Reward Machine Construction Writeup

Maximilian Stollmayer

- general problem of reward specification
- ltl and other logics
- example with ltl
- from ltl to automaton
- construct rm from automaton
- conclusion and outlook

Reward Specification

Specifying the reward function, i.e. the rule that maps each state-action pair to a reward, is often a complex and error-prone process. In our self-driving car example from the previous section, we would need to account for numerous cases to design rewards that encourage the agent to both achieve the primary task of reaching the destination and satisfy constraints like stopping for pedestrians, etc. Crafting a reward function that balances all these considerations can quickly become unmanagable as tasks increase in complexity.

Tied to that is the problem of sparse rewards. In many scenarios the agent may receive no feedback for a majority of its actions. For example the self-driving car might only receive a positive reward after coming to a complete halt in front of a passing pedestrian. However it would get no reward for the crucial action of slowing down earlier, even though this behavior is critical for safety and anticipates the pedestrian's presence before they are fully visible. These sorts of scenarios can be very difficult to encode in the reward function.

There are a number of different ways to express the rewards in other languages than straight up hard coding the reward function in the implementation. An often used one is the linear temporal logic.

Linear Temporal Logics

In linear temporal logic (LTL) we can reason about the future of propositions, such as a proposition will eventually be true or will be true until another condition holds. The basic building blocks, called atomic propositions, are statements that are either true or false. Typically these represent some state of the system that we want to reason about, for example a door being closed or open or a value exceeding a threshold. These can be connected with the usual logical operators like $\vee$, $\neg$ and also temporal operators like *next*, $\bigcirc$, and *until*, U, that check if conditions hold in the future. Formally LTL is defined as follows.

Definition. *For a finite set of atomic propositions $\mathcal{P}$ the set of* LTL formula *is defined inductively as*

- $p \in \mathcal{P} \implies p$ *is a LTL formula*
- ψ, φ *LTL formula* $\implies \neg\psi, \psi \vee \varphi, \bigcirc\psi$ *and* $\varphi \, \mathsf{U} \psi$ *are LTL formula*

LTL operators over sequences of truth assignments and decides if a given formula is satisfied by this sequence. [after sequencing and satisfaction explain next and until] show graphs of these operators

So for the two definitional temporal operators, $\bigcirc\varphi$ is true when φ holds in the next time step. And $\varphi \, \mathsf{U} \, \psi$ means that φ must hold at least until the time step that ψ is true.

Definition. *Further symbols and operators that can be used in LTL formulas are:*

- true*,* $\top := p \vee \neg p$ *for some* $p \in \mathcal{P}$*, is always true*
- false*,* $\bot := \neg\top$*, is always false*
- and*,* $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$
- implication*,* $\varphi \rightarrow \psi := \neg\varphi \vee \psi$
- equivalence*,* $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- eventually*,* $\lozenge\varphi := \top \, \mathsf{U} \varphi$*, is true when* φ *holds at some time step in the future.*
- always*,* $\square\varphi := \neg\lozenge\neg\varphi$*, is true when* φ *holds from now on until forever.*

- release*, $\varphi \, \mathsf{R} \, \psi := \neg(\neg\varphi \, \mathsf{U} \neg\psi)$, ψ must hold until and including the point when φ first becomes true. If φ never becomes true then ψ must remain true forever.*

Parentheses are also allowed in formulas. There are some more temporal operator variations but these will suffice for us.

TODO: show graph of temporal operators

Example

give example of a simple agent and its reward specification in ltl and how that would yield a hard coded implementation. motivate the next step of converting that to an automaton for the rm construction

Construction

definitions and proof of ltl -¿ büchi -¿ dfa -¿ reward machine

Conclusion & Outlook

abstraction over the states an agent can be in, rm can be considered lingua franca since regular languages can always be translated into dfa and thus an rm, write about hierarchy and possible extensions

Basic Definitions

Definition. Propositional symbols *are statements that are either true or false.* Formulas *over propositional symbols consist of combinations of them with operations $\neg$, $\wedge$, $\vee$, $\implies$ and $\iff$. We say a formula ψ* provable *from a formula φ, if ψ can be derived from φ and write $\varphi \vdash \psi$.*

In the following we will suppose that we are in a reinforcement learning setting with a finite set of states S, with s_0 being the initial state, a set $T \subseteq S$ of terminal states and a finite set of actions A. Furthermore we suppose that we have a finite set of propositional symbols $\mathcal{P}$.

Definition. *A* labeling function *$L : S \times A \times S \to 2^{\mathcal{P}}$ maps experiences to truth assigments over propositional symbols $\mathcal{P}$.*

Definition. *A* Non-Markovian Reward Decision Process (NMRDP) *is a tuple* $(S, A, s_0, T, R, \gamma)$*, where* S, A, s_0, T *and* γ *are defined as in a regular MDP and* $R : (S \times A)^+ \times S \to \mathbb{R}$ *is a non-Markovian reward function that maps finite state-action histories into a real value. Note that* $X^+ := \bigcup_{n=1}^{\infty} X^n$ *represents all non-empty finite sequences of a set* X*.*

Reward Machines

Definition. *A* Mealy machine *is a tuple* $(U, u_0, \Sigma, \mathcal{R}, \delta, \rho)$*, where*

- U *is a finite set of states*
- $u_0 \in U$ *is the initial state*
- Σ *is a finite input alphabet*
- $\mathcal{R}$ *is a finite output alphabet*
- $\delta : U \times \Sigma \to U$ *is the transition function*
- $\rho : U \times \Sigma \to \mathcal{R}$ *is the output function*

Definition. *A* reward machine (RM) *is a Mealy machine* $(U, u_0, \Sigma = 2^{\mathcal{P}}, \mathcal{R}, \delta, \rho)$*, where* $\mathcal{R}$ *is a finite set of reward functions* $S \times A \times S \to \mathbb{R}$*.*

Definition. *The non-Markovian reward function* R induced by an RM $(U, u_0, 2^{\mathcal{P}}, \mathcal{R}, \delta, \rho)$ *is*

$$\begin{aligned} R : (S \times A)^+ \times S &\to \mathbb{R} \\ (s_0, a_0), \dots, (s_n, a_n), s_{n+1} &\mapsto \rho\big(u_n, L(s_n, a_n, s_{n+1})\big)(s_n, a_n, s_{n+1}) \\ R\big((s_0, a_0), \dots, (s_n, a_n), s_{n+1}\big) &= r(s_n, a_n, s_{n+1}) \end{aligned}$$

where $u_n = \delta\big(u_{n-1}, L(s_{n-1}, a_{n-1}, s_n)\big)$ *is defined recursively with the base case being the initial state* u_0*.*

Logics and Automata

ltl & co, dfa, dfa construction theorem and proof (source?)

Definition. *A* deterministic finite automaton (DFA) *is a tuple* $(U, u_0, \Sigma, \delta, F)$*, where*

- U *is a finite set of states*

- $u_0 \in U$ *is the initial state*
- Σ *is a finite input alphabet*
- $\delta : U \times \Sigma \to U$ *is the transition function*
- $F \subseteq U$ *is a set of accepting states*

Reward Specifications

Definition. *A* reward specification *is a set* $R = \{(r_1, \varphi_1), \ldots, (r_N, \varphi_N)\}$*, where each* $r_i \in \mathbb{R}$ *and* φ_i *is a formula over the propositional symbols* $\mathcal{P}$ *expressed in some regular language.*

Definition. *Let* $\tau = \big((s_0, a_0), \ldots, (s_n, a_n), s_{n+1}\big) \in (S \times A)^+ \times S$ *be a trace of experiences. We say that the projection of the experiences of* τ *by* L entails *a formula* φ*, and write* $\tau \vdash_L \varphi$*, if* $L(s_0, a_0, s_1) \ldots L(s_n, a_n, s_{n+1}) \vdash \varphi$*.*

Definition. *The non-Markovian reward function* $\hat{R}$ induced by the reward specification $R = \{(r_1, \varphi_1), \ldots, (r_n, \varphi_n)\}$ *assigns reward* $\hat{R}(\tau) := \sum_{k=1}^{N} \mathbb{1}(\tau \vdash_L \varphi_k)$ *to a trace* $\tau = \big((s_0, a_0), \ldots, (s_n, a_n), s_{n+1}\big) \in (S \times A)^+ \times S$*.*

Construction Theorem

Theorem. *There exists a reward machine that induces the same non-Markovian reward function as a given reward specification* $R = \{(r_1, \varphi_1), \ldots, (r_N, \varphi_N)\}$*.*

Proof. Let ... □

Corollary. *also ...*