CertRL : **Formalizing Convergence Proofs of Value and Policy Iteration in Coq**

Koundinya Vajjha
University of Pittsburgh

Avi Shinnar
IBM Research

Barry Trager
IBM Research

Vasily Pestun
IBM Research ; IHES

Nathan Fulton
IBM Research

Introduction - Reinforcement Learning

Reinforcement learning (RL) algorithms solve sequential decision making problems in which the goal is to choose actions that maximize a quantitative utility function.

Introduction - Reinforcement Learning

Reinforcement learning (RL) algorithms solve sequential decision making problems in which the goal is to choose actions that maximize a quantitative utility function. Recent high-profile applications of reinforcement learning include:

- beating the world's best players at Go
- competing against top professionals in Dota
- improving protein structure prediction
- automatically controlling complex robots

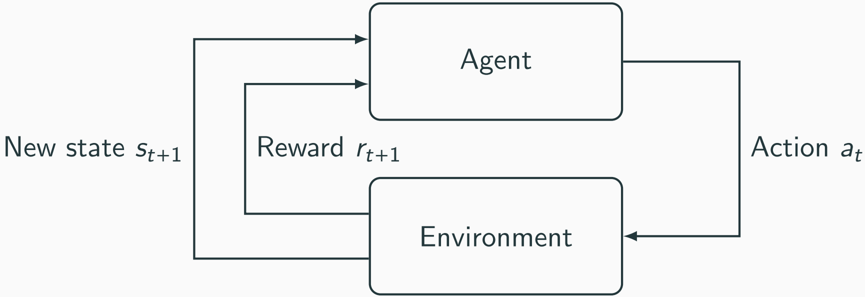
Introduction - Reinforcement Learning

Reinforcement learning (RL) algorithms solve sequential decision making problems in which the goal is to choose actions that maximize a quantitative utility function. Recent high-profile applications of reinforcement learning include:

- beating the world's best players at Go
- competing against top professionals in Dota
- improving protein structure prediction
- automatically controlling complex robots

These successes motivate the use of reinforcement learning in safety-critical and correctness-critical settings.

Reinforcement Learning Theory



Reinforcement Learning Theory

This generates a *trajectory* of states, actions and (expected) rewards.

$$
\begin{aligned}
& s_0 \;\; a_0 \;\; s_1 \;\; a_1 \;\; s_2 \;\; a_2 \;\; \ldots && (1)\\
& \qquad\quad\; r_1 \qquad\quad r_2 \qquad\quad \ldots && (2)
\end{aligned}
$$

Reinforcement Learning Theory

This generates a *trajectory* of states, actions and (expected) rewards.

$$
\begin{aligned}
&s_0 \ a_0 \ s_1 \ a_1 \ s_2 \ a_2 \ \dots && (1) \\
&\qquad\ \ r_1 \qquad r_2 \qquad \dots && (2)
\end{aligned}
$$

A *policy* π is a map from *states* to *actions*.

This generates a *trajectory* of states, actions and (expected) rewards.

$$
\begin{aligned}
s_0 \; a_0 \; s_1 \; a_1 \; s_2 \; a_2 \; \ldots \qquad (1) \\
r_1 \qquad r_2 \qquad \ldots \qquad (2)
\end{aligned}
$$

A *policy* π is a map from *states* to *actions*. The *long-term value* of π is defined as the discounted sum of rewards. For $0 \leq \gamma < 1$:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 \ldots \qquad (3)$$

Reinforcement Learning Theory

This generates a *trajectory* of states, actions and (expected) rewards.

$$s_0 \; a_0 \; s_1 \; a_1 \; s_2 \; a_2 \; \ldots \tag{1}$$

$$r_1 \qquad r_2 \qquad \ldots \tag{2}$$

A *policy* π is a map from *states* to *actions*. The *long-term value* of π is defined as the discounted sum of rewards. For $0 \leq \gamma < 1$:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 \ldots \tag{3}$$

Goal: Find a policy mapping states to actions that maximizes the agent's long-term reward.

Example

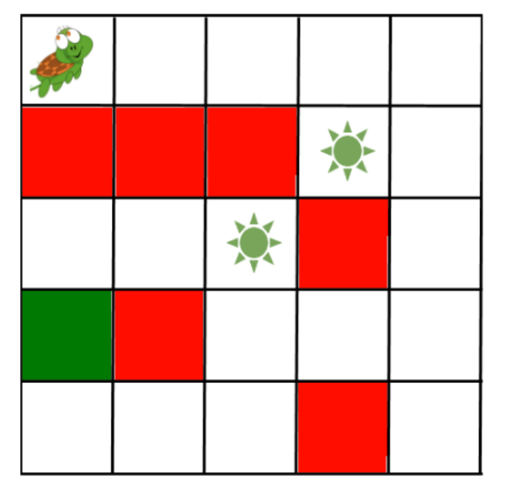


Figure 1: Turtle in a Gridworld ✿

Introduction - Reinforcement Learning

Reinforcement learning algorithms produce, at a minimum, a *policy* that specifies which action(s) should be taken in a given state.

Introduction - Reinforcement Learning

Reinforcement learning algorithms produce, at a minimum, a *policy* that specifies which action(s) should be taken in a given state.

The primary correctness property for reinforcement learning algorithms is *convergence*: in the limit, a reinforcement learning algorithm should converge to a policy that optimizes for the long-term value.

CertRL contains a Coq formalization of *Value iteration* and *Policy iteration*. [1]

- These are canonical RL algorithms, often taught as the first reinforcement learning methods in machine learning courses.
- Their convergence proofs contain the main ingredients of a typical convergence argument for an RL algorithm.
- Their convergence is usually assumed implicitly in implementations.
- These algorithms are at the core of the *dynamic programming* paradigm.

[1] Formalization is available at https://github.com/IBM/FormalML

Markov Decision Processes ✿

```
Record MDP := mkMDP {
    (** State and action spaces. *)
    state : Type;
    act : forall s: state, Type;

    (** The state and action spaces are finite. *)
    fs :> Finite (state) ;
    fa :> forall s, Finite (act s);

    (** The state space and the fibered action spaces are
        nonempty. *)
    ne : NonEmpty (state) ;
    na : forall s, NonEmpty (act s);
```

Markov Decision Processes – continued

```
    (** Probabilistic transition structure. t(s,a,s') is the
        probability that the next state is s’
        given that you take action a in state s.
        One can also consider it to be an act-indexed
        collection of Kleisli arrows of Pmf. *)
    t : forall s : state, (act s -> Pmf state);

    (** Reward when you move to s' from s by taking action a. *)
    reward : forall s : state, (act s -> state -> R)
}
```

The set of time steps may be finite or infinite. That is, the MDP may be *finite-horizon* or *infinite-horizon*.

Markov Decision Processes - Policies

A *decision rule* ✿ is a mapping from states to actions.

```
Definition dec_rule (M : MDP) := forall s : M.(state), (M.(act)) s.
```

Markov Decision Processes - Policies

A *decision rule* ✿ is a mapping from states to actions.

```
Definition dec_rule (M : MDP) := forall s : M.(state), (M.(act)) s.
```

In the finite time horizon case, a policy is a *list of decision rules*.

Markov Decision Processes - Policies

A *decision rule* ✿ is a mapping from states to actions.

```
Definition dec_rule (M : MDP) := forall s : M.(state), (M.(act)) s.
```

In the finite time horizon case, a policy is a *list of decision rules*. In the infinite-time horizon case, a policy ✿ is a *stream of decision rules*.

```
Definition policy (M : MDP) := Stream (dec_rule M).
```

Markov Decision Processes - Policies

A *decision rule* ✿ is a mapping from states to actions.

```
Definition dec_rule (M : MDP) := forall s : M.(state), (M.(act)) s.
```

In the finite time horizon case, a policy is a *list of decision rules*. In the infinite-time horizon case, a policy ✿ is a *stream of decision rules*.

```
Definition policy (M : MDP) := Stream (dec_rule M).
```

A policy of the form:

$$\pi, \pi, \pi, \ldots$$

is called a *stationary policy*.

MDP Transition Structure

- The transitions of an MDP are *stochastic*. If the agent, at state s performs an action a, the next state can be specified only up to a probability.

MDP Transition Structure

- The transitions of an MDP are *stochastic*. If the agent, at state s performs an action a, the next state can be specified only up to a probability.
- However, we are interested in the behaviour of the agent n-steps from the present. For this, we need to *compose* probabilities.

MDP Transition Structure

- The transitions of an MDP are *stochastic*. If the agent, at state s performs an action a, the next state can be specified only up to a probability.
- However, we are interested in the behaviour of the agent n-steps from the present. For this, we need to *compose* probabilities.
- Classically, this is done by a state-transition matrix whose powers are used to calculate the probability that the agent visits a particular state n-steps from now.

MDP Transition Structure

- The transitions of an MDP are *stochastic*. If the agent, at state s performs an action a, the next state can be specified only up to a probability.
- However, we are interested in the behaviour of the agent n-steps from the present. For this, we need to *compose* probabilities.
- Classically, this is done by a state-transition matrix whose powers are used to calculate the probability that the agent visits a particular state n-steps from now.

Is there an alternative to formalizing matrices and matrix multiplication?

Giry Monad

Key idea: Kleisli composition of the Giry monad recovers the Chapman-Kolmogorov formula.

Giry Monad

Key idea: Kleisli composition of the Giry monad recovers the Chapman-Kolmogorov formula.

We first define the type of discrete probability measures ✿ on a type as

```
Record Pmf (A : Type) := mkPmf {
  outcomes : list (nonnegreal * A);
  sum1 : list_fst_sum outcomes = R1
}.
```

Giry Monad

We then define two basic operations:

$$\begin{aligned} \mathrm{ret} : A &\to P(A) \text{ ✿} \\ a &\mapsto \lambda x : A,\ \delta_a(x) \end{aligned}$$

Giry Monad

We then define two basic operations:

$$\begin{aligned} \text{ret} : A &\to P(A) \text{ ✿} \\ a &\mapsto \lambda x : A,\ \delta_a(x) \end{aligned}$$

and

$$\begin{aligned} \text{bind} : P(A) &\to (A \to P(B)) \to P(B) \text{ ✿} \\ \text{bind } p\ f &= \lambda b : B,\ \sum_{a \in A} f(a)(b) * p(a) \end{aligned}$$

where $\delta_a(x) = 1$ if $a = x$ and 0 otherwise.

Giry Monad

We then define two basic operations:

$$\begin{aligned} \mathrm{ret} : A &\to P(A) \text{ ✿} \\ a &\mapsto \lambda x : A,\ \delta_a(x) \end{aligned}$$

and

$$\begin{aligned} \mathrm{bind} : P(A) \to (A \to P(B)) \to P(B) \text{ ✿} \\ \mathrm{bind}\ p\ f = \lambda b : B,\ \sum_{a \in A} f(a)(b) * p(a) \end{aligned}$$

where $\delta_a(x) = 1$ if $a = x$ and 0 otherwise. These operations satisfy the "monad laws" making $(P, \mathrm{bind}, \mathrm{ret})$ into a monad called the Giry monad.

Kleisli Composition

Given $f : A \to P(B)$ and $g : B \to P(C)$, Kleisli composition puts f and g together to give a map $(f \rightarrowtail\!\!\!\rightarrow g) : A \to P(C)$.

Kleisli Composition

Given $f : A \to P(B)$ and $g : B \to P(C)$, Kleisli composition puts f and g together to give a map $(f \rightarrowtail\!\!\!\!\rightarrow g) : A \to P(C)$. It is defined as:

$$
\begin{aligned}
f \rightarrowtail\!\!\!\!\rightarrow g &:= \lambda x : A, \text{bind } (f\ x)\ g && (4)\\
&= \lambda x : A, \left(\lambda c : C, \sum_{b:B} g(b)(c) * f(x)(b) \right) && (5)\\
&= \lambda (x : A)\ (c : C), \sum_{b:B} f(x)(b) * g(b)(c) && (6)
\end{aligned}
$$

Kleisli Composition

Given $f : A \to P(B)$ and $g : B \to P(C)$, Kleisli composition puts f and g together to give a map $(f \rightarrowtail g) : A \to P(C)$. It is defined as:

$$
\begin{aligned}
f \rightarrowtail g &:= \lambda x : A, \text{bind } (f\ x)\ g & (4)\\
&= \lambda x : A, \left(\lambda c : C, \sum_{b:B} g(b)(c) * f(x)(b) \right) & (5)\\
&= \lambda (x : A)\ (c : C), \sum_{b:B} f(x)(b) * g(b)(c) & (6)
\end{aligned}
$$

which is exactly the Chapman-Kolmogorov formula.

Kleisli Composition

Given $f : A \to P(B)$ and $g : B \to P(C)$, Kleisli composition puts f and g together to give a map $(f \rightarrowtail g) : A \to P(C)$. It is defined as:

$$
\begin{aligned}
f \rightarrowtail g &:= \lambda x : A, \mathrm{bind}\ (f\ x)\ g && (4)\\
&= \lambda x : A, \left(\lambda c : C, \sum_{b:B} g(b)(c) * f(x)(b) \right) && (5)\\
&= \lambda (x : A)\ (c : C), \sum_{b:B} f(x)(b) * g(b)(c) && (6)
\end{aligned}
$$

which is exactly the Chapman-Kolmogorov formula. So $(f \rightarrowtail g)\ x\ c$ gives the total probability of transitioning from x to c through an intermediate state in B.

Kleisli Iterates in an MDP

For a fixed decision rule π, we get a Kleisli arrow $T_\pi : S \to P(S)$ defined as $T_\pi(s) := T(s)(\pi(s))$.

Kleisli Iterates in an MDP

For a fixed decision rule π, we get a Kleisli arrow $T_\pi : S \to P(S)$ defined as $T_\pi(s) := T(s)(\pi(s))$. So, starting at an intial state s_0, if we follow the stationary policy induced by π, we recover the entire probability transition structure after k-steps as k-fold Kleisli iterates of T_π.

$$T_\pi^k(s_0) := (\mathrm{ret}\ s_0 \rightarrowtail\!\!\!\!\rightarrow \underbrace{T_\pi \rightarrowtail\!\!\!\!\rightarrow \ldots \rightarrowtail\!\!\!\!\rightarrow T_\pi}_{k\ \text{times}}) : P(S) \tag{7}$$

Kleisli Iterates in an MDP

For a fixed decision rule π, we get a Kleisli arrow $T_\pi : S \to P(S)$ defined as $T_\pi(s) := T(s)(\pi(s))$. So, starting at an intial state s_0, if we follow the stationary policy induced by π, we recover the entire probability transition structure after k-steps as k-fold Kleisli iterates of T_π.

$$T_\pi^k(s_0) := (\text{ret } s_0 \gg\!= \underbrace{T_\pi \gg\!= \ldots \gg\!= T_\pi}_{k \text{ times}}) : P(S) \tag{7}$$

Since this is a probability measure, we can find the expected reward at the k-th step as:

$$r_k^\pi(s) := \mathbb{E}_{T_\pi^k(s)}[r(s,\pi(s))] = \sum_{s' \in S} \left[r(s,\pi(s),s') T_\pi^k(s)(s') \right] ✿ \tag{8}$$

Long-Term Value

Let $\gamma \in \mathbb{R}, 0 \leq \gamma < 1$ be a *discount factor*, and $\pi = (\pi, \pi, \dots)$ be a stationary policy. Then, the long-term value $V_\pi : S \to \mathbb{R}$ is given by

$$V_\pi(s) = \sum_{k=0}^{\infty} \gamma^k r_k^\pi(s) \text{ ✿} \tag{9}$$

Long-Term Value

Let $\gamma \in \mathbb{R}, 0 \leq \gamma < 1$ be a *discount factor*, and $\pi = (\pi, \pi, \dots)$ be a stationary policy. Then, the long-term value $V_\pi : S \to \mathbb{R}$ is given by

$$V_\pi(s) = \sum_{k=0}^{\infty} \gamma^k r_k^\pi(s) \quad \text{✿} \tag{9}$$

We can prove that the long-term value satisfies the Bellman equation:

$$V_\pi(s) = r(s, \pi(s)) + \gamma \mathbb{E}_{T(s, \pi(s))}[V_\pi] \quad \text{✿} \tag{10}$$

Long-Term Value

Let $\gamma \in \mathbb{R}, 0 \le \gamma < 1$ be a *discount factor*, and $\pi = (\pi, \pi, \ldots)$ be a stationary policy. Then, the long-term value $V_\pi : S \to \mathbb{R}$ is given by

$$V_\pi(s) = \sum_{k=0}^{\infty} \gamma^k r_k^\pi(s) \quad \text{✿} \tag{9}$$

We can prove that the long-term value satisfies the Bellman equation:

$$V_\pi(s) = r(s, \pi(s)) + \gamma \mathbb{E}_{T(s,\pi(s))} [V_\pi] \quad \text{✿} \tag{10}$$

Note that the Bellman equation says that V_π is the fixed point of the operator

$$\mathrm{B}_\pi : (S \to \mathbb{R}) \to (S \to \mathbb{R}) \tag{11}$$

$$W \mapsto r(s, \pi(s)) + \gamma \mathbb{E}_{T(s,\pi(s))} W \tag{12}$$

Optimal Long-Term Value

The objective of an RL algorithm is to find an optimal policy, which gives the best long-term value $V_*(s) = \max_\pi\{V_\pi(s)\}$.

Optimal Long-Term Value

The objective of an RL algorithm is to find an optimal policy, which gives the best long-term value $V_*(s) = \max_\pi\{V_\pi(s)\}$.

> *Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (Bellman, 1957)*

Optimal Long-Term Value

The objective of an RL algorithm is to find an optimal policy, which gives the best long-term value $V_*(s) = \max_\pi\{V_\pi(s)\}$.

> *Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (Bellman, 1957)*

This suggests that V_* also satisfies a similar Bellman equation:

$$V_*(s) = \max_{a \in A(s)} \left\{ r(s,a) + \gamma \mathbb{E}_{T(s,a)} [V_*] \right\}$$

So, V_* is the fixed point of the operator

$$\begin{aligned} \hat{\mathrm{B}} :& (S \to \mathbb{R}) \to (S \to \mathbb{R}) \\ & W \mapsto \lambda s, \max_{a \in A(s)} \left(r(s,a) + \gamma \mathbb{E}_{T(s,a)}[W] \right) \end{aligned}$$ ✿

Value Iteration

- The operators B_π and $\hat{\mathrm{B}}$ are contractions wrt the max norm on $S \to \mathbb{R}$.

Value Iteration

- The operators B_π and $\hat{\mathrm{B}}$ are contractions wrt the max norm on $S \to \mathbb{R}$.
- The long-term value V_π and the optimal value V_* are their respective fixed points.

Value Iteration

- The operators B_π and $\hat{\mathrm{B}}$ are contractions wrt the max norm on $S \to \mathbb{R}$.
- The long-term value V_π and the optimal value V_* are their respective fixed points.
- The optimal policy π_* can be computed once we know V_*.
- The Banach Fixed Point theorem says that the function iterates of a contractive map converge to its fixed point.

Value Iteration

- The operators B_π and $\hat{\mathrm{B}}$ are contractions wrt the max norm on $S \to \mathbb{R}$.
- The long-term value V_π and the optimal value V_* are their respective fixed points.
- The optimal policy π_* can be computed once we know V_*.
- The Banach Fixed Point theorem says that the function iterates of a contractive map converge to its fixed point.
- This gives us a recipe to compute V_* and hence also the optimal policy.

Value Iteration

Value iteration proceeds by:

1. Initialize a value function $V_0 : S \to \mathbb{R}$.
2. Define $V_{n+1} = \hat{\mathrm{B}} V_n$ for $n \geq 0$. At each stage, the following policy (greedy policy) is computed

$$\pi_n(s) \in \operatorname{argmax}_{a \in A(s)} \left(r(s, a) + \gamma \mathbb{E}_{T(s,a)}[V_n] \right)$$

Value Iteration – Pseudocode

Data:

Markov decision process (S, A, T, r)

Initial value function $V_0 = 0$

Threshold $\theta > 0$

Discount factor $0 \leq \gamma < 1$

Result: V^*, the value function for an optimal policy.

```
for n from 0 to ∞ do
    for each s ∈ S do
        V_{n+1}[s] = max_a ( r(s, a) + γ E_{T(s,a)}[V_n] )
    end
    if ∀s |V_{n+1}[s] − V_n| < θ then
        return V_{n+1}
    end
end
```

Policy Iteration

Policy iteration is a similar iterative algorithm that benefits from a more definite stopping condition. Define the *Q function* to be:

$$Q_\pi(s, a) := r(s, a) + \gamma \mathbb{E}_{T(s,a)}[V_\pi].$$

The policy iteration algorithm proceeds in the following steps:

1. Initialize the policy to π_0.
2. Policy evaluation: For $n \geq 0$, given π_n, compute V_{π_n}.
3. Policy improvement: From V_{π_n}, compute the greedy policy:

$$\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A(s)} \left[Q_{\pi_n}(s, a) \right]$$

4. Check if $V_{\pi_n} = V_{\pi_{n+1}}$. If yes, stop.
5. If not, repeat (2) and (3).

Contraction Coinduction

The convergence proofs rely on the classical Banach fixed point theorem:

Theorem (Banach fixed point theorem on subsets ✿)

Let (X, d) *be a complete metric space and* ϕ *a closed nonempty subset of* X. *Let* $F : X \to X$ *be a contraction and assume that* F *preserves* ϕ. *Then* F *has a unique fixed point in* ϕ; *i.e., a point* $x^* \in X$ *such that* $\phi(x^*)$ *and* $F(x^*) = x^*$. *The fixed point of* F *is given by* $x^* = \lim_{n\to\infty} F^{(n)}(x_0)$ *where* $F^{(n)}$ *stands for the n-th iterate of the function* F.

Contraction Coinduction

We can restate this as:

$$\frac{\phi : X \to \mathsf{Prop} \quad \phi \text{ closed} \quad \exists x_0, \phi(x_0) \quad \phi(u) \to \phi(F(u))}{\phi(\mathsf{fix}\ F\ x_0)} \text{ ✿}$$

This is called Kozen's *metric coinduction*.

Contraction Coinduction

We can restate this as:

$$\frac{\phi : X \to \mathsf{Prop} \quad \phi \text{ closed} \quad \exists x_0, \phi(x_0) \quad \phi(u) \to \phi(F(u))}{\phi(\mathsf{fix}\ F\ x_0)} ✿$$

This is called Kozen's *metric coinduction*.

- This proof rule helps to streamline and simplify proofs of theorems about streams and stochastic processes.
- It allows us to automatically infer that a given (closed) property holds in the limit whenever it holds *ab initio*.
- Low level $\epsilon - \delta$ arguments – typically needed to show that a given property holds of the limit – are now neatly subsumed by a single proof rule, allowing reasoning at a higher level of abstraction.

Contraction Coinduction

Our convergence proofs use a specialized version of metric coinduction called *contraction coinduction* (following Feys, Hansen and Moss) to reason about order statements concerning fixed points of contractive maps.

Theorem (Contraction coinduction)

Let X be a non-empty, partially ordered, complete metric space in which the sets $\{x | x \leq y\}$ and $\{x | x \geq z\}$ are closed for all $y, z \in X$. If $F : X \to X$ is a contraction and is order-preserving, then:

- $\forall x, F(x) \leq x \Rightarrow x^* \leq x$ *and*
- $\forall x, x \leq F(x) \Rightarrow x \leq x^*$

where x^* is the fixed point of F.

Proofs by Contraction Coinduction

Theorem 15 (Proposition 1 of [FHM18] ✿). *The greedy policy is the policy whose long-term value is the fixed point of $\hat{\mathbf{B}}$:*

$$V_{\sigma_*} = \hat{V}$$

Proof.

(1) $V_{\sigma_*} \leq \hat{V}$ follows by Theorem 14.
(2) Now have to show $\hat{V} \leq V_{\sigma_*}$. Note that we have V_{σ_*} is the fixed point of B_{σ_*} by Theorem 11.
(3) We can now apply contraction coinduction with $F = \mathbf{B}_{\sigma_*}$.
(4) The hypotheses are satisfied since by Theorem 11, the $\mathbf{B}_{\sigma_*}$ is a contraction and it is a monotone operator.
(5) The only hypothesis left to show is $\hat{V} \leq \mathbf{B}_{\sigma_*}\hat{V}$.
(6) But in fact, we have $\mathbf{B}_{\sigma_*}(\hat{V}) = \hat{V}$ by the definition of σ_*.

□

(a) English proof adapted from [FHM18].

```
Lemma exists_fixpt_policy
: forall init,
  let V'  := fixpt (bellman_max_op) in
  let pi' := greedy init in
  ltv gamma pi' = V' init.
Proof.
intros init V' pi';
eapply Rfct_le_antisym; split.
  – eapply ltv_Rfct_le_fixpt.
  – rewrite (ltv_bellman_op_fixpt _ init).
     apply contraction_coinduction_Rfct_ge'.
    + apply is_contraction_bellman_op.
    + apply bellman_op_monotone_ge.
    + unfold V', pi'.
      now rewrite greedy_argmax_is_max.
Qed.
```

(b) Coq proof ✿

Proofs by Contraction Coinduction

The idea behind the proof of the policy improvement theorem is easy to understand. Starting from (4.7), we keep expanding the q_π side with (4.6) and reapplying (4.7) until we get $v_{\pi'}(s)$:

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(a)] && \text{(by (4.6))} \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\
&\ \ \vdots \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Conclusions & Future Work

- Proved convergence of classical value and policy iteration algorithms in Coq.
- Also computed optimal value functions (for a possibly non-stationary policy) for a finite horizon MDP.
- Used the Giry monad and contraction coinduction to streamline and simplify proofs.
- Stochastic Approximation algorithms for model-free RL algorithms such as Q-learning etc.