SAPIENZA
UNIVERSITÀ DI ROMA

Reward shaping in RL for $\mathrm{LTL}_f/\mathrm{LDL}_f$ Goals: Theory and Practice

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Master in Engineering in Computer Science

Candidate

Marrco Favorito

ID number 1609890

Thesis Advisor

Prof. Giuseppe De Giacomo

Co-Advisor

Dr. co-advisor

Academic Year 2017/2018

Thesis defended on 1
in front of a Board of Examiners composed by:

Prof. ... (chairman)
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...

Reward shaping in RL for $\mathrm{LTL}_f/\mathrm{LDL}_f$ Goals: Theory and Practice
Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: favorito.1609890@studenti.uniroma1.it

Abstract

write your abstract here

Contents

Chapter 1

Introduction

Chapter 2

LTL_f and LDL_f

In this chapter we introduce the reader to the main important framework for talk about behaviors over time, which gives the basis for our approach. First we talk about the well known Linear time Temporal Logic LTL, PDL and their main applications; then we go more in deep by presenting a specific formalism, namely *Linear Temporal Logic over Finite Traces* LTL_f and *Linear Dynamic Logic over Finite Traces* LDL_f. We require the reader to know foundations of classical logic (Shapiro and Kouri Kissel, 2018).

2.1 Linear time Temporal Logic (LTL)

Temporal Logic (Goranko and Galton, 2015) is a category of formal languages aimed to talk about properties of a system whose truth value might change over time. This is in contrast with atemporal logics, which can only discuss about statements whose truth value is constant.

Linear time Temporal Logic (Pnueli, 1977), or *Linear Temporal Logic* (LTL) is such a logic. It is the most popular and widely used temporal logic in computer science, especially in formal verification of software/hardware systems, in AI to reasoning about actions and planning, and in the area of Business Process Specification and Verification to specify processes declaratively.

It allows to express temporal patterns about some property p, like *liveness* (p *will eventually happen*), *safety* (p *will never happen*) and *fairness*, combinations of the previous patterns (*infinitely often* p *holds*, *eventually always* p *holds*).

2.1.1 Syntax

A LTL formula φ is defined over a set of propositional symbols $\mathcal{P}$ and are closed under the boolean connectives, the unary temporal operator $\bigcirc$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

$$\varphi \quad ::= \quad A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

With $A \in \mathcal{P}$.

Additional operators can be defined in terms of the ones above: as usual logical operators such as $\vee, \Rightarrow, \Leftrightarrow$, *true*, *false* and temporal formulas like *eventually* as $\Diamond\varphi \doteq \mathit{true}\,\mathcal{U}\,\varphi$, *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$ and *release* as $\varphi_1 \,\mathcal{R}\, \varphi_2 \doteq \neg(\neg\varphi_1 \,\mathcal{U}\, \neg\varphi_2)$.

2.1.2 Semantics

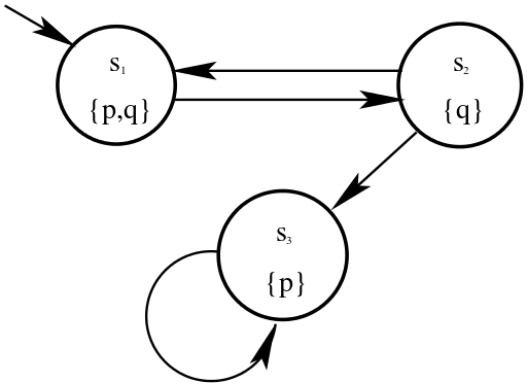
The semantics of LTL is provided by *traces*, i.e. ω-word over the alphabet $2^{\mathcal{P}}$. More formally, a *trace* is a *word* on a *path* of a *Kripke structure*.

Definition 2.1 ((Clarke et al., 1999))**.** a Kripke structure $\mathcal{K}$ over a set of propositional symbols $\mathcal{P}$ is a 4-tuple $\langle S, I, R, L\rangle$ where S is a finite set of states, $I \subseteq S$ is the set of initial states, $R \subseteq S \times S$ is the transition relation such that R is left-total and $L : S \to 2^{\mathcal{P}}$ is a labeling function.

A *path* ρ over $\mathcal{K}$ is a sequence of states $\langle s_1, s_2, \ldots\rangle$ such that $\forall i.R(s_i, s_{i+1})$. From a path we can build a *word* w on the path ρ by mapping each state of the sequence with L, namely:

$$w = \langle L(s_1), L(s_2), \ldots\rangle$$

In simpler words, a trace of propositional symbols $\mathcal{P}$ is a infinite sequence of combinations of propositional symbols in $\mathcal{P}$.



Several interesting temporal properties can be defined in LTL:

- a

2.2 Propositional Dynamic Logic (PDL)

2.2.1 Syntax

2.2.2 Semantics

2.3 Linear Temporal Logic on Finite Traces: LTL_f

Linear-time Temporal Logic over finite traces, LTL_f, is essentially standard LTL (Pnueli, 1977) interpreted over finite, instead of over infinite, traces (De Giacomo and Vardi, 2013).

Indeed, the syntax of LTL_f is the same of LTL_f, i.e. *formulas* of LTL_f are built from a set $\mathcal{P}$ of propositional symbols and are closed under the boolean connectives, the unary temporal operator $\bigcirc$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

$$\varphi \quad ::= \quad \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

With $A \in \mathcal{P}$.

We use the standard abbreviations: $\varphi_1 \vee \varphi_2 \doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$; *eventually* as $\Diamond\varphi \doteq \mathit{true}\,\mathcal{U}\,\varphi$; *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$; week next $\bullet\varphi \doteq \neg\bigcirc\neg\varphi$ (note that on finite traces $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$); and $\mathit{Last} \doteq \bullet\mathit{false}$ denoting the end of the trace.

Formally, a *finite trace* π is a finite word over the alphabet $2^{\mathcal{P}}$, i.e. as alphabet we have all the possible propositional interpretations of the propositional symbols in $\mathcal{P}$. For the semantics we refer to (De Giacomo and Vardi, 2013).

LTL$_f$ is as expressive as first-order logic (FO) over finite traces and star-free regular expressions (RE).

2.4 Linear Dynamic Logic on Finite Traces: LDL$_f$

LDL$_f$, *Linear Dynamic Logic of Finite Traces* merges LTL$_f$ with RE $_f$ (RE on finite traces) in a very natural way. The logic is called LDL f LTL$_f$ can be extended to

which is expressive as monadic second-order logic (MSO) over finite traces (De Giacomo and Vardi, 2013).

Linear Dynamic Logic of Finite Traces

Formally, LDL$_f$ formulas φ are built as follows:

$$\begin{array}{lcl} \varphi & ::= & \mathit{tt} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\varrho\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^* \end{array}$$

where tt stands for logical true; ϕ is a propositional formula over $\mathcal{P}$; ϱ denotes path expressions, which are RE over propositional formulas ϕ with the addition of the test construct $\varphi?$ typical of PDL. We use abbreviations $[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi$ as in PDL. Intuitively, $\langle\varrho\rangle\varphi$ states that, from the current step in the trace, there exists an execution satisfying the RE ϱ such that its last step satisfies φ, while $[\varrho]\varphi$ states that, from the current step, all executions satisfying the RE ϱ are such that their last step satisfies φ. Tests are used to insert into the execution path checks for satisfaction of additional LDL$_f$ formulas.

Given an LTL$_f$/LDL$_f$ formula φ, we can construct a deterministic finite state automaton (DFA) (**?**) $\mathcal{A}_\varphi$ that tracks satisfaction of φ, given a finite trace

accepting a sequence of propositional interpretations *iff* the sequence satisfies φ. This construction is a key element in the efficient transformation from non-Markovian rewards to Markovian rewards over an extended MDP (**?**). The idea is to use LTL$_f$/LDL$_f$ formulas to specify when sequences of state-action pairs, rather than one pair only, should be rewarded. Notice that we can easily incorporate the executed action in the current state by using propositions. In this way, we can make LTL$_f$/LDL$_f$ deal with actions, as well. From now on, we assume this is the case.

2.5 LTL$_f$ and LDL$_f$ translation to automata

Both LTL$_f$ and LDL$_f$ can be directly translated into alternating automata on words (AFW). Formally, an

delta function for the NFA:

$$
\begin{array}{rcl}
\partial(tt, \Pi) & = & \mathit{true} \\
\partial(\mathit{ff}, \Pi) & = & \mathit{false} \\
\partial(\phi, \Pi) & = & a \\
\partial(\varphi_1 \wedge \varphi_2, \Pi) & = & \partial(\varphi_1, \Pi) \wedge \partial(\varphi_2, \Pi) \\
\partial(\varphi_1 \vee \varphi_2, \Pi) & = & \partial(\varphi_1, \Pi) \vee \partial(\varphi_2, \Pi) \\
\partial(\langle\phi\rangle\varphi, \Pi) & = & \begin{cases} \boldsymbol{E}(\varphi) & \text{if } \Pi \models \phi \\ \mathit{false} & \text{if } \Pi \not\models \phi \end{cases} \\
\partial(\langle\varrho?\rangle\varphi, \Pi) & = & \partial(\varrho, \Pi) \wedge \partial(\varphi, \Pi) \\
\partial(\langle\varrho_1 + \varrho_2\rangle\varphi, \Pi) & = & \partial(\langle\varrho_1\rangle\varphi, \Pi) \vee \partial(\langle\varrho_2\rangle\varphi, \Pi) \\
\partial(\langle\varrho_1; \varrho_2\rangle\varphi, \Pi) & = & \partial(\langle\varrho_1\rangle\langle\varrho_2\rangle\varphi, \Pi) \\
\partial(\langle\varrho^*\rangle\varphi, \Pi) & = & \partial(\varphi, \Pi) \vee \partial(\langle\varrho\rangle\boldsymbol{F}_{\langle\varrho^*\rangle\varphi}, \Pi) \\
\partial([\phi]\varphi, \Pi) & = & \begin{cases} \boldsymbol{E}(\varphi) & \text{if } \Pi \models \phi \\ \mathit{true} & \text{if } \Pi \not\models \phi \end{cases} \\
\partial([\varrho?]\varphi, \Pi) & = & \partial(nnf(\neg\varrho), \Pi) \vee \partial(\varphi, \Pi) \\
\partial([\varrho_1 + \varrho_2]\varphi, \Pi) & = & \partial([\varrho_1]\varphi, \Pi) \wedge \partial([\varrho_2]\varphi, \Pi) \\
\partial([\varrho_1; \varrho_2]\varphi, \Pi) & = & \partial([\varrho_1][\varrho_2]\varphi, \Pi) \\
\partial([\varrho^*]\varphi, \Pi) & = & \partial(\varphi, \Pi) \wedge \partial([\varrho]\boldsymbol{T}_{\langle\varrho^*\rangle\varphi}, \Pi) \\
\partial(\boldsymbol{T}_\psi, \Pi) & = & \mathit{true} \\
\partial(\boldsymbol{F}_\psi, \Pi) & = & \mathit{false}
\end{array}
\tag{2.1}
$$

Chapter 3

FLLOAT

Chapter 4

RL for $\mathrm{LTL}_f/\mathrm{LDL}_f$ Goals

4.1 Reinforcement Learning

Reinforcement Learning (Sutton and Barto, 1998) is a sort of optimization problem where an *agent* interacts with an *environment* and obtains a *reward* for each action he chooses and the new observed state. The task is to maximize a numerical reward signal obtained after each action during the interaction with the environment. The agent does not know a priori how the environment works (i.e. the effects of his actions), but he can make observations in order to know the new state and the reward. Hence, learning is made in a *trial-and-error* fashion. Moreover, it is worth to notice that in many situation reward might not been affected only from the last action but from an indefinite number of previous action. In other words, the reward can be *delayed*, i.e. the agent should be able to foresee the effect of his actions in terms of future expected reward.

In the next subsections we introduce some of the classical mathematical frameworks for RL: Markov Decision Process (MDP) and Non-Markovian Reward Decision Process (NMRDP).

4.1.1 Markov Decision Process

A Markov Decision Process (MDP) $\mathcal{M}$ is a tuple $\langle S, A, T, R, \gamma \rangle$ containing a set of *states* S, a set of *actions* A, a *transition function* $T : S \times A \to Prob(S)$ that returns for every pair state-action a probability distribution over the states, a *reward function* $R : S \times A \times S \to \mathbb{R}$ that returns the reward received by the agent when he performs action a in s and transitions in s', and a *discount factor* γ, with $0 \leq \gamma \leq 1$, that indicates the present value of future rewards.

A *policy* $\rho : S \to A$ for an MDP $\mathcal{M}$ is a mapping from states to actions, and represents a solution for $\mathcal{M}$. Given a sequence of rewards $R_{t+1}, R_{t+2}, \ldots, R_T$, the *expected return* G_t at time step t is defined as:

$$G_t \coloneqq \sum_{t=k+1}^{T} \gamma^{k-t-1} R_k \tag{4.1}$$

where can be $T = \infty$ and $\gamma = 1$ (but not both).

The *value function* of a state s, the *state-value function* $v_\rho(s)$ is defined as the expected return when starting in s and following policy ρ, i.e.:

$$v_\rho(s) \coloneqq \mathbb{E}_\rho[G_t | S_t = s], \forall s \in S \tag{4.2}$$

Similarly, we define q_ρ, the *action-value function for policy* ρ, as:

$$q_\rho(s,a) \coloneqq \mathbb{E}_\rho[G_t|S_t = s, A_t = a], , \forall s \in S, \forall a \in A \tag{4.3}$$

Notice that we can rewrite 4.2 and 4.3 recursively, yielding the *Bellman equations*:

$$v_\rho(s) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma v(s')] \tag{4.4}$$

where we used the definition of the transition function:

$$T(s,a,s') = P(s'|s,a) \tag{4.5}$$

We define the *optimal state-value function* and the *optimal action-value function* as follows:

$$v^*(s) \coloneqq \max_\rho v_\rho(s), \forall s \in S \tag{4.6}$$

$$q^*(s,a) \coloneqq \max_\rho q_\rho(s,a), \forall s \in S, \forall a \in A \tag{4.7}$$

Notice that with 4.6 and 4.7 we can show the correlation between $v^*_\rho(s)$ and $q^*_\rho(s,a)$:

$$q^*(s,a) = \mathbb{E}_\rho[R_{t+1} + \gamma v^*_\rho(S_{t+1})|S_t = s, A_t = a] \tag{4.8}$$

We can define a partial order over policies using value functions, i.e. $\forall s \in S.\rho \geq \rho' \iff v_\rho(s) \geq v_{\rho'}(s)$. An *optimal policy* ρ^* is a policy such that $\rho^* \geq \rho$ for all ρ.

4.1.2 Temporal Difference Learning

Temporal difference learning (TD) refers to a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function. These methods sample from the environment, like Monte Carlo (MC) methods, and perform updates based on current estimates, like dynamic programming methods (DP). We do not discuss MC and DP methods here.

Q-Learning and Sarsa are such a methods. They updates $Q(s,a)$, i.e. the estimation of $q^*(s,a)$ at each transition $(s,a) \to (s',r)$. Q-Learning uses the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4.9}$$

while Sarsa:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{4.10}$$

TD(λ) is an algorithm which uses *eligibility traces*. The parameter λ refers to the use of an eligibility trace. The algorithm generalizes MC methods and TD learning, obtained respectively by setting $\lambda = 1$ and $\lambda = 0$. Intermediate values of λ yield methods that are often better of the extreme methods. Q-Learning and Sarsa that has been shown before can be rephrased with this new formalism as Q-Learning(0) and Sarsa(0), special cases of Watkin's Q(λ) and Sarsa(λ).

4.2 LTL$_f$ and LDL$_f$

In this section we introduce LTL$_f$ and LDL$_f$, two formalisms that we will use for define the reward function of a RL task over *sequence of transitions* rather than one transition.

4.3 RL for NMRDP with LTL$_f$/LDL$_f$ rewards

In this section we introduce the formalism of Non-Markovian Reward Decision Process (**?**)

4.3.1 Non-Markovian Reward Decision Process

A Non-Markovian Reward Decision Process (NMRDP) $\mathcal{N}$ is a tuple $\langle S, A, T, \bar{R}, \gamma \rangle$ where everything is defined as in the MDP but the reward function is defined as $\bar{R} : (S \times A)^* \to \mathbb{R}$, i.e. is defined over sequences of states and actions. Given a trace $\pi = \langle s_0, a_0, s_1, a_1, \ldots, s_n, a_n \rangle$, the *value of* π is:

$$v(\pi) = \sum_{i=1}^{|\pi|} \gamma^{i-1} \bar{R}(\langle \pi(1), \pi(2), \ldots, \pi(i) \rangle)$$

where $\pi(i) = (s_i, a_i)$.

The policy $\bar{\rho}$ in this setting is defined over sequences of states, i.e. $\bar{\rho} : S^* \to A$.

The *value of* $\bar{\rho}$ given an initial state s_0 is defined as:

$$v^{\bar{\rho}}(s) = \mathbb{E}_{\pi \sim \mathcal{M}, \bar{\rho}, s_0}[v(\pi)]$$

i.e. the expected value in state s considering the distribution of traces defined by the transition function of $\mathcal{M}$, the policy $\bar{\rho}$ and the initial state s_0.

4.4 RL for LTL$_f$/LDL$_f$ Goals

Chapter 5

Automata-based Reward shaping

5.1 Reward Shaping

5.2 Reward shaping over automaton φ

Chapter 6

RLTG

Chapter 7

Experiments

Chapter 8

Conclusions

Bibliography

Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03270-8.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, volume 13, pages 854–860, 2013.

Valentin Goranko and Antony Galton. Temporal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2015 edition, 2015.

Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Stewart Shapiro and Teresa Kouri Kissel. Classical logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2018 edition, 2018.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.