A Composable Specification Language for Reinforcement Learning Tasks

Kishor Jothimurugan, Rajeev Alur, Osbert Bastani
Control System

$s \in S$
$a \in A$

$S = $ Set of System States
$A = $ Set of Control Inputs

- Continuous states and actions
- System can be probabilistic
- Discrete Time
- Finite time horizon - $T$
Use neural networks to map states to actions

Design reward function $R$ mapping runs to rewards

Learn NN parameters optimizing:

$$\pi^* \in \arg\max_{\pi} \mathbb{E}_{\rho \sim D_\pi} [R(\rho)]$$
Reward Functions

- Too low-level as compared to logical specification
- No obvious way to compose rewards

\[ R_1 : \text{Reward function for “Reach q”} \]
\[ R_2 : \text{Reward function for “Reach p”} \]

Reward function for “Reach q and then Reach p”?
Need to generate reward function from a given logical specification
Need For Memory

- **Specification:** Reach q, then Reach p, then Reach r
- **Controller maps states to actions**
- **Action at p depends on the history of the run**

**Solution:** Add additional state component to indicate whether q has already been visited
Need to generate reward function from a given logical specification

Need to automatically infer the additional state components from the specification
Our Framework

- **System MDP** = \((S, A, P, T, s_0)\) where \(P(s, a, s') = \Pr(s' | s, a)\) given as a black-box forward simulator

- **Specification** \(\varphi\) given in our task specification language

Synthesizes a control policy \(\pi^*\) such that,

\[\pi^* \in \arg\max_{\pi} \Pr[\rho \models \varphi]\]
Our Framework

System → Product MDP

Specification → Nondeterministic Task Monitor → Reward Function

Monitor Automaton → Reinforcement Learning Algorithm

Control Policy
**Task Specification Language**

\[ \phi := \text{achieve } b \mid \phi_1 \text{ ensuring } b \mid \phi_1 ; \phi_2 \mid \phi_1 \text{ or } \phi_2 \]

- Example base predicates:
  - \( \text{Near}_q \) is satisfied if and only if the distance to \( q \) is less than 1
  - \( \text{Away}_o \) is satisfied if and only if there is a positive distance to \( O \)

- Specification for navigation example:
  \[
  (\text{achieve } \text{Near}_q; \text{achieve } \text{Near}_p) \text{ ensuring } \text{Away}_o
  \]
Quantitative Semantics

- Assume each base predicate $b \in P$ is associated with a quantitative semantics, $\llbracket b \rrbracket: S \rightarrow \mathbb{R}$ such that,

  $$s \models b \text{ if and only if } \llbracket b \rrbracket(s) > 0$$

  - $\llbracket \text{Near}_q \rrbracket(s) = 1 - \text{dist}(s, q)$
  - $\llbracket \text{Away}_0 \rrbracket(s) = \text{dist}(s, O)$

- Extend to positive Boolean combinations by,

  - $\llbracket b_1 \lor b_2 \rrbracket = \max(\llbracket b_1 \rrbracket, \llbracket b_2 \rrbracket)$
  - $\llbracket b_1 \land b_2 \rrbracket = \min(\llbracket b_1 \rrbracket, \llbracket b_2 \rrbracket)$
Task Monitor

- Finite State Machine
- Registers that store quantitative information
- Compilation similar to NFA construction from regular expressions

Task Monitor for $\phi = \text{achieve } b$
Task Monitor

Task monitor for \((\text{achieve } \text{Near}_q; \text{achieve } \text{Near}_p)\) ensuring \(\text{Away}_0\)

\[ u: \quad x_3 \leftarrow \min(x_3, \text{dist}(s, O)) \]
Extended Policy

Monitor state (q) -> Map state q to neural network

System state -> Neural Network for state q

Register values -> Neural Network for state q

System action -> Next monitor transition
Assigning Rewards

Given a sequence of extended system states, \( \rho = (q_0, s_0, v_0) \rightarrow \cdots \rightarrow (q_T, s_T, v_T) \) what should be its reward?

- **Case 1:** \( q_T \) is a final state. Reward is given by monitor
- **Case 2:** \( q_T \) not a final state. Not all tasks have been completed
  - Suggestion 1: \( R(\rho) = -\infty \)
  - Suggestion 2: Find a reward function \( R' \) that preserves ordering of runs \( R \),

\[
R(\rho) > R(\rho') \text{ implies } R'(\rho) > R'(\rho')
\]
Reward Shaping

Given \( \rho = (q_0, s_0, v_0) \to \cdots \to (q_T, s_T, v_T) \) with \( q_T \) non-final,

\[
R''(q_T)(s, v) = C_l + 2C_u(d_{q_T} - D) + \max_i [\sigma_i](s, v)
\]

\[
R'(\rho) = \max_{t: q_t = q_T} R''(q_T)(s_t, v_t)
\]

Higher reward for states farther from start

Prefer runs that get close to satisfying some predicate on transitions that make progress

- \( d_q \): Length of the longest path from \( q_0 \) to \( q \) without using self loops
- \( C_l \): Lower bound for possible reward in any final state
- \( C_u \): Upper bound on the third term for all \( q \)
Experiments

- Implemented our approach in a tool called SPECTRL (SPECifying Tasks for RL)
- Case study in the 2D navigation setting:
  - $S = \mathbb{R}^2$ and $A = \mathbb{R}^2$
  - Transitions given by $s_{t+1} = s_t + a_t + \epsilon$ where $\epsilon$ is a small gaussian noise
2D Navigation Tasks

Learning curves for different tasks
2D Navigation Tasks

Sample Complexity Curve
Y-axis denotes number of sample trajectories needed to learn
X-axis denotes number of nested goals
Cartpole

Learning Curve for Cartpole
Spec: Go to the right and return to start position without letting the pole fall
THANK YOU!