Traces via Strategies

(Games via Coalgebra)

Ben Plummer, Corina Cîrstea

University of Southampton

April, 2024

Outline

- Games
- Traces
- Representing games coalgebraically
- Strategies
- Traces via strategies

Motivation: controller synthesis

- Model the possible actions of the controller and the environment as a game.

Motivation: controller synthesis

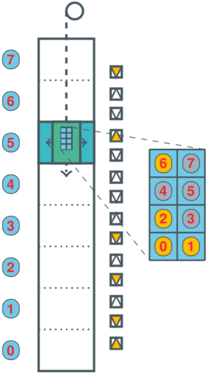
- Model the possible actions of the controller and the environment as a game.
- We have specification (as a logical formula).

Motivation: controller synthesis

- Model the possible actions of the controller and the environment as a game.
- We have specification (as a logical formula).
- Synthesis question: is there are a controller strategy which every play satisfies the specification?

Motivation: controller synthesis

- Model the possible actions of the controller and the environment as a game.
- We have specification (as a logical formula).
- Synthesis question: is there are a controller strategy which every play satisfies the specification?
- Example "every request is served" (a liveness property)



Two-player games

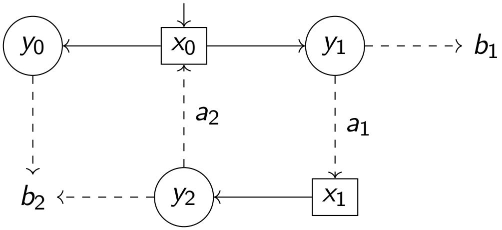
- Bipartite game graph

Two-player games

- Bipartite game graph
- Observation after environment transition

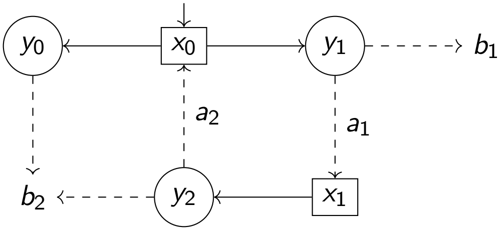
Two-player games

- Bipartite game graph
- Observation after environment transition



Two-player games

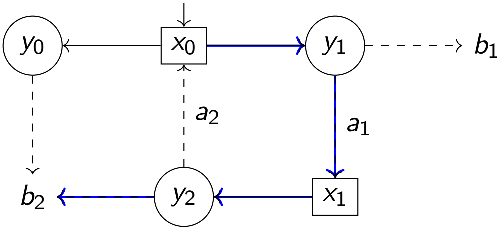
- Bipartite game graph
- Observation after environment transition



- A play is a sequence of states and observations, arising from controller and environment moves, ending in a terminating observation.

Two-player games

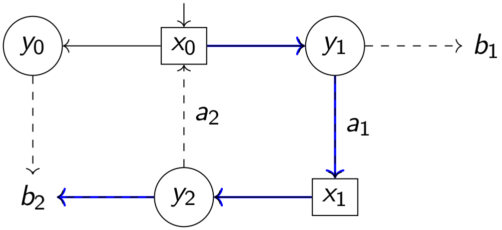
- Bipartite game graph
- Observation after environment transition



- A play is a sequence of states and observations, arising from controller and environment moves, ending in a terminating observation. e.g. $x_0 y_1 a_1 x_1 y_2 b_2$

Two-player games

- Bipartite game graph
- Observation after environment transition



- A play is a sequence of states and observations, arising from controller and environment moves, ending in a terminating observation. e.g. $x_0 y_1 a_1 x_1 y_2 b_2$
- A strategy is a partial function which extends partial plays, it must be defined over all partial plays which conform to it.

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
-
-

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a function:

$$c : X \to P(B + A \times X)$$

-

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a function:

$$c : X \to P(B + A \times X)$$

- A *trace* starting at a state $x_0 \in X$ is a sequence

$$a_1 a_2, \dots a_n b \in A^* B$$

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a function:

$$c : X \to P(B + A \times X)$$

- A *trace* starting at a state $x_0 \in X$ is a sequence

$$a_1 a_2, \ldots a_n b \in A^* B$$

such that there is an *execution*

$$x_0 a_1 x_1 a_2 \ldots a_n x_n b \in (XA)^* XB$$

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a function:

$$c : X \to P(B + A \times X)$$

- A *trace* starting at a state $x_0 \in X$ is a sequence

$$a_1 a_2, \ldots a_n b \in A^* B$$

such that there is an *execution*

$$x_0 a_1 x_1 a_2 \ldots a_n x_n b \in (XA)^* XB$$

with the property

$$\forall i < n : (a_{i+1}, x_{i+1}) \in c(x_i) \text{ and } b \in c(x_n)$$

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a relation:

$$R \subseteq X \times (B + A \times X)$$

- A *trace* starting at a state $x_0 \in X$ is a sequence

$$a_1 a_2, \ldots a_n b \in A^* B$$

such that there is an *execution*

$$x_0 a_1 x_1 a_2 \ldots a_n x_n b \in (XA)^* XB$$

defined by the property

$$\forall i < n : R(x_i, (a_{i+1}, x_{i+1})) \text{ and } R(x_n, b)$$

(Finite) Traces for labelled transition systems

- A *trace* is a sequence of observations from a process.
- A labelled transition system with termination is a relation:

$$R \subseteq X \times (B + A \times X)$$

- A *trace* starting at a state $x_0 \in X$ is a sequence

$$a_1 a_2, \ldots a_n b \in A^* B$$

such that there is an *execution*

$$x_0 a_1 x_1 a_2 \ldots a_n x_n b \in (XA)^* XB$$

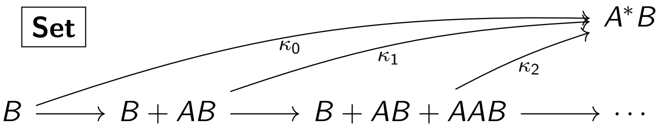
defined by the property

$$\forall i < n : R(x_i, (a_{i+1}, x_{i+1})) \text{ and } R(x_n, b)$$

P is a monad with $\mathbf{Kl}(P) \cong \mathbf{Rel}$

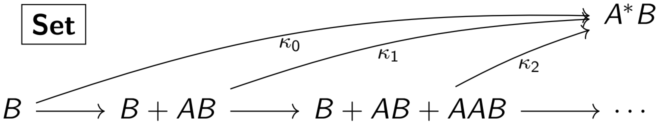
Traces, coalgebraically

A^*B is the *initial algebra* for the functor $B + A(-) : \mathbf{Set} \to \mathbf{Set}$

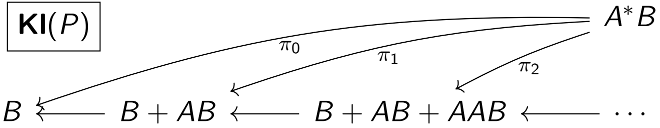


Traces, coalgebraically

A^*B is the *initial algebra* for the functor $B + A(-) : \mathbf{Set} \to \mathbf{Set}$



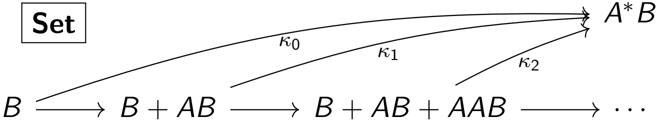
General categorical machinery allows us to lift this chain to the category of relations, and reverse the arrows[1]:



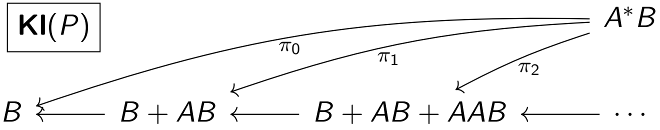
[1] With various assumptions, which we will come back to later

Traces, coalgebraically

A^*B is the *initial algebra* for the functor $B + A(-) : \mathbf{Set} \to \mathbf{Set}$



General categorical machinery allows us to lift this chain to the category of relations, and reverse the arrows[1]:



Thus A^*B is a *final coalgebra* in the category of relations!

[1] With various assumptions, which we will come back to later

Traces by coinduction

For every LTS $c : X \to P(B + A \times X)$, there is a *unique coalgebra morphism* into A^*B.

$$\boxed{\textbf{Rel}} \qquad \begin{array}{ccc} X & \dashrightarrow & A^*B \\ \downarrow c & & \downarrow \wr \\ B + A \times X & \dashrightarrow & B + A \times A^*B \end{array}$$

This dashed morphism in **Rel** is a function $X \to P(A^*B)$ which assigns each state to it's set of traces!

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$
- Now use a modified version $H_X := X \times (B + A \times (-))$

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$
- Now use a modified version $H_X := X \times (B + A \times (-))$
- With $c^* : X \to PH_X(X)$ defined as the composite

$$(X \xrightarrow{\langle \mathrm{id}, c \rangle} X \times P(B + A \times X) \xrightarrow{\mathrm{stl}} P(X \times (B + A \times X))$$

$$x \mapsto \{(x, u) \mid u \in c(x)\}$$

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$
- Now use a modified version $H_X := X \times (B + A \times (-))$
- With $c^* : X \to PH_X(X)$ defined as the composite

$$(X \xrightarrow{\langle \mathrm{id}, c\rangle} X \times P(B + A \times X) \xrightarrow{\mathrm{stl}} P(X \times (B + A \times X))$$

$$x \mapsto \{(x, u) \mid u \in c(x)\}$$

- With the same apparatus as before, we can obtain an execution map $\mathrm{exec}_c : X \to P((XA)^* XB)$

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$
- Now use a modified version $H_X := X \times (B + A \times (-))$
- With $c^* : X \to PH_X(X)$ defined as the composite

$$(X \xrightarrow{\langle \mathrm{id}, c \rangle} X \times P(B + A \times X) \xrightarrow{\mathrm{stl}} P(X \times (B + A \times X))$$

$$x \mapsto \{(x, u) \mid u \in c(x)\}$$

- With the same apparatus as before, we can obtain an execution map $\mathrm{exec}_c : X \to P((XA)^* XB)$
- And it follows from a general coalgebraic result that:

Executions by coinduction

- We have been using a functor $H := B + A \times (-)$
- With a PH-coalgebra $c : X \to PHX$
- Now use a modified version $H_X := X \times (B + A \times (-))$
- With $c^* : X \to PH_X(X)$ defined as the composite

$$(X \xrightarrow{\langle \mathrm{id}, c\rangle} X \times P(B + A \times X) \xrightarrow{\mathrm{stl}} P(X \times (B + A \times X))$$

$$x \mapsto \{(x, u) \mid u \in c(x)\}$$

- With the same apparatus as before, we can obtain an execution map $\mathrm{exec}_c : X \to P((XA)^* XB)$
- And it follows from a general coalgebraic result that:

$$X \xrightarrow{\mathrm{exec}_c} (XA)^* XB \xrightarrow{f_{\pi_2}} A^* B \quad \text{where } \pi_2 : H_X(Y) \to H(Y).$$

$$X \xrightarrow{\mathrm{tr}_c} A^* B$$

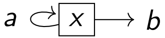
Rel

Recap

Because the monad P has lots of nice properties, we automatically get trace/execution maps:

$$X \xrightarrow{\text{exec}_c} (XA)^* XB \xrightarrow{f_{\pi_2}} A^* B, \qquad \text{tr}_c : X \to A^* B$$

Rel

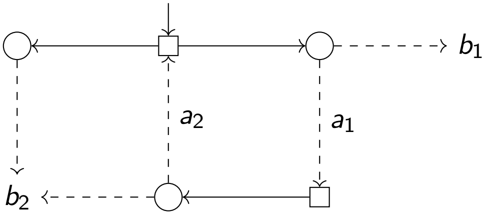


$$\text{exec}_c(x) = \{xb, xaxb, xaxaxb, xaxaxaxb, \ldots\}$$

$$\text{tr}_c(x) = \{b, ab, aab, aaab, \ldots\}$$

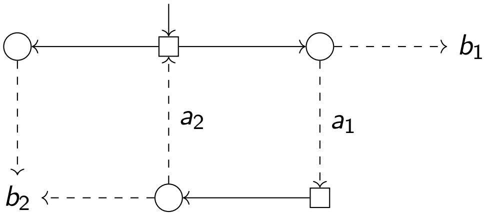
Games

Recall:

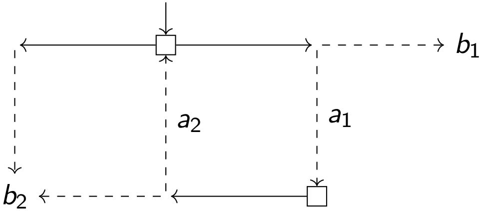


How do we turn this into a function $X \to M(HX)$?
i.e. Which monad M do we choose?

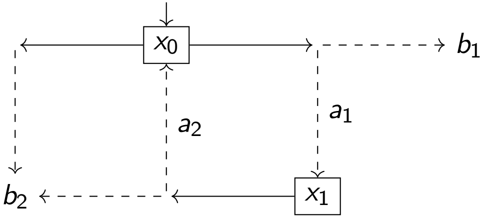
Finding the monad



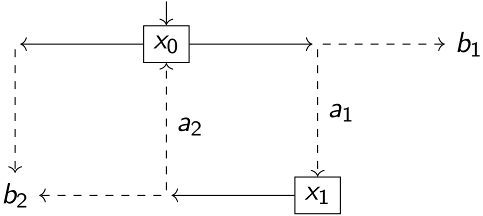
Finding the monad



Finding the monad

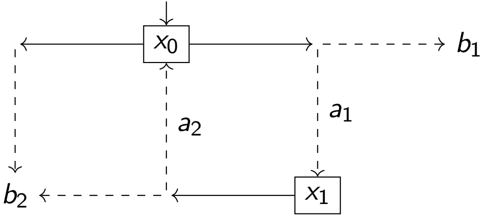


Finding the monad



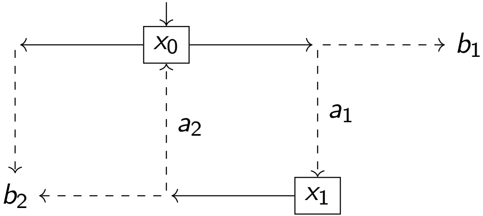
$$c : X \to PP(B + A \times X)$$

Finding the monad



$$c : X \to PP(B + A \times X)$$
$$c(x_0) = \{\{b_1, a_1x_1\}, \{b_2\}\}$$
$$c(x_1) = \{\{b_2, a_2x_0\}\}$$

Finding the monad

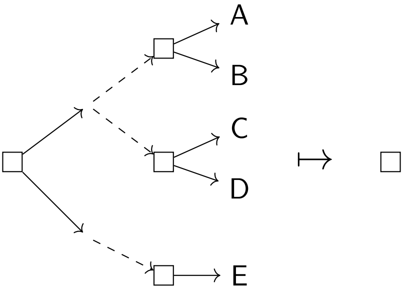


$$c : X \to PP(B + A \times X)$$
$$c(x_0) = \{\{b_1, a_1 x_1\}, \{b_2\}\}$$
$$c(x_1) = \{\{b_2, a_2 x_0\}\}$$

Is PP a monad?

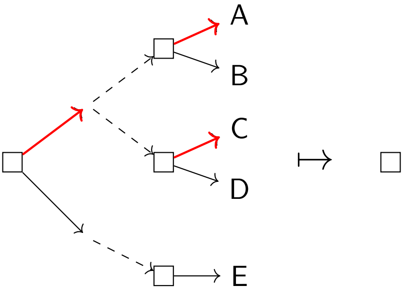
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



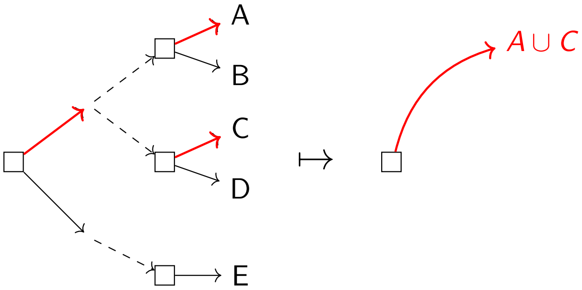
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



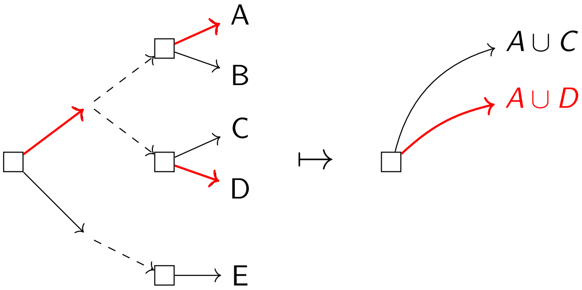
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



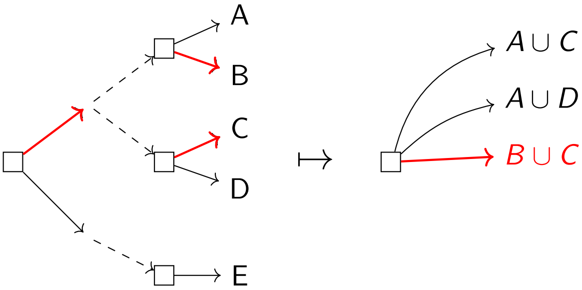
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



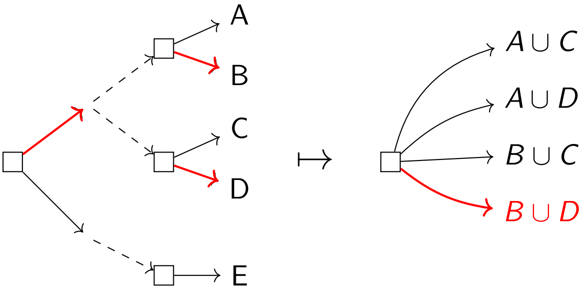
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



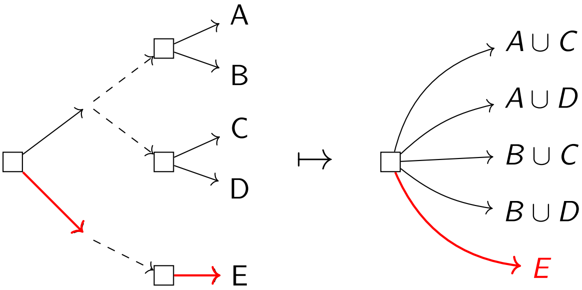
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



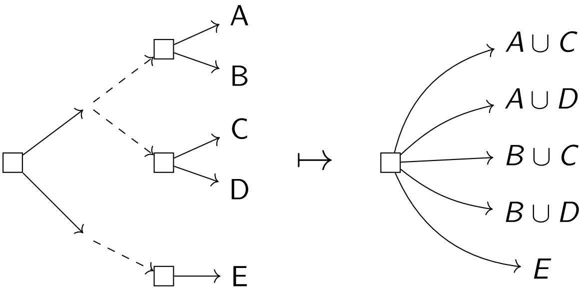
How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$



How do I squash $PPPP(X) \to PP(X)$?

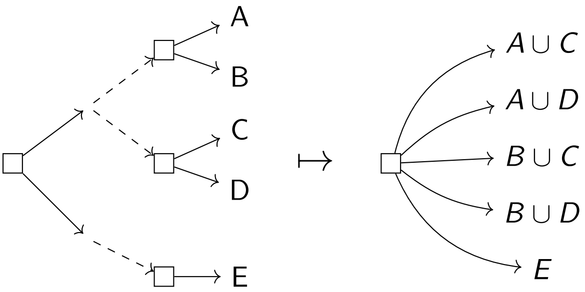
Let $A, B, C, D, E \subseteq X$



$$\{\{\{A, B\}, \{C, D\}\}, \{\{E\}\}\} \mapsto \{A \cup C, A \cup D, B \cup C, B \cup D, E\}$$

How do I squash $PPPP(X) \to PP(X)$?

Let $A, B, C, D, E \subseteq X$

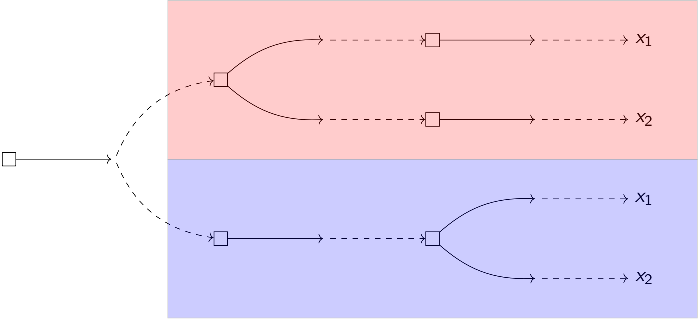


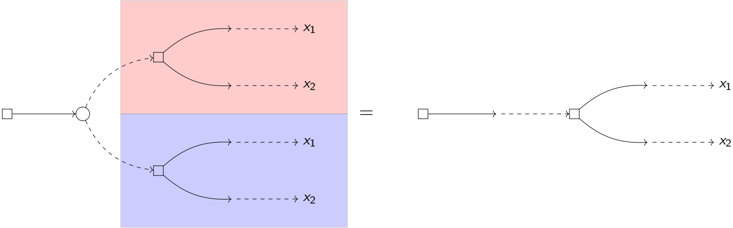
$$\{\{\{A, B\}, \{C, D\}\}, \{\{E\}\}\} \mapsto \{A \cup C, A \cup D, B \cup C, B \cup D, E\}$$

$$\Upsilon \in PPPP(X) \mapsto \{\bigcup \mathrm{Im}(f) \mid \exists \upsilon \in \Upsilon,\ f : \upsilon \xrightarrow{*} PP(X)\}$$

where $f : \upsilon \xrightarrow{*} PP(X)$ is a *choice function*: $\forall \mathcal{U} \in \upsilon : \mathcal{U} \in f(\mathcal{U})$.

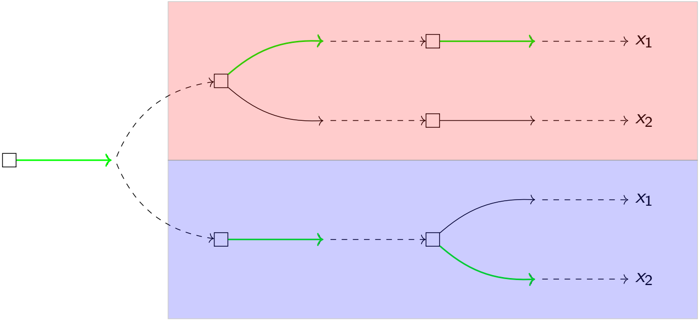
Failure of associativity

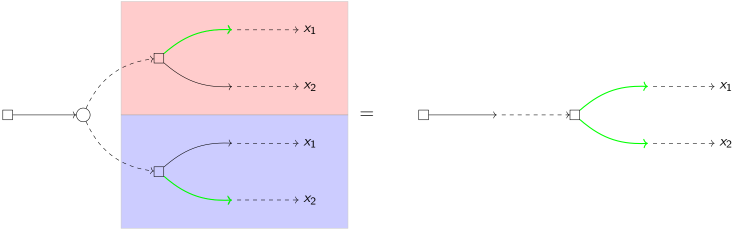




A monad structure on PP is a "no-go" [KS18]

Failure of associativity





(Weak) Distributive laws

Two solutions:

- Use multiplicities for the environment

(Weak) Distributive laws

Two solutions:

- Use multiplicities for the environment
- Modify our strategy picking procedure to include "convex choices"

This weakening of distributive laws were popularised by [Gar20], for discussion of convexity see [BS22]

(Weak) Distributive laws

Two solutions:

- Use multiplicities for the environment
- Modify our strategy picking procedure to include "convex choices"
- Both of these can be phrased in terms of monad *distributive laws*

This weakening of distributive laws were popularised by [Gar20], for discussion of convexity see [BS22]

(Weak) Distributive laws

Two solutions:

- Use multiplicities for the environment
- Modify our strategy picking procedure to include "convex choices"
- Both of these can be phrased in terms of monad *distributive laws*
- Given two monads (S, μ^T, η^T) and (T, μ^T, η^T), a *distributive law* $\delta : TS \to ST$ is a natural transformation satisfying some coherence conditions involving $\mu^T, \mu^S, \eta^T, \eta^S$.

This weakening of distributive laws were popularised by [Gar20], for discussion of convexity see [BS22]

(Weak) Distributive laws

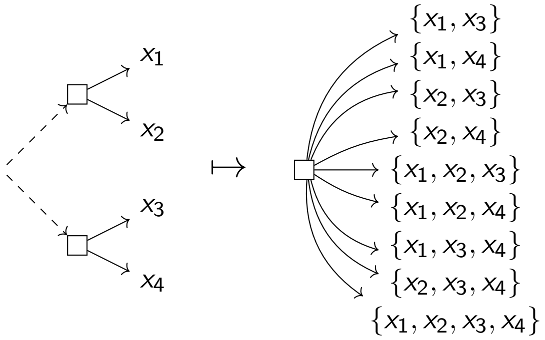
Two solutions:

- Use multiplicities for the environment
- Modify our strategy picking procedure to include "convex choices"
- Both of these can be phrased in terms of monad *distributive laws*
- Given two monads (S, μ^T, η^T) and (T, μ^T, η^T), a *distributive law* $\delta : TS \to ST$ is a natural transformation satisfying some coherence conditions involving $\mu^T, \mu^S, \eta^T, \eta^S$.
- A *weak distributive law* $\delta : TS \to ST$ only satisfies the diagrams involving μ^T, μ^S, η^S.

This weakening of distributive laws were popularised by [Gar20], for discussion of convexity see [BS22]

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

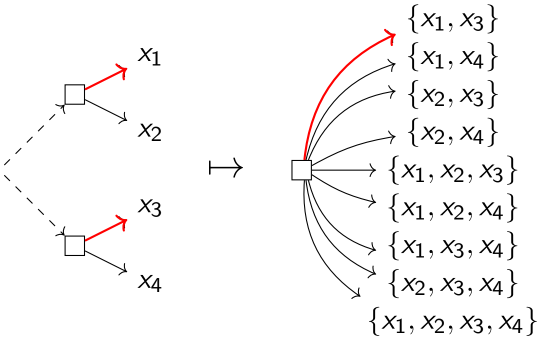


$$\delta(\{U_i\}_{i \in I}) = \{\bigcup_{i \in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

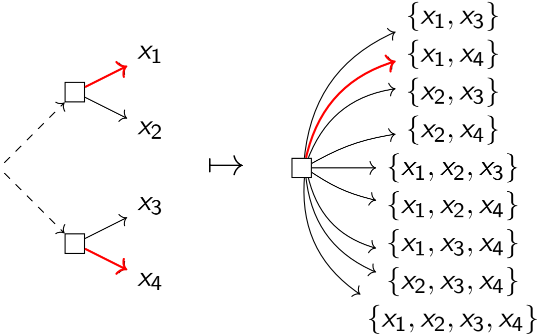


$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

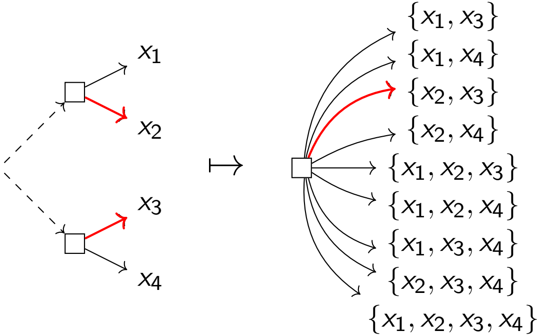


$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

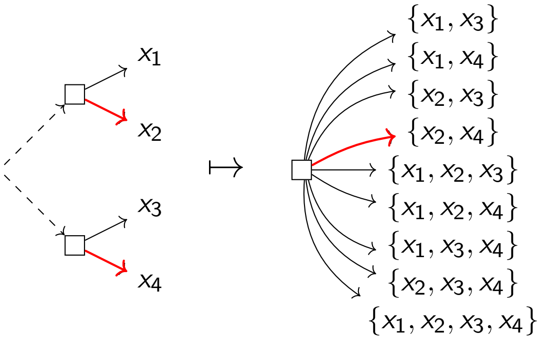


$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

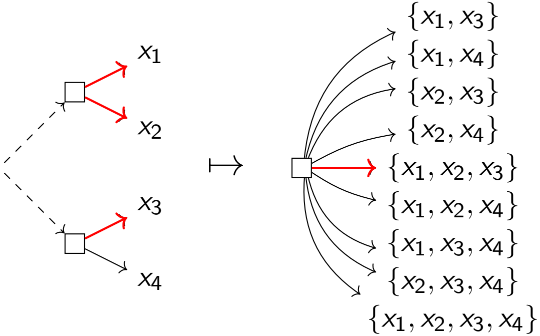


$$\delta(\{U_i\}_{i \in I}) = \{\bigcup_{i \in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

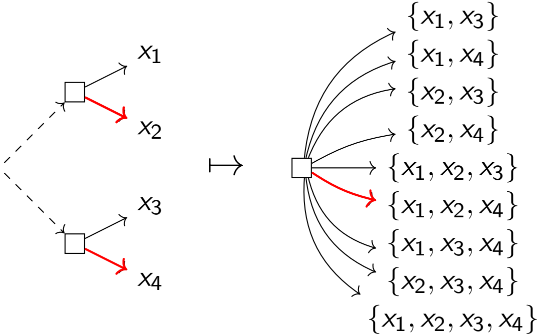


$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

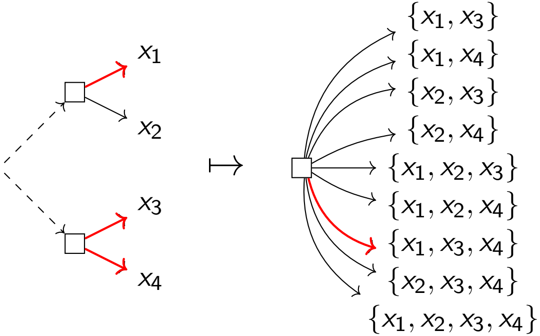


$$\delta(\{U_i\}_{i \in I}) = \{\bigcup_{i \in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

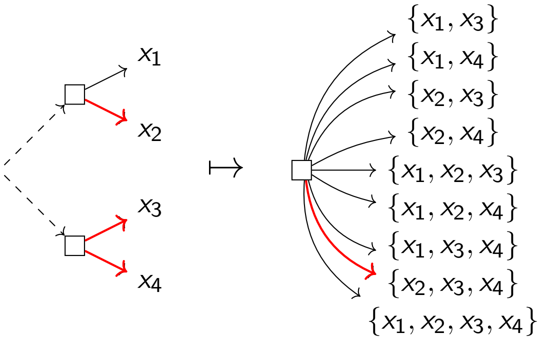


$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.

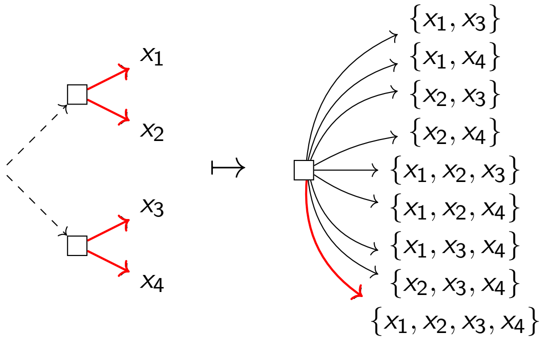


$$\delta(\{U_i\}_{i \in I}) = \{\bigcup_{i \in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

A weak distributive law $\delta : PP \to PP$

- Gives us a way of swapping environment-then-controller branching into controller-then-environment.



$$\delta(\{U_i\}_{i\in I}) = \{\bigcup_{i\in I} V_i \mid V_i \subseteq^+ U_i \text{ for all } i \in I\}$$

This law is used in [Goy21, Chapter 4] to model alternating automata

Trace semantics

- We can build a monad

$$\widetilde{PP}(X) = \{\mathcal{U} \subseteq X \mid \mathcal{U} \text{ is closed under arbitrary union}\}$$

$$\eta(x) = \{\{x\}\} \qquad \mu \text{ uses } \delta$$

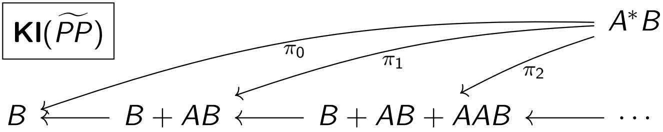
Trace semantics

- We can build a monad

$$\widetilde{PP}(X) = \{\mathcal{U} \subseteq X \mid \mathcal{U} \text{ is closed under arbitrary union}\}$$

$$\eta(x) = \{\{x\}\} \qquad \mu \text{ uses } \delta$$

- Recall: General categorical machinery allows us to lift this chain to the category of relations, and reverse the arrows:



with various assumptions on $\widetilde{PP}$

Refining the monad

-
-
-
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
-
-
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
- Composition in the Kleisli category is not left-strict.
-
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
- Composition in the Kleisli category is not left-strict.
- The monad is not commutative.
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
- The monad is not commutative.
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
  - Restrict the inner powerset to non-empty.
- The monad is not commutative.
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
  - Restrict the inner powerset to non-empty.
- The monad is not commutative.
  - Only consider linear functors (rather than polynomial)
-

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
  - Restrict the inner powerset to non-empty.
- The monad is not commutative.
  - Only consider linear functors (rather than polynomial)
- Let Q be the finite non-empty powerset monad.

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
  - Restrict the inner powerset to non-empty.
- The monad is not commutative.
  - Only consider linear functors (rather than polynomial)
- Let Q be the finite non-empty powerset monad.

$$Q(X) = \{U \subseteq_{\omega}^{+} X\}$$

These fixes are discussed in [PC25] (to appear)

Refining the monad

- The Kleisli category is not ω-cpo enriched.
  - Restrict the inner powerset to finite.
- Composition in the Kleisli category is not left-strict.
  - Restrict the inner powerset to non-empty.
- The monad is not commutative.
  - Only consider linear functors (rather than polynomial)
- Let Q be the finite non-empty powerset monad.

$$Q(X) = \{U \subseteq^{+}_{\omega} X\}$$

$$\widetilde{PQ}(X) = \{\mathcal{U} \subseteq Q(X) \mid \mathcal{U} \text{ is closed under binary union}\}$$

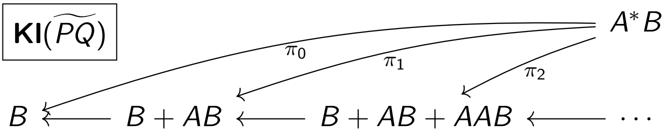
$$\delta : QP \to PQ$$

$$\delta(\{U_1, \ldots, U_n\}) := \{V_1 \cup \cdots \cup V_n \mid V_i \subseteq^{+}_{\omega} U\}$$

These fixes are discussed in [PC25] (to appear)

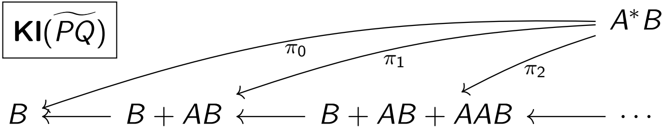
Traces and Executions

A^*B is the final $B + A(-)$-coalgebra in $\mathbf{Kl}(\widetilde{PQ})$.

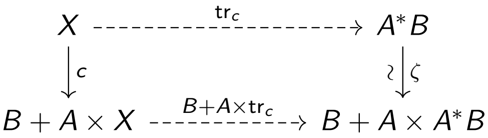


Traces and Executions

A^*B is the final $B + A(-)$-coalgebra in $\mathbf{Kl}(\widetilde{PQ})$.



Thus we have trace and execution maps by coinduction:



$$\mathrm{tr}_c : X \to \widetilde{PQ}(A^*B)$$

Traces and Executions

A^*B is the final $B + A(-)$-coalgebra in $\mathbf{Kl}(\widetilde{PQ})$.

$\mathbf{Kl}(\widetilde{PQ})$

$$B \leftarrow B + AB \leftarrow B + AB + AAB \leftarrow \cdots$$

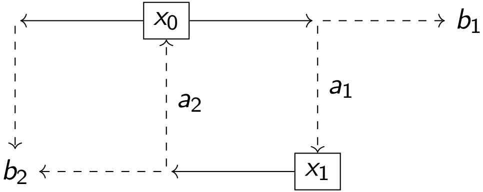
$$\pi_0 : A^*B \to B, \quad \pi_1 : A^*B \to B + AB, \quad \pi_2 : A^*B \to B + AB + AAB$$

Thus we have trace and execution maps by coinduction:

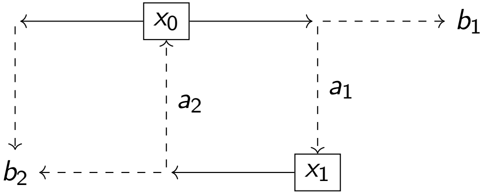
$$\begin{array}{ccc}
X & \xrightarrow{\ \mathrm{exec}_c\ } & (XA)^*XB \\
\downarrow{\scriptstyle c^*} & & \wr\downarrow{\scriptstyle \zeta} \\
X \times (B + A \times X) & \xrightarrow{\ B+A\times\mathrm{exec}_c\ } & X \times (B + A \times (XA)^*XB)
\end{array}$$

$$\mathrm{exec}_c : X \to \widetilde{PQ}((XA)^*XB)$$

What is $\mathrm{tr}_c(x_0)$?

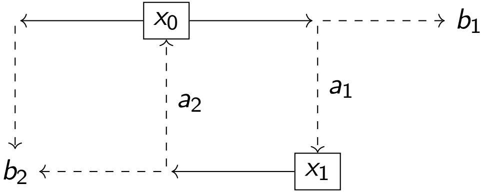


What is $\mathrm{tr}_c(x_0)$?



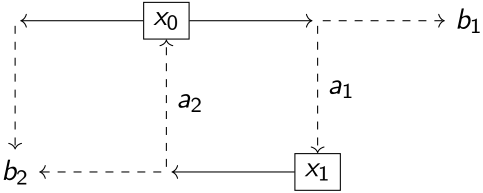
$$\{b_1\}$$

What is $\mathrm{tr}_c(x_0)$?



$\{b_1\}$ No

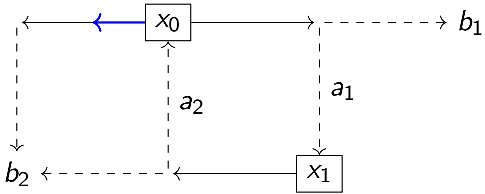
What is $\mathrm{tr}_c(x_0)$?



$\{b_1\}$ No

$\{b_2\}$

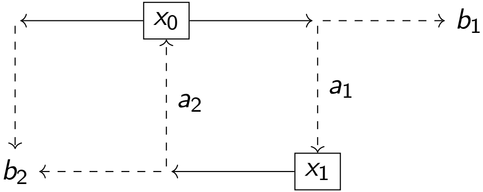
What is $tr_c(x_0)$?



$\{b_1\}$ No

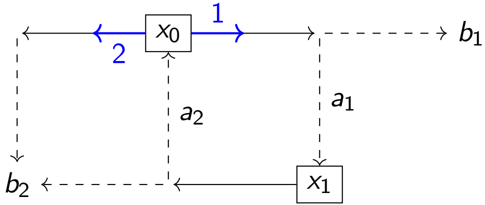
$\{b_2\}$ Yes

What is $\mathrm{tr}_c(x_0)$?



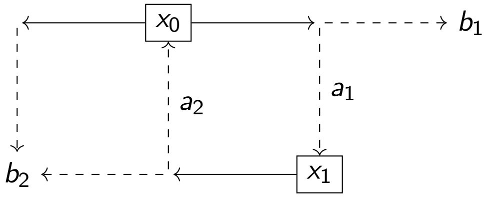
$\{b_1\}$	No
$\{b_2\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1\}$	

What is $\mathrm{tr}_c(x_0)$?



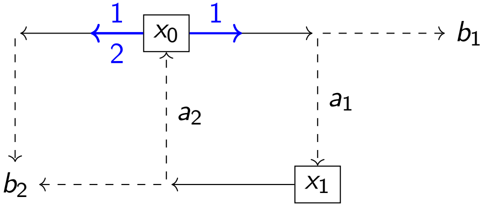
$\{b_1\}$	No
$\{b_2\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1\}$	Yes

What is $\mathrm{tr}_c(x_0)$?



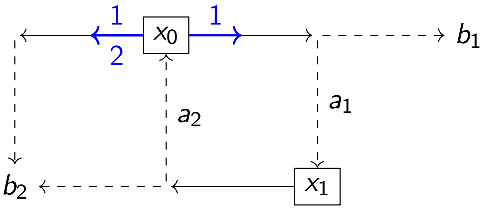
$\{b_1\}$	No
$\{b_2\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1, b_2\}$	

What is $\mathrm{tr}_c(x_0)$?



$\{b_1\}$	No
$\{b_2\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1, b_2\}$	Yes

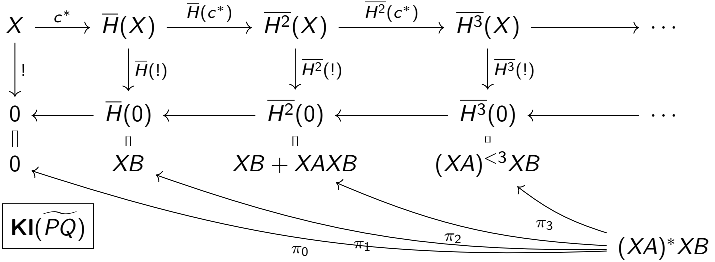
What is $\mathrm{tr}_c(x_0)$?



$\{b_1\}$	No
$\{b_2\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1\}$	Yes
$\{a_1 a_2 b_2, a_1 b_2, b_1, b_2\}$	Yes

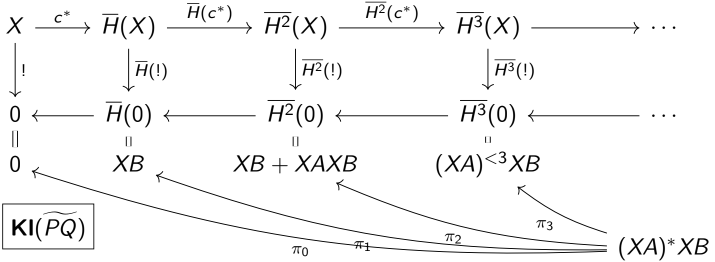
(Theorem sketch) for all $U \subseteq A^* B$:
$U \in \mathrm{tr}_c(x) \implies$ there is a strategy which enforces U
$U \in \mathrm{tr}_c(x) \impliedby^*$ there is a strategy which enforces U
*almost

Executions on the final sequence



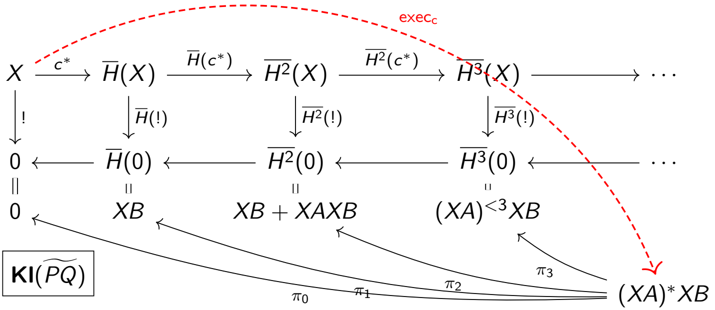
where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

Executions on the final sequence



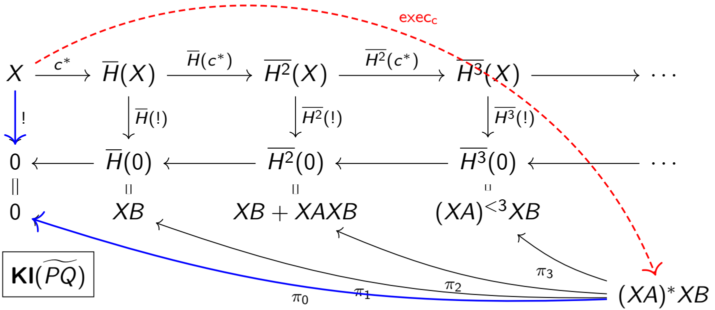
where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

Executions on the final sequence



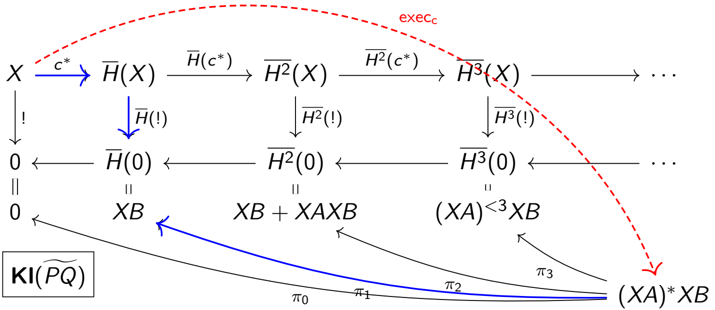
where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

Executions on the final sequence



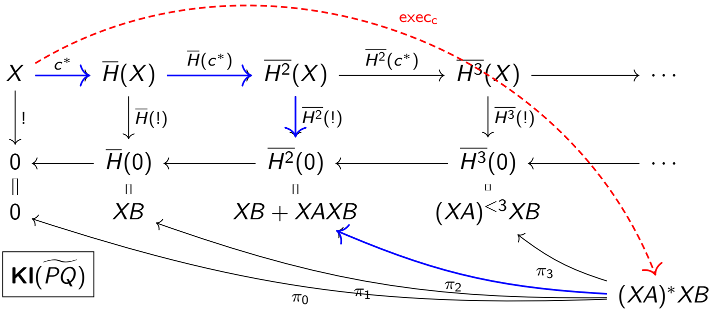
where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

Executions on the final sequence



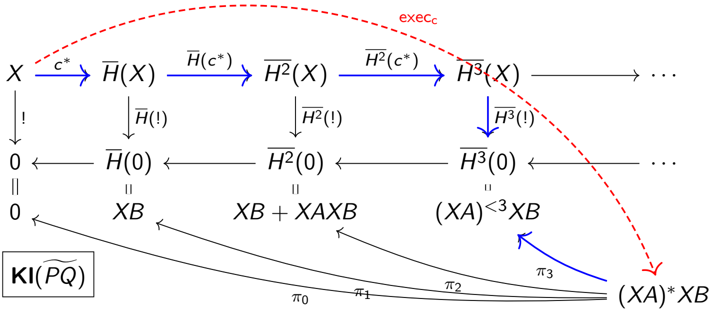
where $\overline{H} : \mathbf{Kl}(\widetilde{\mathcal{P}\mathcal{Q}}) \to \mathbf{Kl}(\widetilde{\mathcal{P}\mathcal{Q}})$ is the lifting of $X \times (B + A \times (-))$

Executions on the final sequence



where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

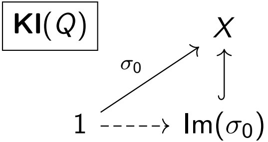
Executions on the final sequence



where $\overline{H} : \mathbf{Kl}(\widetilde{PQ}) \to \mathbf{Kl}(\widetilde{PQ})$ is the lifting of $X \times (B + A \times (-))$

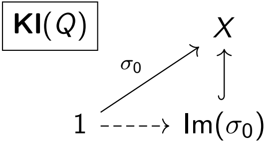
Where are the strategies?

- $\sigma_0 : 1 \to Q(X)$ will pick an initial state



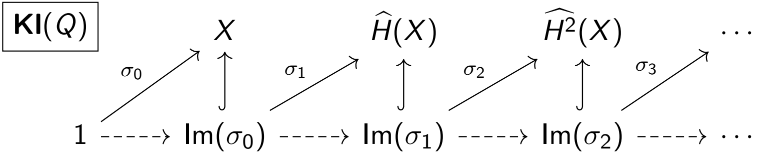
Where are the strategies?

- $\sigma_0 : 1 \to Q(X)$ will pick an initial state
- $\sigma_{n+1} : \mathrm{Im}(\sigma_n) \to QH^{n+1}(X)$ extends an n-length play



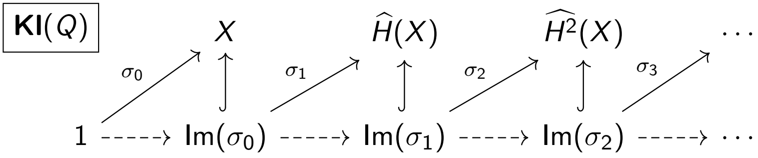
Where are the strategies?

- $\sigma_0 : 1 \to Q(X)$ will pick an initial state
- $\sigma_{n+1} : \mathrm{Im}(\sigma_n) \to QH^{n+1}(X)$ extends an *n*-length play

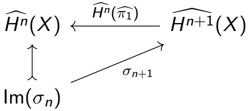


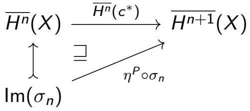
Where are the strategies?

- $\sigma_0 : 1 \to Q(X)$ will pick an initial state
- $\sigma_{n+1} : \mathrm{Im}(\sigma_n) \to QH^{n+1}(X)$ extends an n-length play



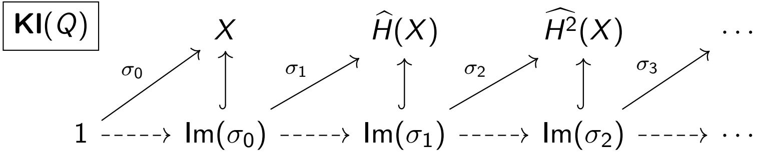
(left) σ extends partial plays, (right) we choose a successor in c:



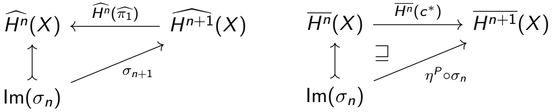


Where are the strategies?

- $\sigma_0 : 1 \to Q(X)$ will pick an initial state
- $\sigma_{n+1} : \mathrm{Im}(\sigma_n) \to QH^{n+1}(X)$ extends an n-length play



(left) σ extends partial plays, (right) we choose a successor in c:



The n-depth plays comes from composition:

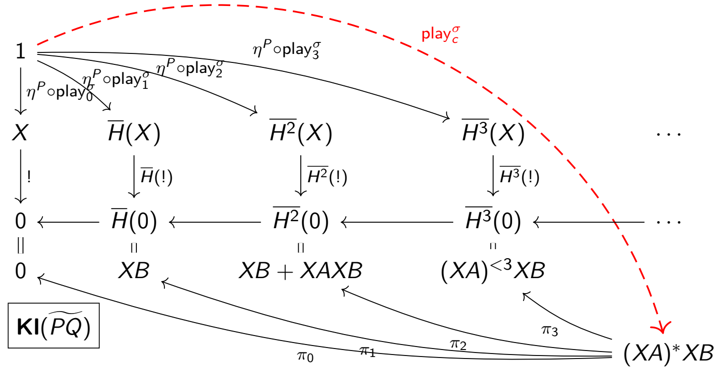
$$\mathrm{play}_n^{\sigma} = (1 \dashrightarrow \mathrm{Im}(\sigma_0) \dashrightarrow \cdots \dashrightarrow \mathrm{Im}(\sigma_n) \rightarrowtail H^n(X))$$

Play outcomes

To define the play outcome, first lift a strategy into $\mathbf{Kl}(\widetilde{PQ})$

$$\eta^P \circ \sigma_n : \mathrm{Im}(\sigma_n) \to \widetilde{PQ} H^{n+1}(X)$$

Then we can reuse that $(XA)^*XB$ is the limit of the final sequence:



Main theorem

Theorem

$$\mathrm{exec}_c(x) = \bigcup_{\sigma \text{ starts in } x} \mathrm{play}_c^\sigma$$

Lemma

$$c_n^*(x) = \{\mathrm{play}_n^\sigma \mid \sigma \text{ starts in } x\}$$

- What do we gain from doing this coalgebraically?

Main theorem

Theorem

$$\mathrm{exec}_c(x) = \bigcup_{\sigma \text{ starts in } x} \mathrm{play}_c^\sigma$$

Lemma

$$c_n^*(x) = \{\mathrm{play}_n^\sigma \mid \sigma \text{ starts in } x\}$$

- What do we gain from doing this coalgebraically?

Main theorem

Theorem

$$\mathrm{exec}_c(x) = \bigcup_{\sigma \text{ starts in } x} \mathrm{play}_c^\sigma$$

Lemma

$$c_n^*(x) = \{\mathrm{play}_n^\sigma \mid \sigma \text{ starts in } x\}$$

- What do we gain from doing this coalgebraically?
  - Replace Q with the finite distribution monad D!

Main theorem

Theorem

$$\mathrm{exec}_c(x) = \bigcup_{\sigma \text{ starts in } x} \mathrm{play}_c^\sigma$$

Lemma

$$c_n^*(x) = \{\mathrm{play}_n^\sigma \mid \sigma \text{ starts in } x\}$$

- What do we gain from doing this coalgebraically?
  - Replace Q with the finite distribution monad D!
  - Generic coinductive algorithms for strategy synthesis.

Conclusion

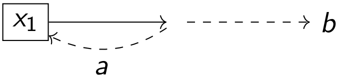
- Towards strategy synthesis...
  - Product construction?
  - General theorem about memoryless strategies?
  - Infinite traces, continuous probability monads?
- An axiomatic presentation.
- Simple stochastic games?

Bibliography I

- Filippo Bonchi and Alessio Santamaria, *Convexity via Weak Distributive Laws*, Logical Methods in Computer Science **Volume 18, Issue 4** (2022), 8389, arXiv:2108.10718 [cs, math].
- Richard Garner, *The Vietoris Monad and Weak Distributive Laws*, Applied Categorical Structures **28** (2020), no. 2, 339–354 (en).
- Alexandre Goy, *On the compositionality of monads via weak distributive laws*, phdthesis, Université Paris-Saclay, October 2021.
- Ichiro Hasuo, Bart Jacobs, and Ana Sokolova, *Generic trace semantics via coinduction*, Logical Methods in Computer Science **Volume 3, Issue 4** (2007).

Bibliography II

- Bartek Klin and Julian Salamanca, *Iterated covariant powerset is not a monad*, Electronic Notes in Theoretical Computer Science **341** (2018), 261–276, Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXIV).
- Benjamin Plummer and Cîrstea Corina, *Traces via strategies in two-player games*, Unpublished manuscript, 2025, Submitted to Mathematical Foundations of Programming Semantics, under review.
- M. B. Smyth and G. D. Plotkin, *The category-theoretic solution of recursive domain equations*, SIAM Journal on Computing **11** (1982), no. 4, 761–783.



x1
b
a