Universal RL

Sridhar Mahadevan

Adobe Research and U.Mass

www.people.cs.umass.edu/~mahadeva

Universal Reinforcement Learning

We now generalize reinforcement learning to Universal Reinforcement Learning (URL) by embedding MDPs into a broader class of dynamical systems defined by universal coalgebras.

A coalgebra for a functor $F : \mathcal{C} \to \mathcal{C}$ is a pair (X, α) where

$$\alpha : X \to F(X),$$

with X the carrier object and F defining the system dynamics.
Coalgebras provide a unifying representation of dynamical systems, including:

- finite automata,
- grammars and Turing machines,
- stochastic processes such as Markov chains and MDPs,
- and more general probabilistic systems.

What is Coinduction?

In category theory, induction is associated with *initial algebras* and well-founded constructions.

Coinduction is associated with *final coalgebras* and potentially non-well-founded structures. While induction builds finite objects by recursion, coinduction specifies possibly infinite or self-referential objects through their unfolding behavior.

> *Induction constructs an object from its generators; coinduction characterizes an object by the observations it supports.*

This makes coinduction especially well suited to dynamical systems, infinite processes, and sequential decision problems.

Finite State Machine as a Coalgebra

A deterministic finite-state machine may be specified as a labeled transition system

$$(S, \to_S, A),$$

where S is a set of states, A is a set of input labels, and

$$\to_S \subseteq S \times A \times S$$

is the transition relation.
Define the endofunctor

$$\mathcal{B}(X) = \mathcal{P}(A \times X).$$

Then the transition system can be represented as a coalgebra

$$\alpha_S : S \to \mathcal{B}(S), \qquad s \mapsto \{(a, s') \mid s \xrightarrow{a} s'\}.$$

This example illustrates the key idea: the coalgebra records the observable one-step behavior of the system.

Induction vs. Coinduction

The classical problem of asynchronous distributed minimization concerns the computation of a fixed point

$$F(x^*) = x^*, \quad x^* \in X,$$

where $X = X_1 \times \cdots \times X_n$ is a product space, and the mapping F decomposes into component functions

$$F(x) = (f_1(x), \ldots, f_n(x)).$$

Each component x_i is updated iteratively according to

$$x_i \leftarrow f_i(x),$$

possibly using stale or delayed information from other components.
This procedure admits a natural interpretation as an *inductive* construction: starting from an initial estimate $x(0)$, successive iterations refine the solution toward a fixed point.

Coinduction

The convergence theory of asynchronous computation is fundamentally inductive. It relies on constructing a sequence of nested sets

$$\cdots \subset X(k+1) \subset X(k) \subset \cdots \subset X$$

and proving, via mathematical induction, that the iterates remain within these sets and converge to a fixed point.
In contrast, the URL framework replaces this inductive viewpoint with a *coinductive* one, in which solutions are characterized as fixed points of behavioral consistency conditions rather than constructed step-by-step from initial approximations.

RL and Coinduction

The relevance of coinduction to reinforcement learning can now be stated succinctly. In RL, the value of a state is not defined directly by finite construction, but by a consistency condition relating it to future states. This is the essence of the Bellman equation.

From the categorical perspective developed in this course:

- local reward and transition structure define a functor,
- global value semantics arise as a right Kan extension,
- and coinduction characterizes the fixed point of this extension process.

  Coinduction is the semantic principle underlying right Kan extensions in sequential decision systems.

Thus, coinduction is not merely an abstract alternative to induction: it is the correct language for value functions, policies, and long-horizon behavior.

Asynchronous Distributed Minimization as a Right Kan Extension

From the perspective of Universal Decision Models, the fixed point equation

$$F(x) = x$$

can be interpreted as a consistency condition defining a global object from local update rules.

In particular:

- each component update f_i encodes a local constraint,
- the fixed point x^* is a globally consistent solution,
- and the asynchronous iteration approximates a limit over these constraints.

This aligns naturally with the categorical notion of a right Kan extension:

$$x^* \simeq \mathrm{Ran}_J F,$$

where F encodes local update structure and J embeds local contexts into the global system.

Asynchronous distributed minimization can be viewed as an iterative approximation to a right Kan extension.

Stochastic Approximation

Reinforcement learning algorithms introduce stochasticity into this framework. A typical update takes the form

$$x_i(t+1) = x_i(t) + \alpha_i(t)\{F_i(x(t)) - x_i(t) + w_i(t)\}, \tag{1}$$

where $w_i(t)$ represents noise and $\alpha_i(t)$ is a step size.
This can be interpreted as a stochastic approximation to the fixed point equation.
From the Kan perspective:

- the deterministic mapping F defines a right Kan semantics,
- stochastic approximation provides a noisy iterative method for approximating this semantics,
- convergence corresponds to recovering the fixed point of the Kan extension.

Metric Coinduction

Let (V, d) be a metric space, and let

$$F : V \to V$$

be a contractive map, meaning that for some $0 \leq c < 1$,

$$d(F(u), F(v)) \leq c\, d(u, v) \qquad \forall u, v \in V.$$

More generally, F is *eventually contractive* if F^n is contractive for some $n \geq 1$.
The standard fixed-point iteration

$$u,\ F(u),\ F^2(u),\ \ldots$$

then converges to a unique fixed point u^*, since the resulting sequence is Cauchy.

Definition (Coinduction Rule)

Let ϕ be a closed nonempty subset of a metric space V, and let $F : V \to V$ be an eventually contractive map that preserves ϕ. Then the unique fixed point u^* of F lies in ϕ.

Metric Coinduction

This rule is coinductive because it proves a property of the fixed point not by constructing the point explicitly, but by showing that the property is preserved by the unfolding dynamics. The fixed point belongs to every closed invariant set preserved by the operator.
This is precisely the form of reasoning needed in RL:

- the Bellman operator preserves a class of admissible value functions,
- its unique fixed point is therefore characterized coinductively,
- and convergence follows from the metric structure.

To make the coalgebraic connection explicit, one organizes contractive systems into a category of coalgebras in which the fixed point corresponds to a final object. The coinduction rule then states that any property preserved by the coalgebra structure must hold of the final coalgebra.
In this way, metric coinduction provides the analytic counterpart to the categorical notion of finality.

RL Exact Solution Methods

Exact dynamic programming methods compute value functions by solving the Bellman equations exactly. In the language of the previous chapter, they compute the globally consistent value function determined by the local reward and transition structure.

One of the most familiar exact methods is value iteration, which computes successive approximations

$$V^{t+1} = T^*(V^t),$$

where

$$V^{t+1}(s) = T^*(V^t)(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s') V^t(s') \right). \tag{2}$$

Exact Solution Methods are Coalgebraic

The Bellman optimality equation

$$T^*(V) = V$$

defines a fixed point, and is therefore naturally coalgebraic. The value iteration algorithm computes this fixed point by successive approximation.

From the perspective of Universal Decision Models:

- the local reward and transition structure define a functor on local contexts,
- the optimal value function V^* is the corresponding right Kan extension,
- and value iteration is an iterative method for computing this extension.

Value iteration is a metric-coinductive procedure for computing the right Kan semantics of an MDP.

Approximate MDP Solution Methods

We now turn to approximate solution methods. These methods retain the exact representation of value functions, but replace full knowledge of the MDP by sampled transitions

$$(s_t, a_t, r_t, s_{t+1}).$$

In this sense, they are *simulation-based* procedures for approximating the same value semantics computed exactly by dynamic programming.

The conceptual shift is simple:

- exact methods compute the right Kan extension using the full model,
- simulation-based methods estimate it from samples.

Q-learning as Kan Extensions

A particularly important TD method is Q-learning, which estimates the optimal action-value function:

$$Q_t(s,a) \leftarrow (1-\alpha_t)Q_t(s,a) + \alpha_t \left(r_t + \gamma \max_{a'} Q_t(s',a') \right). \tag{3}$$

Equivalently,

$$Q_t(s,a) \leftarrow Q_t(s,a) + \alpha_t \left[r_t + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a) \right].$$

Categorically:

- the maximization over actions corresponds to a colimit-like local aggregation,
- the Bellman target enforces right Kan consistency,
- and the update rule performs stochastic coinduction toward the optimal semantic fixed point.

Thus, Q-learning approximates optimal right Kan extension in the category $\mathcal{C}_Q$.

Category of Coalgebras

Definition

Let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor. A *homomorphism* of F-coalgebras (A, α) and (B, β) is an arrow $f : A \to B$ in the category $\mathcal{C}$ such that the following diagram commutes:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \beta} \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

Lambek's Lemma

An F-system (P, π) is termed **final** if for another F-system (S, α_S), there exists a unique homomorphism $f_S : (S, \alpha_S) \to (P, \pi)$. That is, (P, π) is the terminal object in the category of coalgebras Set_F defined by some set-valued endofunctor F. Since the terminal object in a category is unique up to isomorphism, any two final systems must be isomorphic.

Definition

An F-coalgebra (A, α) is a *fixed point* for F, written as $A \simeq F(A)$ if α is an isomorphism between A and $F(A)$. That is, not only does there exist an arrow $A \to F(A)$ by virtue of the coalgebra α, but there also exists its inverse $\alpha^{-1} : F(A) \to A$ such that

$$\alpha \circ \alpha^{-1} = \mathbf{id}_{F(A)} \quad \text{and} \quad \alpha^{-1} \circ \alpha = \mathbf{id}_A$$

Lambek's Lemma

Theorem

Lambek: *A final F-coalgebra is a fixed point of the endofunctor F.*

Proof: The proof is worth understanding, as it provides a classic example of the power of diagram chasing. Let (A, α) be a final F-coalgebra. Since $(F(A), F(\alpha)$ is also an F-coalgebra, there exists a unique morphism $f : F(A) \to A$ such that the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{f} & A \\ \downarrow{\scriptstyle F(\alpha)} & & \downarrow{\scriptstyle \alpha} \\ F(F(A)) & \xrightarrow{F(f)} & F(A) \end{array}$$

However, by the property of finality, the only arrow from (A, α) into itself is the identity.

Lambek's Lemma

We know the following diagram also commutes, by virtue of the definition of coalgebra homomorphism:

$$\begin{array}{ccc} A & \xrightarrow{\alpha} & F(A) \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \alpha} \\ F(A) & \xrightarrow{F(\alpha)} & F(F(A)) \end{array}$$

Combining the above two diagrams, it clearly follows that $f \circ \alpha$ is the identity on object A, and it also follows that $F(\alpha) \circ F(f)$ is the identity on $F(A)$. Therefore, it follows that:

$$\alpha \circ f = F(f) \circ F(\alpha) = F(f \circ \alpha) = F(\mathbf{id}_A) = \mathbf{id}_{F(A)} \quad \square$$

Diagrams in URL

A central theme in URL is the use of diagrams

$$F : \mathcal{J} \to \mathcal{C},$$

where $\mathcal{J}$ is an indexing category and $\mathcal{C}$ is a category of decision or state objects.
In the asynchronous setting:

- each node in $\mathcal{J}$ corresponds to a component x_i,
- arrows represent dependencies between components,
- the global system corresponds to a limit of the diagram.

More complex RL architectures correspond to richer diagrams:

- table-based methods correspond to discrete diagrams,
- neural networks correspond to compositional diagrams,
- distributed systems correspond to partially ordered diagrams of dependencies.