical counterpart model M, there exists a categorical counterpart model W which satisfies exactly the same QLTL formulae ϕ.

- Idea: construct the freely generated category $\mathcal{W}$ from the transition system; define a relational presheaf $D : \mathcal{W}^{op} \rightarrow$ **Rel** assigning worlds and relations; restrict the morphisms with the temporal structure T.

Categorical and classical semantics

- *How is the semantics of our logic defined?*
- Semantics, the standard way: an *inductively defined relation* between formulae ϕ and (tuples of) individuals *in each world*
- Semantics, the categorical way via *classical attributes*: *for each world*, define the *set* of (tuples of) individuals satisfying ϕ
- *How are the two semantics and their models related?*

Theorem (Classical models $\leftrightarrow$ categorical models)

($\rightarrow$) For any classical counterpart model M, there exists a categorical counterpart model W which satisfies exactly the same QLTL formulae ϕ.

- Idea: construct the freely generated category $\mathcal{W}$ from the transition system; define a relational presheaf $D : \mathcal{W}^{op} \rightarrow$ **Rel** assigning worlds and relations; restrict the morphisms with the temporal structure T.
- The *theorem* has not been formalized, but the *construction* is!

Algebraic QLTL

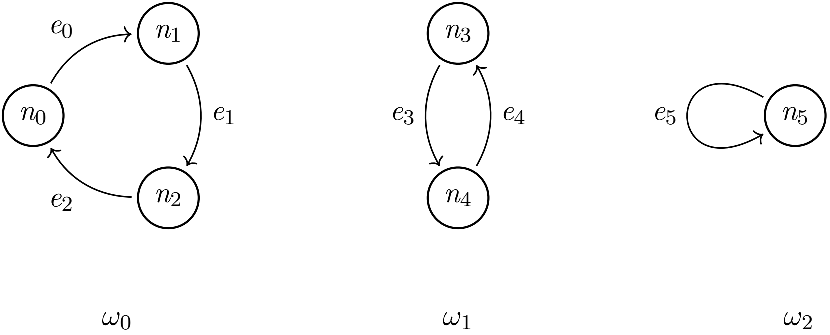
- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*

Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.

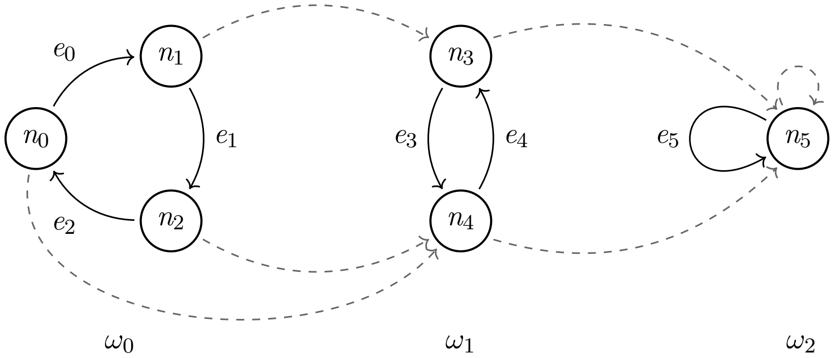
Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:

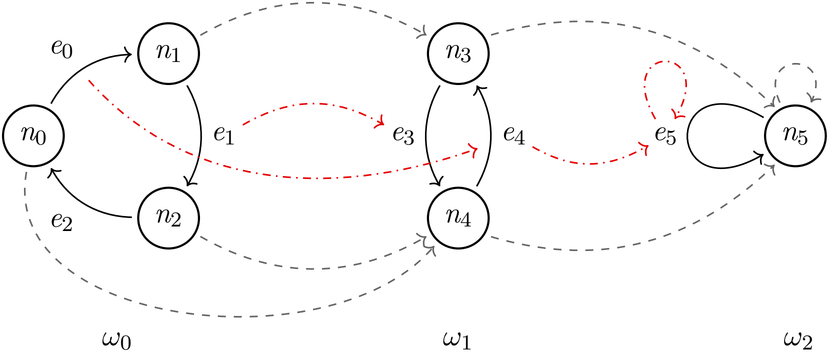


Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:

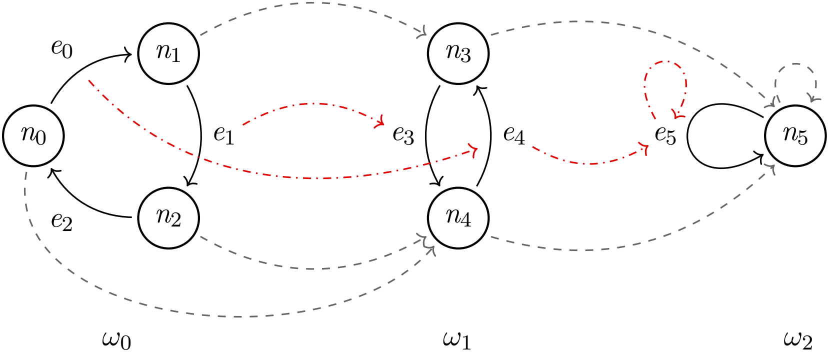


- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:



Algebraic QLTL

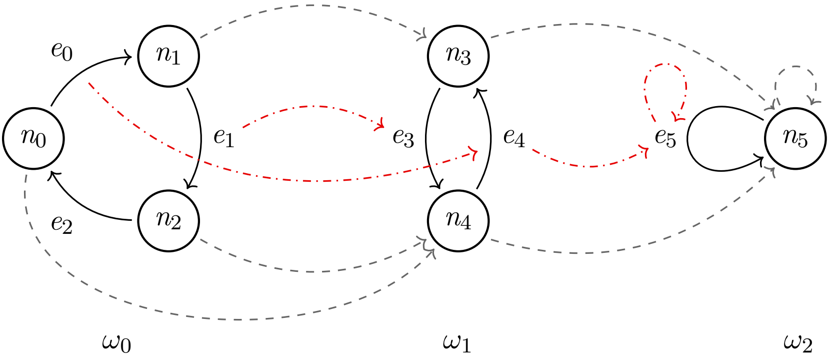
- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:



- *Algebraic* QLTL: equality between *terms*, instead of individuals:

Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:

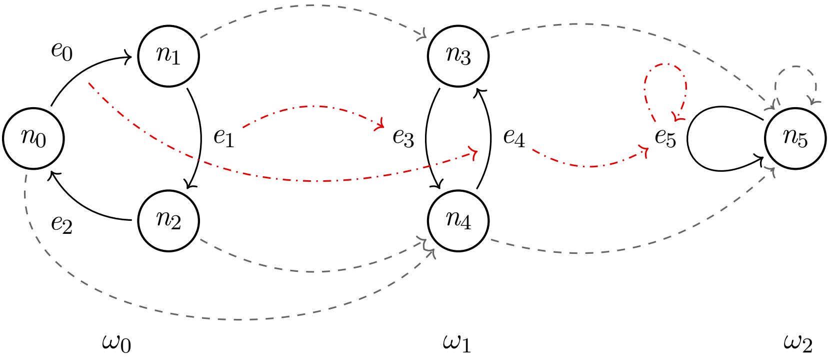


- *Algebraic* QLTL: equality between *terms*, instead of individuals:

$$\textbf{loop}(e) := s(e) = t(e),$$

Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:

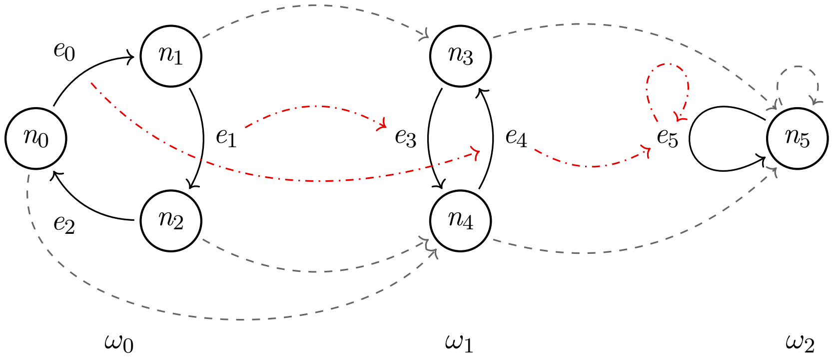


- *Algebraic* QLTL: equality between *terms*, instead of individuals:

$$\textbf{loop}(e) := s(e) = t(e), \quad e_5 \vDash_{\omega_2} \textbf{loop}(x),$$

Algebraic QLTL

- *Worlds-as-algebras*: generalize sets to *many-sorted algebras*
- Examples: graphs, trees, lists, etc.
- A counterpart model on the signature of directed graphs:



- *Algebraic* QLTL: equality between *terms*, instead of individuals:

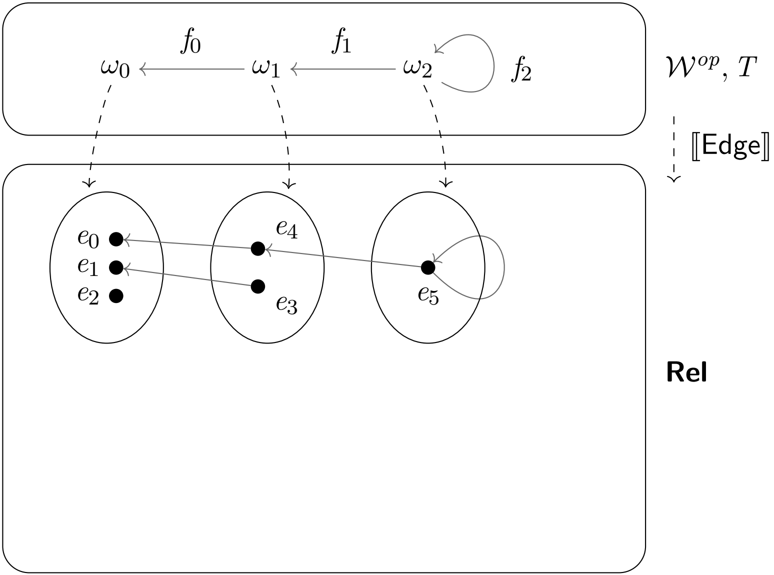
$$\textbf{loop}(e) := s(e) = t(e), \quad e_5 \vDash_{\omega_2} \textbf{loop}(x), \quad e_4 \vDash_{\omega_1} \mathsf{Next}(\textbf{loop}(x)).$$

Algebraic QLTL with relational presheaves

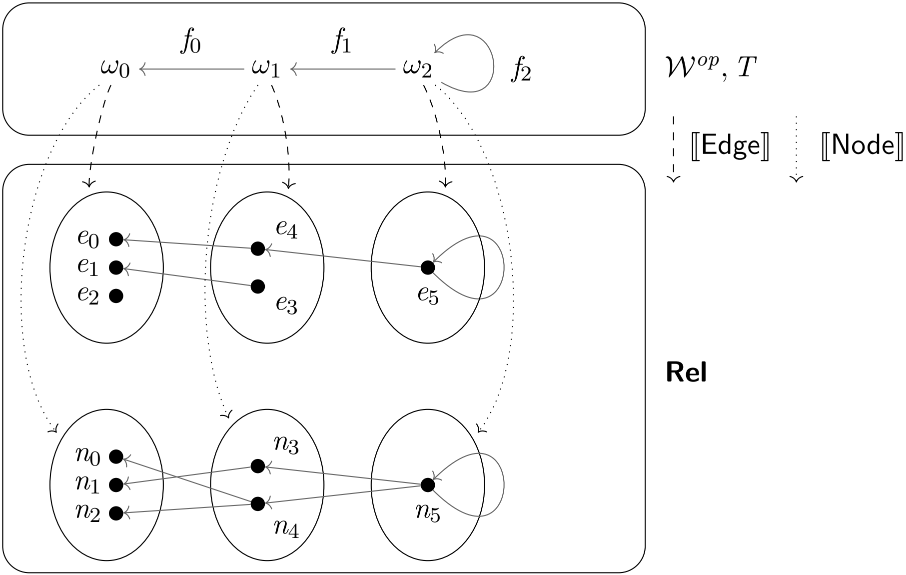
$$\begin{tikzcd} \omega_0 & \omega_1 \arrow[l, "f_0"'] & \omega_2 \arrow[l, "f_1"'] \arrow[loop right, "f_2"] \end{tikzcd}$$

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

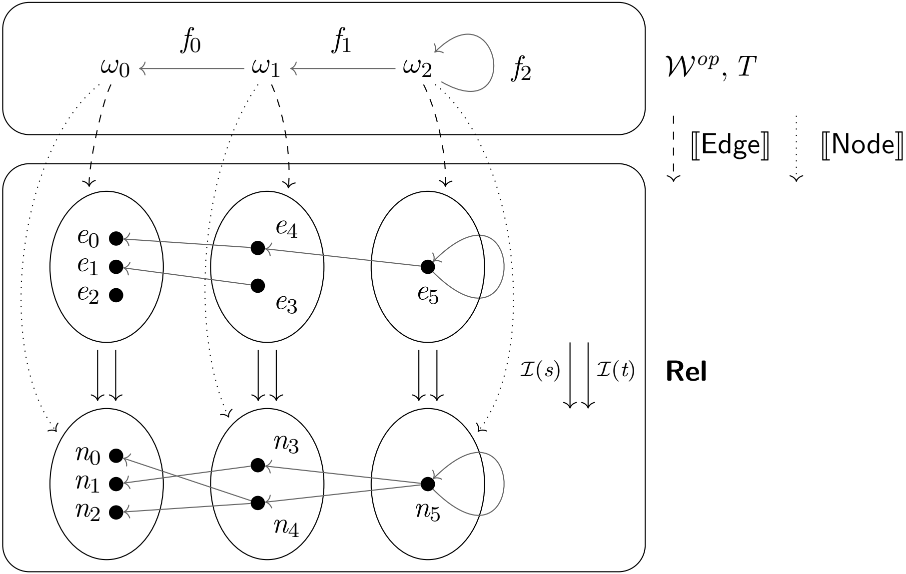
Algebraic QLTL with relational presheaves



Algebraic QLTL with relational presheaves



Algebraic QLTL with relational presheaves



Agda formalization

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

Agda

- 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: *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

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

Categorical semantics with `agda-categories`

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

Categorical semantics with `agda-categories`

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

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- Extremely practical and flexible, no magic involved
- Design choices of the library (e.g. setoid-flavoured reasoning) do not necessarily get in the way of practical applications
- Main definitions used:

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- Extremely practical and flexible, no magic involved
- Design choices of the library (e.g. setoid-flavoured reasoning) do not necessarily get in the way of practical applications
- Main definitions used:
  - Categories, functors, natural transformations

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices of the library (e.g. setoid-flavoured reasoning) do not necessarily get in the way of practical applications
- Main definitions used:
  - Categories, functors, natural transformations
  - **Rel**: category of sets and relations

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices of the library (e.g. setoid-flavoured reasoning) 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`)

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices of the library (e.g. setoid-flavoured reasoning) 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, category of presheaves is complete

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices of the library (e.g. setoid-flavoured reasoning) 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, category of presheaves is complete
  - Relational presheaves (easy to define) and morphisms between them

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- Extremely practical and flexible, no magic involved
- Design choices of the library (e.g. setoid-flavoured reasoning) 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, category of presheaves is complete
  - Relational presheaves (easy to define) and morphisms between them
- Functoriality and setoid-equality preservation can be annoying to prove

Categorical semantics with `agda-categories`

- The *de-facto* (non-univalent) standard *category theory library* in Agda
- ☺ Extremely practical and flexible, no magic involved
- ☺ Design choices of the library (e.g. setoid-flavoured reasoning) 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, category of presheaves is complete
  - Relational presheaves (easy to define) and morphisms between them
- 😐 Functoriality and setoid-equality preservation can be annoying to prove
- 😐 Relatively limited use of the constructions of the library in our setting

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

Negation is defined as $\neg A := A \to \bot$,
and the law of excluded middle $A \vee \neg A$ is *not* provable

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

  Negation is defined as $\neg A := A \to \bot$,
  and the law of excluded middle $A \vee \neg A$ is *not* provable

- *Consequence:* the logic we embed in Agda is also intuitionistic!

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

$$\text{Negation is defined as } \neg A := A \to \bot,$$
$$\text{and the law of excluded middle } A \vee \neg A \text{ is } \textit{not} \text{ provable}$$

- *Consequence:* the logic we embed in Agda is also intuitionistic!
- e.g., we also cannot prove in Agda that, for any choice of QLTL formula ϕ and counterpart model M,

$$M \vDash \neg\neg\phi \iff \phi.$$

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

  Negation is defined as $\neg A := A \to \bot$,
  and the law of excluded middle $A \vee \neg A$ is *not* provable

- *Consequence:* the logic we embed in Agda is also intuitionistic!
- e.g., we also cannot prove in Agda that, for any choice of QLTL formula ϕ and counterpart model M,

$$M \vDash \neg\neg\phi \iff \phi.$$

- *Problem*: in temporal logic we usually define the essential operators of the logic, and then derive the other ones with negation:

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

  Negation is defined as $\neg A := A \to \bot$,
  and the law of excluded middle $A \vee \neg A$ is *not* provable

- *Consequence:* the logic we embed in Agda is also intuitionistic!
- e.g., we also cannot prove in Agda that, for any choice of QLTL formula ϕ and counterpart model M,

$$M \vDash \neg\neg\phi \iff \phi.$$

- *Problem*: in temporal logic we usually define the essential operators of the logic, and then derive the other ones with negation:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

  Negation is defined as $\neg A := A \to \bot$,
  and the law of excluded middle $A \vee \neg A$ is *not* provable

- *Consequence:* the logic we embed in Agda is also intuitionistic!
- e.g., we also cannot prove in Agda that, for any choice of QLTL formula ϕ and counterpart model M,

$$M \vDash \neg\neg\phi \iff \phi.$$

- *Problem*: in temporal logic we usually define the essential operators of the logic, and then derive the other ones with negation:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- In Agda, having $\wedge$ but not $\vee$ in QLTL is *not* equivalent to having both!

Embedding temporal logics in Agda

- The metalogic provided by Agda is *intuitionistic* in nature:

  Negation is defined as $\neg A := A \to \bot$,
  and the law of excluded middle $A \vee \neg A$ is *not* provable

- *Consequence:* the logic we embed in Agda is also intuitionistic!
- e.g., we also cannot prove in Agda that, for any choice of QLTL formula ϕ and counterpart model M,

$$M \vDash \neg\neg\phi \iff \phi.$$

- *Problem*: in temporal logic we usually define the essential operators of the logic, and then derive the other ones with negation:

$$\phi := \mathsf{true} \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{Next}(\phi) \mid \phi_1 \,\mathsf{Until}\, \phi_2 \mid x = y \mid \exists x.\phi \mid P(x)$$

- In Agda, having $\wedge$ but not $\vee$ in QLTL is *not* equivalent to having both!
- *Problem:* it can be tricky to show contradictions with negated formulae

Positive normal forms

- *How can we tackle these issues?*

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

an extension of QLTL which can express ***exactly***
the same properties without using negation

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

  an extension of QLTL which can express ***exactly***
  the same properties without using negation

- Simplifies algorithms and model checking with temporal logics

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

> *an extension of QLTL which can express* ***exactly***
> *the same properties without using negation*

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

*an extension of QLTL which can express **exactly** the same properties without using negation*

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*
2. *Separately* prove that the logic given is a PNF of the original logic

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

*an extension of QLTL which can express **exactly***
the same properties without using negation

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*
2. *Separately* prove that the logic given is a PNF of the original logic
3. Use LEM *only* in the PNF conversion proof, *not* when using the logic

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

*an extension of QLTL which can express **exactly***
the same properties without using negation

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*
2. *Separately* prove that the logic given is a PNF of the original logic
3. Use LEM *only* in the PNF conversion proof, *not* when using the logic

$\Rightarrow$ Guarantees that no extra expressivity is gained/lost

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

an extension of QLTL which can express ***exactly***
the same properties without using negation

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*
2. *Separately* prove that the logic given is a PNF of the original logic
3. Use LEM *only* in the PNF conversion proof, *not* when using the logic

$\Rightarrow$ Guarantees that no extra expressivity is gained/lost

$\Rightarrow$ Proving this theorem gives a program to *convert* formulae in PNF

Positive normal forms

- *How can we tackle these issues?*
- Positive normal form (PNF):

*an extension of QLTL which can express **exactly** the same properties without using negation*

- Simplifies algorithms and model checking with temporal logics
- Our approach:

1. Provide the logic in Agda with a *positive normal form*
2. *Separately* prove that the logic given is a PNF of the original logic
3. Use LEM *only* in the PNF conversion proof, *not* when using the logic

$\Rightarrow$ Guarantees that no extra expressivity is gained/lost

$\Rightarrow$ Proving this theorem gives a program to *convert* formulae in PNF

$\Rightarrow$ Directly prove another formula instead of working with contradiction

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

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$
$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2$$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$

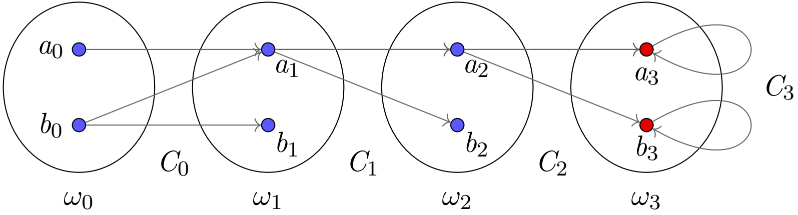
$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2 \mid \underline{\mathsf{NextF}}(\phi) \mid \phi_1 \underline{\mathsf{UntilF}} \phi_2 \mid \phi_1 \underline{\mathsf{Then}} \phi_2$$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$

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

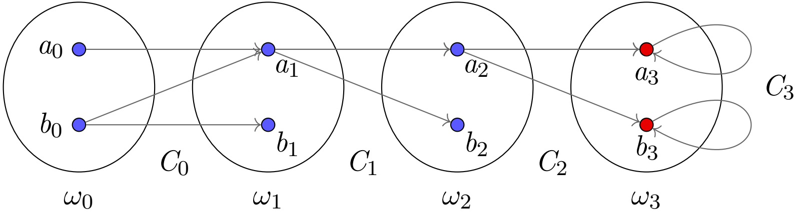


Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$

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



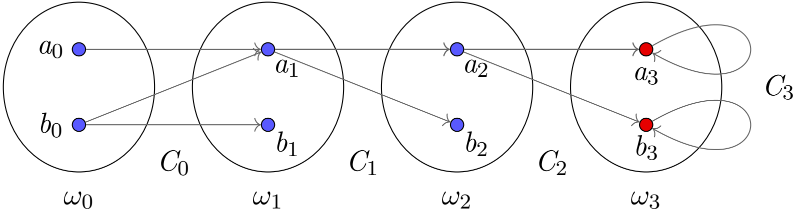
- $\neg\mathsf{Next}(\phi) \equiv \mathsf{NextF}(\neg\phi)$
- $\neg(\phi_1 \mathsf{Until} \phi_2) \equiv (\neg\phi_2)\mathsf{Then}(\neg\phi_1 \wedge \neg\phi_2)$
- $\neg(\phi_1 \mathsf{WUntil} \phi_2) \equiv (\neg\phi_2)\mathsf{UntilF}(\neg\phi_1 \wedge \neg\phi_2)$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \mathsf{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$

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

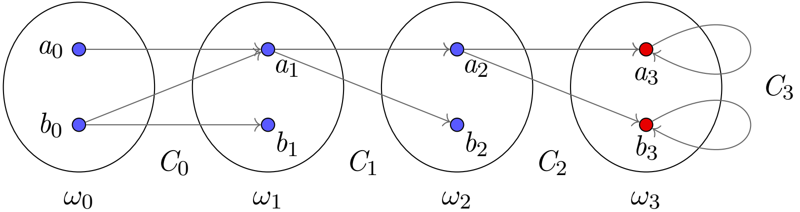


- $\neg\mathsf{Next}(\phi) \equiv \mathsf{NextF}(\neg\phi)$
- $\neg(\phi_1 \mathsf{Until} \phi_2) \equiv (\neg\phi_2)\mathsf{Then}(\neg\phi_1 \wedge \neg\phi_2)$
- $\neg(\phi_1 \mathsf{WUntil} \phi_2) \equiv (\neg\phi_2)\mathsf{UntilF}(\neg\phi_1 \wedge \neg\phi_2)$
- $b_0 \vDash_{\omega_0} \mathsf{NextF}(\mathsf{Blue}(x))$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \text{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$
$$\mid \text{Next}(\phi) \mid \phi_1 \text{Until} \phi_2 \mid \phi_1 \text{WUntil} \phi_2 \mid \underline{\text{NextF}}(\phi) \mid \phi_1 \underline{\text{UntilF}} \phi_2 \mid \phi_1 \underline{\text{Then}} \phi_2$$



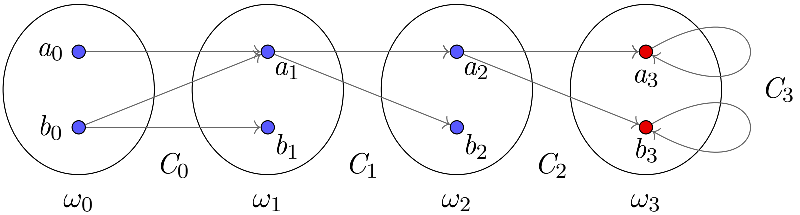
- $\neg\text{Next}(\phi) \equiv \text{NextF}(\neg\phi)$
- $\neg(\phi_1 \text{Until} \phi_2) \equiv (\neg\phi_2)\text{Then}(\neg\phi_1 \wedge \neg\phi_2)$
- $\neg(\phi_1 \text{WUntil} \phi_2) \equiv (\neg\phi_2)\text{UntilF}(\neg\phi_1 \wedge \neg\phi_2)$

- $b_0 \vDash_{\omega_0} \text{NextF}(\text{Blue}(x))$
- $b_1 \vDash_{\omega_1} \text{NextF}(\text{Blue}(x))$

Positive normal forms for QLTL

- PNF for QLTL:

$$\psi := \text{true} \mid x = y \mid P(x), \quad \phi := \psi \mid \neg\psi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \forall x.\phi$$
$$\mid \mathsf{Next}(\phi) \mid \phi_1 \mathsf{Until} \phi_2 \mid \phi_1 \mathsf{WUntil} \phi_2 \mid \underline{\mathsf{NextF}}(\phi) \mid \phi_1 \underline{\mathsf{UntilF}} \phi_2 \mid \phi_1 \underline{\mathsf{Then}} \phi_2$$



- $\neg\mathsf{Next}(\phi) \equiv \mathsf{NextF}(\neg\phi)$
- $\neg(\phi_1 \mathsf{Until} \phi_2) \equiv (\neg\phi_2)\mathsf{Then}(\neg\phi_1 \wedge \neg\phi_2)$
- $\neg(\phi_1 \mathsf{WUntil} \phi_2) \equiv (\neg\phi_2)\mathsf{UntilF}(\neg\phi_1 \wedge \neg\phi_2)$
- $b_0 \vDash_{\omega_0} \mathsf{NextF}(\mathsf{Blue}(x))$
- $b_1 \vDash_{\omega_1} \mathsf{NextF}(\mathsf{Blue}(x))$
- $a_0 \vDash_{\omega_0} \mathsf{Blue}(x) \mathsf{UntilF}\, \mathsf{Red}(x)$

Conclusion and Future Work

In this work we present the categorical semantics of a counterpart-based temporal logic and formalize it in Agda along with results on its PNF.

- Many possible extensions of this work:

Conclusion and Future Work

In this work we present the categorical semantics of a counterpart-based temporal logic and formalize it 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 the categorical semantics of a counterpart-based temporal logic and formalize it 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 the categorical semantics of a counterpart-based temporal logic and formalize it 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 the categorical semantics of a counterpart-based temporal logic and formalize it 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 the categorical semantics of a counterpart-based temporal logic and formalize it 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

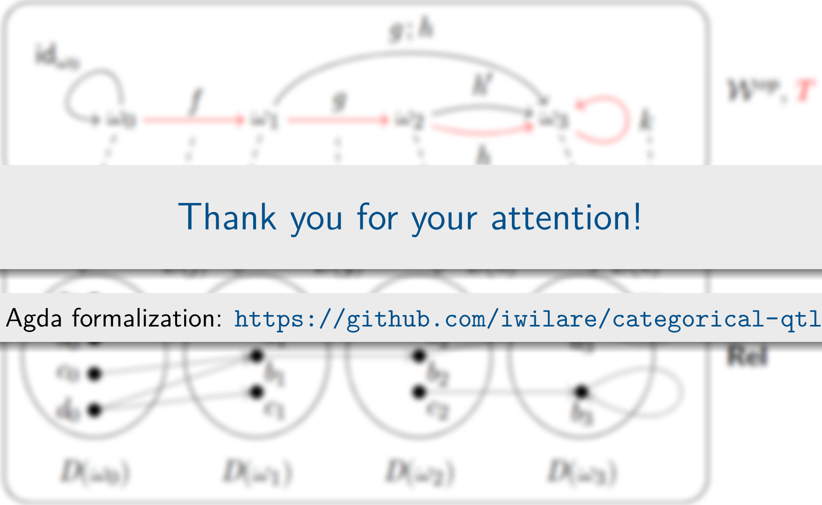
In this work we present the categorical semantics of a counterpart-based temporal logic and formalize it in Agda along with results on its PNF.

- 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 the categorical semantics of a counterpart-based temporal logic and formalize it 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/categorical-qtl`