SAPIENZA
UNIVERSITÀ DI ROMA

Reinforcement Learning for $\mathrm{LTL}_f/\mathrm{LDL}_f$ Goals: Theory and Implementations

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Master in Engineering in Computer Science

Candidate

Marco Favorito
ID number 1609890

Thesis Advisor

Prof. Giuseppe De Giacomo

Co-Advisor

Prof. Daniele Nardi

Academic Year 2017/2018

Thesis defended on 20$^{\text{th}}$ July 2018
in front of a Board of Examiners composed by:

Prof. Riccardo Rosati (chairman)
Prof. Silvia Bonomi
Prof. Giorgio Grisetti
Prof. Massimo Mecella
Prof. Daniele Cono D'Elia

Reinforcement Learning for $\mathrm{LTL}_f/\mathrm{LDL}_f$ **Goals: Theory and Implementations**
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

1 Introduction **1**

2 LTL_f and LDL_f **2**

2.1 Linear time Temporal Logic (LTL) 2
2.1.1 Syntax 2
2.1.2 Semantics 3
2.2 Propositional Dynamic Logic (PDL) 4
2.2.1 Syntax 5
2.2.2 Semantics 6
2.3 Linear Temporal Logic on Finite Traces: LTL_f 8
2.3.1 Syntax 8
2.3.2 Semantics 10
2.3.3 Complexity and Expressiveness 11
2.4 Regular Temporal Specifications (RE_f) 11
2.5 Linear Dynamic Logic on Finite Traces: LDL_f 12
2.5.1 Syntax 12
2.5.2 Semantics 13
2.6 LTL_f and LDL_f translation to automata 14
2.6.1 ∂ function for LTL_f 15
2.6.2 ∂ function for LDL_f 16
2.6.3 The $\text{LDL}_f\text{2NFA}$ algorithm 17
2.6.4 Complexity of $\text{LTL}_f/\text{LDL}_f$ reasoning 22
2.7 Conclusions 24

3 FLLOAT **25**

3.1 Introduction 25
3.2 Package structure 25
3.3 Code examples 26
3.4 License 27

4 RL for $\text{LTL}_f/\text{LDL}_f$ Goals **28**

4.1 Reinforcement Learning 28
4.2 Markov Decision Process (MDP) 28
4.3 Temporal Difference Learning 32
4.4 Non-Markovian Reward Decision Process (NMRDP) 33
4.4.1 Preliminaries 33
4.4.2 Find an optimal policy $\bar{\rho}$ for NMRDPs 34
4.4.3 Define the non-Markovian reward function $\bar{R}$ 35
4.4.4 Using PLTL 36
4.5 NMRDP with $\text{LTL}_f/\text{LDL}_f$ rewards 36

4.6 RL for $\text{LTL}_f/\text{LDL}_f$ Goals . . . 38
4.6.1 Problem definition . . . 38
4.6.2 Examples . . . 40
4.6.3 Reduction to MDP . . . 40
4.6.4 An episodic goal-based view . . . 43
4.7 Conclusions . . . 44

5 Automata-based Reward shaping 45
5.1 Reward Shaping Theory . . . 45
5.1.1 Potential-Based Reward Shaping . . . 46
5.1.2 Dynamic Potential-Based Reward Shaping . . . 46
5.1.3 Relevant considerations about PBRS . . . 47
5.2 Off-line Reward shaping over $\mathcal{A}_\varphi$ . . . 49
5.3 On-The-Fly Reward shaping . . . 50

6 RLTG 52
6.1 Introduction . . . 52
6.2 Package structure . . . 52
6.3 Code examples . . . 53
6.4 License . . . 53

7 Experiments 54
7.1 BREAKOUT . . . 54
7.2 SAPIENTINO . . . 54
7.3 MINECRAFT . . . 54

8 Conclusions 55

Bibliography 56

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 foundations for our approach. First we talk about the well known Linear time Temporal Logic (LTL), Propositional Dynamic Logic (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. Finally, we study the translation from $\text{LTL}_f/\text{LDL}_f$ formulas to Deterministic Finite Automata (DFA). We require the reader to be acquainted with classical logic (Shapiro and Kouri Kissel, 2018) and automata theory (Hopcroft et al., 2000).

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 $\circ$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

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

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 $\Diamond\varphi \doteq true\,\mathcal{U}\,\varphi$, *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$ and *release* as $\varphi_1 \,\mathcal{R}\, \varphi_2 \doteq \neg(\neg\varphi_1 \,\mathcal{U}\, \neg\varphi_2)$.

Example 2.1. Several interesting temporal properties can be defined in LTL:

- *Liveness*: $\Diamond\varphi$, which means "condition expressed by φ *at some time* in the future will be satisfied", "sooner or later φ will hold" or "eventually φ will hold". E.g., $\Diamond \mathit{rich}$ (eventually I will become rich), $\mathit{Request} \implies \Diamond \mathit{Response}$ (if someone requested the service, sooner or later he will receive a response).
- *Safety*: $\Box\varphi$, which means "condition expressed by φ, *every time* in the future will be satisfied", "always φ will hold". E.g., $\Box \mathit{happy}$ (I'm always happy), $\Box\neg(\mathit{temperature} > 30)$ (the temperature of the room must never be over 30).
- *Response*: $\Box\Diamond\varphi$ which means "at any instant of time there exists a moment later where φ holds". This temporal pattern is known in computer science as *fairness*.
- *Persistence*: $\Diamond\Box\varphi$, which stand for "There exists a moment in the future such that from then on φ always holds". E.g. $\Diamond\Box \mathit{dead}$ (at a certain point you will die, and you will be dead forever)
- *Strong fairness*: $\Box\Diamond\varphi_1 \implies \Box\Diamond\varphi_2$, "if something is attempted/requested infinitely often, then it will be successful/allocated infinitely often". E.g., $\Box\Diamond \mathit{ready} \implies \Box\Diamond \mathit{run}$ (if a process is in ready state infinitely often, then infinitely often it will be selected by the scheduler).

2.1.2 Semantics

The semantics of LTL is provided by (infinite) *traces*, i.e. ω-word over the alphabet $2^{\mathcal{P}}$. More formally, a *trace* π 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 $\langle s_1, s_2, \dots\rangle$ 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:

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

In simpler words, a trace of propositional symbols $\mathcal{P}$ is a infinite sequence of combinations of propositional symbols in $\mathcal{P}$. Moreover, we denote by $\pi(i)$ with $i \in \mathbb{N}$ the labels associated to s_i, i.e. $L(s_i)$.

Example 2.2. In figure 2.1 is depicted an example of Kripke structure $\mathcal{K}$ over $\mathcal{P} = \{p, q\}$ where:

$$\begin{aligned}
S &= \{s_1, s_2, s_3\}\\
I &= \{s_1\}\\
R &= \{(s_1, s_2), (s_2, s_1), (s_2, s_3), (s_3, s_3)\}\\
L &= \{(s_1, \{p, q\}), (s_2, \{q\}), (s_3, \{p\})\}
\end{aligned}$$

The path $\langle s_1, s_2, s_3, s_3, s_3 \dots\rangle$ yields the following trace π:

$$\begin{aligned}
\pi &= \langle L(s_1), L(s_2), L(s_3), L(s_3), L(s_3), \dots\rangle\\
&= \langle \{p, q\}\{q\}, \{p\}, \{p\}, \{p\}, \dots\rangle
\end{aligned}$$

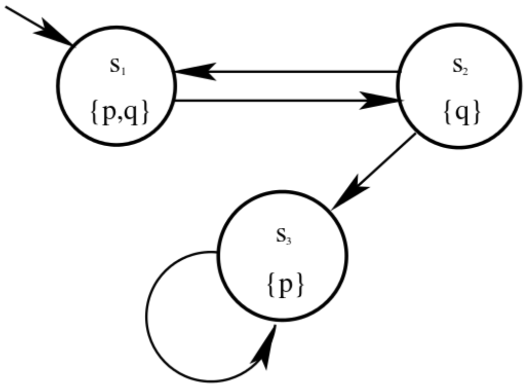


Figure 2.1. An example of Kripke structure.

Definition 2.2. Given a infinite trace π, we define that a LTL formula φ is *true* at time i, in symbols $\pi, i \models \varphi$ inductively as follows:

$$\pi, i \models A, \text{ for } A \in \mathcal{P} \text{ iff } A \in \pi(i)$$

$$\pi, i \models \neg\varphi \text{ iff } \pi, i \not\models \varphi$$

$$\pi, i \models \varphi_1 \wedge \varphi_2 \text{ iff } \pi, i \models \varphi_1 \wedge \pi, i \models \varphi_2$$

$$\pi, i \models \bigcirc\varphi \text{ iff } \pi, i+1 \models \varphi$$

$$\pi, i \models \varphi_1 \mathcal{U} \varphi_2 \text{ iff } \exists j.(j \geq i) \wedge \pi, j \models \varphi \wedge \forall k.(i \leq k < j) \Rightarrow \pi, k \models \varphi_1$$

Similiarly as in classical logic we give the following definitions:

Definition 2.3. A LTL formula is *true* in π, in notation $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula φ is *satisfiable* if it is true in some π and is *valid* if it is true in every π. φ_1 *entails* φ_2, in symbols $\varphi_1 \models \varphi_2$ iff $\forall \pi, \forall i. \pi, i \models \varphi_1 \implies \pi, i \models \varphi_2$.

Now we state an important result:

Theorem 2.1 (Sistla and Clarke (1985))**.** *Satisfiability, validity, and entailment for* LTL *formulas are* PSPACE*-complete.*

Indeed, Linear Temporal Logic can be thought of as a specific decidable (PSPACE-complete) fragment of classical first-order logic (FOL).

2.2 Propositional Dynamic Logic (PDL)

Dynamic Logics (Pratt, 1976b; Troquard and Balbiani, 2015) (DL) are modal logics[1] for representing the states and the events of dynamic systems. We can speak about

[1] *Modal Logic* extends classical logics to include operator expressing *modality* (e.g. "necessarily", "possibly", "usually"). However, the term "modal logic" is used more broadly to cover a family of logics with similar rules and a variety of different symbols. Temporal Logic and Dynamic Logic described in this chapter are examples of modal logics. (Garson, 2016)

the properties that holds in a state (assertion language) and about properties on transitions between states (programming language). Dynamic Logics are indeed called *logics of programs.*

Propositional Dynamic Logic (Fischer and Ladner, 1979) (PDL), probably the most well-known (propositional) logic of programs in computer science, is the propositional counterpart of Pratt's original dynamic logic (Pratt, 1976b), which was a *first-order* modal logic. Basically, this means that from three types of terms, *assertions*, *data* (as in FOL) and *actions* we drop the *data* terms, hence we can reason only about abstract propositions and the actions for modify them.

As we did with LTL, in the following sections we describe syntax and semantics of PDL.

2.2.1 Syntax

A PDL formula φ is defined over a set of propositional symbols $\mathcal{P}$ and a set of atomic programs Π built as follows:

$$\begin{array}{rcl} \varphi & ::= & A \mid \mathbf{0} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid [\alpha]\varphi \\ \alpha & ::= & \phi \mid \varphi? \mid \alpha_1 + \alpha_2 \mid \alpha_1 ; \alpha_2 \mid \varrho^* \end{array}$$

with $A \in \mathcal{P}$ and $\phi \in \Pi$. We can define classical logic operators $\vee, \Rightarrow, \Leftrightarrow, \mathbf{1}$ as usual, and the *possibility* operator $\langle\ \rangle$ from the *necessity* operator $[\ ]$, namely $\langle\alpha\rangle\varphi \doteq \neg[\alpha]\neg\varphi$. The propositions $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ are read "box α φ" and "diamond α φ", respectively.

Notice that φ stands for the propositional component of the logic, while program α stands for the dynamic component. Moreover, notice that propositions and programs are intertwined and cannot be separated: the definition of propositions depends on the definition of programs because of the construct $[\alpha]\varphi$, and the definition of programs depends on the definition of propositions because of the construct $\varphi?$.

Example 2.3. Now we provide some example of compound formulas and programs:

- $[\alpha]\varphi$: "It is necessary that after executing α, φ is true";
- $\langle\alpha\rangle\varphi$ "There exists a computation of α that terminates in a state satisfying φ.
- $\alpha;\beta$: "Execute α, then execute β";
- $\alpha;\cup\beta$: "Choose either α or β nondeterministically and execute it";
- α^*: "Choose α a nondeterministically chosen finite number of times (zero or more);
- $\varphi?$: "Test φ: proceed if true, fail if false".

Example 2.4. To better understand the expressive power of PDL, it is worth to notice this correspondence between basic programming language constructs and PDL formulas:

$$\begin{aligned} \mathbf{skip} &\coloneqq \mathbf{1} \\ \mathbf{fail} &\coloneqq \mathbf{0} \\ \mathbf{if}\ \varphi\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta &\coloneqq \varphi?;\alpha \cup \neg\varphi?;\beta \\ \mathbf{while}\ \varphi\ \mathbf{do}\ \alpha &\coloneqq (\varphi?;\alpha)^*;\neg\varphi? \end{aligned}$$

$$\textbf{repeat } \alpha \textbf{ until } \varphi \coloneqq \alpha; (\neg\varphi?;\alpha)^*; \varphi?$$
$$\{\varphi\}\alpha\{\psi\} \coloneqq \varphi \implies [\alpha]\varphi$$

The programs **skip** and **fail** are the program that does nothing (no-op) and the failing program, respectively. The ternary **if-then-else** operator and the binary **while-do** operator are the usual conditional and while loop constructs found in conventional programming languages. The construct $\{\varphi\}\alpha\{\psi\}$ is the Hoare partial correctness assertion (Pratt, 1976a).

2.2.2 Semantics

The semantics for PDL formulas is provided by *Labelled Transition System* (LTS).

Definition 2.4. A *Labelled Transition System* over a set of propositional symbols $\mathcal{P}$ and a set of atomic programs Π is a 3-tuple $\langle S, R_p, V\rangle$ where S is the set of *states*, $R_p : \Pi \to 2^{S\times S}$ is a mapping from atomic programs to a binary relation over S and $V : \mathcal{P} \to 2^S$ is a mapping from propositional symbols to subsets of S.

Example 2.5. In figure 2.2 two examples of LTS defined over $\mathcal{P} = \{p, q\}$ and $\Pi = \{\pi_1, \pi_2\}$ are depicted. For the LTS on the left, $\mathcal{M}_1$, we have:

$$\begin{aligned} S &= \{x_1, x_2\} \\ R_p(\pi_1) &= \{(x_1, x_1)\} \\ R_p(\pi_2) &= \{(x_1, x_2)\} \\ V(p) &= \{x_1\} \\ V(q) &= \{x_2\} \end{aligned}$$

While for the LTS on the right, $\mathcal{M}_2$, we have:

$$\begin{aligned} S &= \{y_1, y_2, y_3, y_4\} \\ R_p(\pi_1) &= \{(y_1, y_2), (y_2, y_2)\} \\ R_p(\pi_2) &= \{(y_1, y_3), (y_2, y_4)\} \\ V(p) &= \{y_1, y_2\} \\ V(q) &= \{y_3, y_4\} \end{aligned}$$

In order to formally define the semantics of a PDL formula φ, we use the following notation:

- $xR(\pi)y$ iff there exists an execution of π from x that leads to y;
- $x \in V(p)$ iff p is true in x.

In order to include all possible propositions and programs, we extend R_p and V inductively as follows:

- $xR_p(\alpha;\beta)y$ iff there exists a state z such that $xR_p(\alpha)z$ and $zR_p(\beta)y$

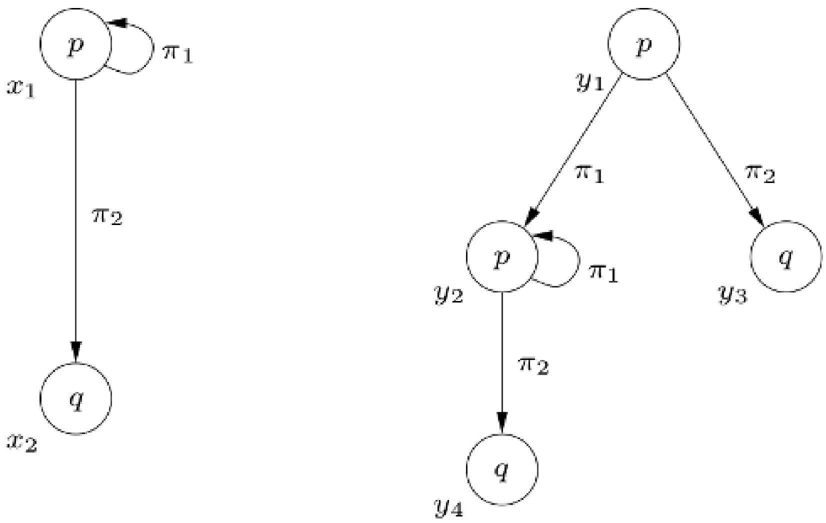


Figure 2.2. Two examples of LTS

- $xR_p(\alpha \cup \beta)y$ iff $xR_p(\alpha)y$ and $xR_p(\beta)y$
- $xR_p(\alpha*)y$ iff there exists an integer n and there exist states $z_0, \ldots, z_n$ such that $z_0 = x, z_n = y$ and $\forall.k = 1, \ldots, n$, $z_{k-1}R_p(\alpha)z_k$
- $xR_p(\varphi?)y$ iff $x = y \wedge y \in V(\varphi)$
- $V(0) = \emptyset$
- $V(\neg\varphi) = S \setminus V(\varphi)$
- $V(\varphi_1 \wedge \varphi_2) = V(\varphi_1) \wedge V(\varphi_2)$,
- $V([\alpha]\varphi) = \{x | \forall y.y \in S \wedge xR_p(\alpha)y \implies y \in V(\varphi)\}$

Now we give a definition for PDL formula satisfaction as we did in Definition 2.2:

Definition 2.5. Given a LTS $\mathcal{M}$, we define that a PDL formula φ is *true in a state* s, in symbols $\mathcal{M}, s \models \varphi$ iff $s \in V(\varphi)$:

Example 2.6. Considering $\mathcal{M}_1$ and $\mathcal{M}_2$ introduced in Example 2.5, we can give the following statements:

- $\mathcal{M}_1, x_1 \models p$
- $\mathcal{M}_1, x_2 \models q$
- $\mathcal{M}_1, x_1 \models \langle\pi_1\rangle p \wedge \langle\pi_2\rangle q$
- $\mathcal{M}_1, x_1 \models [\pi_1^*]p$
- $\mathcal{M}_2, y_1 \models \langle\pi_1^*; \pi_2\rangle q$
- $\mathcal{M}_2, y_1 \models [\pi_1 \cup \pi_2](q \vee p)$
- $\mathcal{M}_2, y_3 \models [\pi_1 \cup \pi_2]\mathbf{0}$

Definition 2.6. We define *satisfiability*, *validity* and *entailment* for PDL formulas in an analogous fashion as we did for LTL formulas in Definition 2.3.

Now we cite a result about complexity of reasoning in PDL:

Theorem 2.2 (Pratt (1980))**.** satisfiability*,* validity *and* entailment *in* PDL *is* EXPTIME*-complete.*

In (De Giacomo and Massacci, 2000) has been proposed an algorithm more effective in practice, though still running in deterministic exponential time in the worst case.

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). This apparently trivial difference has big impact: as we will see, some LTL formula has a different meaning if interpreted over infinite traces or finite ones.

2.3.1 Syntax

In fact, the syntax of LTL$_f$ is the same of the one showed in Section **??**, 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 $\circ$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

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

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

We use the standard abbreviations for classical logic formulas:

$$\begin{aligned}
\varphi_1 \vee \varphi_2 &\doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)\\
\varphi_1 \Rightarrow \varphi_2 &\doteq \neg\varphi_1 \vee \varphi_2\\
\varphi_1 \Leftrightarrow \varphi_2 &\doteq \varphi_1 \Rightarrow \varphi_2 \wedge \varphi_2 \Rightarrow \varphi_1\\
\mathit{true} &\doteq \neg\varphi \vee \varphi\\
\mathit{false} &\doteq \neg\varphi \wedge \varphi
\end{aligned}$$

And for temporal formulas:

$$\Diamond\varphi \doteq \mathit{true}\,\mathcal{U}\,\varphi \tag{2.1}$$

$$\Box\varphi \doteq \neg\Diamond\neg\varphi \tag{2.2}$$

$$\bullet\varphi \doteq \neg\circ\neg\varphi \tag{2.3}$$

$$\mathit{Last} \doteq \bullet\mathit{false} \tag{2.4}$$

$$\mathit{End} \doteq \Box\mathit{false} \tag{2.5}$$

As the reader might already noticed, 2.1 and 2.2 are defined as in Section **??**; Equation 2.3 is called *weak next* (notice that on finite traces $\neg\circ\varphi \not\equiv \circ\neg\varphi$); 2.4 denotes the end of the trace, while 2.5 denotes that the trace is ended.

Example 2.7. Here we recall Example 2.1 and we see the impact on *Always*, *Eventually Response* and *Persistence* LTL formulas if interpreted on finite traces (i.e. formulas in LTL$_f$):

- *Safety*: $\Box A$ means that always *till the end of the trace* φ holds;
- *Liveness*: $\Diamond A$ means that eventually *before the end of the trace* φ holds;
- *Response*: $\Box\Diamond\varphi$ on finite traces becomes equivalent to *last point in the trace satisfies* φ, i.e. $\Diamond(Last \wedge \varphi)$. Intuitively, this is true because $\Box\Diamond\varphi$ implies that at the last point in the trace φ holds (because there are no successive instants of time that make φ true); but if this is the case, then what happens at previous points in the trace does not matter because the formula evaluates always to true, since as we just said φ must hold at the last point in the trace, hence the equivalence with $\Diamond(Last \wedge \varphi)$.
- *Persistence*: $\Diamond\Box\varphi$ on finite traces becomes equivalent to *last point in the trace satisfies* φ, i.e. $\Diamond(Last \wedge \varphi)$. Analogously to the previous case, the equivalence holds because $\Diamond\Box\varphi$ implies that at the last point in the trace $\Box\varphi$ holds (and so φ), since we have no further successive instants of time that makes $\Box\varphi$ true. But if this is the case, then what happens at previous points in the trace does not matter because the formula evaluates always to true, since as we just said $\Box\varphi$ (and so φ) must hold at the last point in the trace, hence the equivalence with $\Diamond(Last \wedge \varphi)$.

In other words, no direct nesting of eventually and always connectives is meaningful in LTL$_f$, and this contrast what happens in LTL of infinite traces.

Example 2.8. Another remarkable evidence about the relevance of the assumption about finiteness of traces is provided by the DECLARE approach (Pesic and van der Aalst, 2006).

DECLARE is a declarative approach to business process modeling based on LTL interpreted over finite traces. The intuition is to map finite traces describing a domain of interest (e.g. processes) into infinite traces under the assumption that

$$\Diamond end \wedge \Box(end \Rightarrow \bigcirc end) \wedge \Box(end \Rightarrow \bigwedge_{p \in \mathcal{P}} \neg p) \tag{2.6}$$

which means that the following english statements hold:

- *end* eventually holds ($end \notin \mathcal{P}$);
- once *end* is true, it is true forever;
- when *end* is true all other propositions must be false

In other words, every finite trace π_f is extended with an infinite sequence of *end*, or in symbols $\pi_{inf} = \pi_f\{end\}^\omega$. By construction we have that

$$\pi_{inf} \models \Diamond end \wedge \Box(end \Rightarrow \bigcirc end) \wedge \Box(end \Rightarrow \bigwedge_{p \in \mathcal{P}} \neg p)$$

Despite it seems a nice construction to adapt LTL on finite traces, in fact it is wrong due to the *next* operator: in an infinite trace a successor state always exists, whereas in a finite one this does not hold. There exists a counterexample showing that the interpretation of LTL formulas on finite traces with the construction just explained is **not** equivalent with proper interpretation over finite traces offered by LTL$_f$, i.e. in general:

$$\pi_f\{end\}^\omega \models \varphi \not\Leftrightarrow \pi_f \models_f \varphi \tag{2.7}$$

To see why this is the case, consider the DECLARE "negation chain succession" $\Box(a \Rightarrow \circ\neg b)$ which requires that at any point in the trace, the state after we see a, b is false. Consider also the finite trace $\pi_f = \{a\}$ and the associated infinite trace $\pi_{inf} = \{a\}\{end\}^\omega$ built as explained before. We have that

$$\pi_{inf} \models \Box(a \Rightarrow \circ\neg b)$$

where $\models$ has been defined in 2.2. This is true because there is only one occurrence of a and then *end* holds forever (and so b does not).

But if the same formula is interpreted on finite traces (namely $\models_f$):

$$\pi_f \not\models_f \Box(a \Rightarrow \circ\neg b)$$

because the finite trace a is true at the last instant, but then there is no next instance where b is false, so $\circ\neg b$ is evaluated to *false* and the formula does not hold. The correct way to express "negation chain succession" on finite traces would be $\Box(a \Rightarrow \bullet\neg b)$.

The LTL formulas φ that are insensitive to the problem just shown, i.e. such that

$$\pi_f\{end\}^\omega \models \varphi \text{ iff } \pi_f \models_f \varphi \tag{2.8}$$

holds are defined *insensitive to infiniteness* (De Giacomo et al., 2014). This is another important evidence about the the relevance of the finiteness trace assumption.

2.3.2 Semantics

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 $\mathcal{P}$. We can see π as a *finite* word on a path of a Kripke structure, similarly as we discussed in Section 2.1.2 (but in that case the traces were *infinite*). Given a finite path $\rho = \langle s_1, s_2, \dots, s_n\rangle$ on a Kripke structure $\mathcal{K}$, a finite trace π associated to the path ρ is defined as $\langle L(s_1), L(s_2), \dots, L(s_n)\rangle$.

We use the following notation. We denote the *length* of a trace π as $length(\pi)$. We denote the i_{th} *position* on the trace as $\pi(i) = L(s_i)$, i.e. the propositions that hold in the i_{th} state of the path, with $0 \leq i \leq last$ where $last = length(\pi) - 1$ is the last element of the trace. We denote by $\pi(i,j)$, the *segment* of π, the trace $\pi' = \langle \pi(i), \pi(i+1), \dots, \pi(j)\rangle$, with $0 \leq i \leq j \leq last$

Definition 2.7. Given a finite trace π, we define that a LTL$_f$ formula φ is *true* at time i ($0 \leq i \leq last$), in symbols $\pi, i \models \varphi$ inductively as follows:

$$\pi, i \models A, \text{ for } A \in \mathcal{P} \text{ iff } A \in \pi(i)$$

$$\pi, i \models \neg\varphi \text{ iff } \pi, i \not\models \varphi$$

$$\pi, i \models \varphi_1 \wedge \varphi_2 \text{ iff } \pi, i \models \varphi_1 \wedge \pi, i \models \varphi_2$$

$$\pi, i \models \circ\varphi \text{ iff } i < last \wedge \pi, i+1 \models \varphi \tag{2.9}$$

$$\pi, i \models \varphi_1 \mathcal{U} \varphi_2 \text{ iff } \exists j.(i \leq j \leq \mathit{last}) \wedge \pi, j \models \varphi \wedge$$
$$\forall k.(i \leq k < j) \Rightarrow \pi, k \models \varphi_1 \tag{2.10}$$

Notice that Definition 2.7 is pretty similar to Definition 2.2, except the bounding of indexes in Equation 2.9 and Equation 2.10, to recognize that the trace is ended.

Analogously to Definition 2.3 we give the following definitions:

Definition 2.8. A LTL_f formula is *true* in π, in notation $\pi \models \varphi$, if $\pi, 0 \models \varphi$. A formula φ is *satisfiable* if it is true in some π and is *valid* if it is true in every π. φ_1 *entails* φ_2, in symbols $\varphi_1 \models \varphi_2$ iff $\forall \pi, \forall i.\pi, i \models \varphi_1 \implies \pi, i \models \varphi_2$.

2.3.3 Complexity and Expressiveness

Thanks to reduction of LTL_f satisfiability (Definition 2.8) into LTL satisfiability for PSPACE membership and reduction of STRIPS planning into LTL_f satisfiability for PSPACE-hardness, as proposed in (De Giacomo and Vardi, 2013), we have this result:

Theorem 2.3 (De Giacomo and Vardi (2013))**.** *Satisfiability, validity and entailment for* LTL_f *formulas are* PSPACE*-complete.*

About expressiveness of LTL_f, we have that:

Theorem 2.4 (De Giacomo and Vardi (2013); Gabbay et al. (1997))**.** LTL_f *has exactly the same expressive power of* FOL *over finite ordered sequences.*

2.4 Regular Temporal Specifications (RE_f)

In this section we talk about regular languages as a form of temporal specification over finite traces. In particular we focus on regular expressions (Hopcroft et al., 2000).

A regular expression ϱ is defined inductively as follows, considering as alphabet the set of propositional interpretations $2^{\mathcal{P}}$, from a set of propositional symbols $\mathcal{P}$:

$$\varrho \quad ::= \quad \phi \mid \varrho_1 + \varrho_2 \mid \varrho_1 ; \varrho_2 \mid \varrho^*$$

where ϕ is a propositional formula that is an abbreviation for the union of all the propositional interpretations that satisfy ϕ, i.e. $\phi = \sum_{\Pi \models \phi} \Pi$ and $\Pi \in 2^{\mathcal{P}}$.

We denote by $\mathcal{L}(\varrho)$ the language recognized by a RE_f expression. We interpret these expression over finite traces, introduced in Section 2.3.2.

Definition 2.9. We say that a regular expression ϱ *is true* in the finite trace π ifs $\pi \in \mathcal{L}(\varrho)$. We say that ϱ *is true at instant* i if $\pi(i, \mathit{last}) \in \mathcal{L}(\varrho)$. We say that ϱ *is true between instants* i, j if $\pi(i, j) \in \mathcal{L}(\varrho)$.

Example 2.9. We recall Example 2.2. The trace resulting from path $\langle s_1, s_2, s_3, s_3, \ldots \rangle$, i.e.:

$$\pi = \langle \{p, q\}\{q\}, \{p\}, \{p\}, \{p\}, \ldots \rangle$$

belongs to the language generated by the following regular expression:

$$\varrho_1 = p \wedge q; q; p^*$$

But also by this one:

$$\varrho_2 = \mathit{true}; q + p; \mathit{true}^*$$

Example 2.10. We can express some of the formulas shown in Example 2.7, and many others, in RE$_f$:

- *Safety*: φ^*, equivalent to $\Box\varphi$
- *Liveness*: $true^*; \varphi; true^*$, equivalent to $\Diamond\varphi$;
- *Response* and *Persistence*: as said before, when interpreted on finite traces, they are equivalent to $\Diamond(Last \wedge \varphi)$; hence, they can be rewritten in RE$_f$ as $true^*; \varphi$
- *Ordered occurrence*: $true^*; \varphi_1; true^*; \varphi_2; true^*$, equivalent to $\Diamond(\varphi_1 \wedge \bigcirc\Diamond\varphi_2)$ means that φ_1 and φ_2 happen in order;
- *Alternating sequence*: $(\psi, \varphi)^*$ means that ψ and φ alternate from the beginning of the trace, starting with ψ and ending with φ.

The *Alternating sequence* is an example of formula that has not a counterpart in LTL$_f$. More generally, LTL$_f$ (and LTL) are not able to capture regular structural properties on path (Wolper, 1981).

This observation about expressiveness of RE$_f$ is confirmed by Theorem 6 of (De Giacomo and Vardi, 2013), which is a consequence of several classical results (Büchi, 1960; Elgot, 1961; Trakhtenbrot, 1961; Thomas, 1979):

Theorem 2.5 (De Giacomo and Vardi (2013))**.** *RE$_f$ is strictly more expressive than LTL$_f$*

More precisely, RE$_f$ is expressive as *monadic second-order logic* MSO over bounded ordered sequences (Khoussainov and Nerode, 2001).

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

The problem with RE$_f$ is that, although is strictly more expressive than LTL$_f$, is considered a low-level formalism for temporal specifications. For instance RE$_f$ misses a direct construct for negation and for conjunction. Moreover, negation requires an exponential blow-up, hence adding complementation and intersection constructs is not advisable.

Linear Dynamic Logic of Finite Traces LDL$_f$ (De Giacomo and Vardi, 2013) merges LTL$_f$ with RE$_f$ in a very natural way, borrowing the syntax of PDL, described in Section 2.2. It keep the declarativeness and convenience of LTL$_f$ while having the same expressive power of RE$_f$.

2.5.1 Syntax

Formally, LDL$_f$ formulas φ are built over a set of propositional symbols $\mathcal{P}$ as follows (Brafman et al., 2017):

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

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 φ? typical of PDL. Moreover, we use the following abbreviations for classical logic operators:

$$\begin{aligned}
\varphi_1 \vee \varphi_2 &\doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 &\doteq \neg\varphi_1 \vee \varphi_2 \\
\varphi_1 \Leftrightarrow \varphi_2 &\doteq \varphi_1 \Rightarrow \varphi_2 \wedge \varphi_2 \Rightarrow \varphi_1 \\
\mathit{ff} &\doteq \neg tt
\end{aligned}$$

And for temporal formulas:

$$[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi \tag{2.11}$$

$$End \doteq [true]\mathit{ff} \tag{2.12}$$

$$Last \doteq \langle true\rangle End \tag{2.13}$$

$[\varrho]\varphi$ and $\langle\varrho\rangle\varphi$ are analogous to box and diamond operators in PDL; Formula 2.13 denotes the last element of the trace, whereas Formula 2.12 denotes that the trace is ended. 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. It is worth to notice that this interpretation of $[\,]$ and $\langle\,\rangle$ is pretty similar to the one shown in Section 2.2.1, as well as the test construct φ? are used to insert into the execution path checks for satisfaction of additional LDL$_f$ formulas.

2.5.2 Semantics

As we did in the previous sections, we formally give a semantics to LDL$_f$ (interpreted over finite traces, like LTL$_f$ and RE).

Definition 2.10. Given a finite trace π, we define that a LDL$_f$ formula φ is *true* at time i ($0 \leq i \leq last$), in symbols $\pi, i \models \varphi$ inductively as follows:

$\pi, i \models tt$

$\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$

$\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1 \wedge \pi, i \models \varphi_2$

$\pi, i \models \langle\phi\rangle\varphi$ iff $i < last \wedge \pi(i) \models \phi \wedge \pi, i+1 \models \varphi$

$\pi, i \models \langle\psi?\rangle\varphi$ iff $\pi, i \models \psi \wedge \pi, i \models \varphi$

$\pi, i \models \langle\varrho_1 + \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\varphi \vee \langle\varrho_2\rangle\varphi$

$\pi, i \models \langle\varrho_1; \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\langle\varrho_2\rangle\varphi$

$\pi, i \models \langle\varrho^*\rangle\varphi$ iff $\pi, i \models \varphi \vee i < last \wedge \pi, i \models \langle\varrho\rangle\langle\varrho^*\rangle\varphi$ and ϱ is not *test-only*

We say that ϱ is *test-only* if it is a RE$_f$ expression whose atoms are only tests, i.e. ψ?.

Notice that LDL$_f$ fully captures LTL$_f$. For every formula in LTL$_f$ there exists a LDL$_f$ formula with the same meaning, namely:

LTL$_f$	LDL$_f$
A	$\langle A\rangle tt$

$$
\begin{array}{rl}
\neg\varphi & \neg\varphi \\
\varphi_1 \wedge \varphi_2 & \varphi_1 \wedge \varphi_2 \\
\bigcirc\varphi & \langle true\rangle(\varphi \wedge \neg End) \\
\varphi_1 \,\mathcal{U}\, \varphi & \langle(\varphi_1?; true)^*\rangle(\varphi_2 \wedge \neg End)
\end{array}
$$

Notice also that every RE$_f$ expression ϱ is captured in LDL$_f$ by $\langle\varrho\rangle End$. Moreover, since also the converse holds, i.e. every LDL$_f$ formula can be expressed in RE (by Theorem 11 in (De Giacomo and Vardi, 2013)), the following theorem holds:

Theorem 2.6 (De Giacomo and Vardi (2013))**.** LDL$_f$ *has exactly the same expressive power of* MSO

Now we show several LDL$_f$ examples.

Example 2.11. Formulas described in Examples 2.7 and 2.10 can be rewritten in LDL$_f$ as:

- *Safety*: $[true^*]\varphi$, equivalent to LTL$_f$ formula $\Box\varphi$
- *Liveness*: $\langle true^*\rangle\varphi$, equivalent to LTL$_f$ formula $\Diamond\varphi$
- *Conditional Response*: $[true^*](\varphi_1 \Rightarrow \langle true^*\rangle\varphi_2)$, equivalent to LTL$_f$ formula $\Box(\varphi_1 \Rightarrow \Diamond\varphi_2)$
- *Ordered occurrence*: $\langle true^*; \varphi_1; true^*; \varphi_2; true^*\rangle End$ equivalent to the RE$_f$ expression $true^*; \varphi_1; true^*; \varphi_2; true^*$
- *Alternating occurrence*: $\langle(\psi;\varphi)^*\rangle End$ equivalent to the RE$_f$ expression $(\psi;\varphi)^*$

Example 2.12. Consider the Example 2.2 and 2.9. ϱ_1 and ϱ_2 are translated into LDL$_f$ as $\langle\varrho_1\rangle End$ and $\langle\varrho_2\rangle End$ respectively.

Other LDL$_f$ formulas satisfiable in the Kripke structure $\mathcal{K}$ depicted in Figure 2.1 are:

- $\langle p\rangle tt$ by every (non-empty) path, since s_1 is the initial state and we have that $\{p, q\} \models p$
- $\langle q\rangle tt$ as the previous case
- $\langle(p; q); (p; q)^*; p; p^*\rangle tt$ by paths of the form $\rho = s_1, s_2, (s_1, s_2)^\omega, s_3, (s_3)^\omega$
- $[true^*]\langle p \vee q\rangle tt$ is satisfied for every path, since for every reachable state either p or q are true;

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

Given an LTL$_f$/LDL$_f$ formula φ, we can construct a deterministic finite state automaton (DFA) (Rabin and Scott, 1959) $\mathcal{A}_\varphi$ that accept the same finite traces that makes φ true. In order to do this, we proceed in two steps: First we translate LTL$_f$ and LDL$_f$ formulas into (NFA) (De Giacomo and Vardi, 2015). Then the NFA obtained can be transformed into a DFA following the standard procedure of *determinization.*

Now we recall definitions of NFA and DFA:

Definition 2.11. An NFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where:

- Σ is the input alphabet;
- Q is the finite set of states;
- $q_0 \in Q$ is the initial state;
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation;
- $F \subseteq Q$ is the set of final states;

Definition 2.12. A DFA is a NFA where δ is a function $\delta : Q \times \Sigma \to Q$

By $\mathcal{L}(A)$ we mean the set of all traces over Σ accepted by $\mathcal{A}$.

In the next two subsections we give some definition that will be used in the algorithm; then we describe the algorithm for the translation and give some example.

2.6.1 ∂ function for LTL$_f$

We give the following definition:

Definition 2.13. The *delta function ∂ for LTL$_f$ formulas* is a function that takes as input an (implicitly quoted) LTL$_f$ formula φ in NNF and a propositional interpretation Π for $\mathcal{P}$, and returns a positive boolean formula whose atoms are (implicitly quoted) φ subformulas. It is defined as follows:

$$\begin{aligned}
\partial(A, \Pi) &= \begin{cases} \mathit{true} & \text{if } A \in \Pi \\ \mathit{false} & \text{if } A \notin \Pi \end{cases} \\
\partial(\neg A, \Pi) &= \begin{cases} \mathit{false} & \text{if } A \in \Pi \\ \mathit{true} & \text{if } A \notin \Pi \end{cases} \\
\partial(\varphi_1 \wedge \varphi_2, \Pi) &= \partial(\varphi_1, \Pi) \wedge \partial(\varphi_2, \Pi) \\
\partial(\varphi_1 \vee \varphi_2, \Pi) &= \partial(\varphi_1, \Pi) \vee \partial(\varphi_2, \Pi) \\
\partial(\circ\varphi, \Pi) &= \varphi \wedge \neg \mathit{End} \equiv \varphi \wedge \Diamond \mathit{true} \\
\partial(\varphi_1 \mathcal{U} \varphi_2, \Pi) &= \partial(\varphi_2, \Pi) \vee (\partial(\varphi_1, \Pi) \wedge \partial(\circ(\varphi_1 \mathcal{U} \varphi_2), \Pi)) \\
\partial(\Diamond\varphi, \Pi) &= \partial(\varphi, \Pi) \vee \partial(\circ\Diamond\varphi, \Pi) \\
\partial(\bullet\varphi, \Pi) &= \varphi \vee \mathit{End} \equiv \varphi \vee \Box \mathit{false} \\
\partial(\varphi_1 \mathcal{R} \varphi_2, \Pi) &= \partial(\varphi_2, \Pi) \wedge (\partial(\varphi_1, \Pi) \vee \partial(\bullet(\varphi_1 \mathcal{R} \varphi_2), \Pi)) \\
\partial(\Box\varphi, \Pi) &= \partial(\varphi, \Pi) \wedge \partial(\bullet\Box\varphi, \Pi)
\end{aligned} \tag{2.14}$$

where *End* is defined as Equation 2.5.

Moreover, we define $\partial(\varphi, \epsilon)$ which is inductively defined as Equation 2.14, except for the following cases:

$$\begin{array}{rcl} \partial(A, \epsilon) & = & \mathit{false} \\ \partial(\neg A, \epsilon) & = & \mathit{false} \\ \partial(\circ\varphi, \epsilon) & = & \mathit{false} \\ \partial(\bullet\varphi, \epsilon) & = & \mathit{true} \end{array} \tag{2.15}$$

Note that $\partial(\varphi, \epsilon)$ is always either *true* or *false*.

2.6.2 ∂ function for LDL$_f$

We give the following definition:

Definition 2.14. The *delta function ∂ for* LDL$_f$ *formulas* is a function that takes as input an (implicitly quoted) LDL$_f$ formula φ in NNF, extended with auxiliary constructs $\boldsymbol{F}_\psi$ and $\boldsymbol{T}_\psi$, and a propositional interpretation Π for $\mathcal{P}$, and returns a positive boolean formula whose atoms are (implicitly quoted) φ subformulas (not including $\boldsymbol{F}_\psi$ or $\boldsymbol{T}_\psi$). It is defined as follows:

$$\begin{array}{rcl} \partial(tt, \Pi) & = & \mathit{true} \\ \partial(\mathit{ff}, \Pi) & = & \mathit{false} \\ \partial(\phi, \Pi) & = & a \\ \partial(\varphi_1 \wedge \varphi_2, \Pi) & = & \partial(\varphi_1, \Pi) \wedge \partial(\varphi_2, \Pi) \\ \partial(\varphi_1 \vee \varphi_2, \Pi) & = & \partial(\varphi_1, \Pi) \vee \partial(\varphi_2, \Pi) \\ \partial(\langle\phi\rangle\varphi, \Pi) & = & \begin{cases} \boldsymbol{E}(\varphi) & \text{if } \Pi \models \phi \\ \mathit{false} & \text{if } \Pi \not\models \phi \end{cases} \\ \partial(\langle\varrho?\rangle\varphi, \Pi) & = & \partial(\varrho, \Pi) \wedge \partial(\varphi, \Pi) \\ \partial(\langle\varrho_1 + \varrho_2\rangle\varphi, \Pi) & = & \partial(\langle\varrho_1\rangle\varphi, \Pi) \vee \partial(\langle\varrho_2\rangle\varphi, \Pi) \\ \partial(\langle\varrho_1; \varrho_2\rangle\varphi, \Pi) & = & \partial(\langle\varrho_1\rangle\langle\varrho_2\rangle\varphi, \Pi) \\ \partial(\langle\varrho^*\rangle\varphi, \Pi) & = & \partial(\varphi, \Pi) \vee \partial(\langle\varrho\rangle\boldsymbol{F}_{\langle\varrho^*\rangle\varphi}, \Pi) \\ \partial([\phi]\varphi, \Pi) & = & \begin{cases} \boldsymbol{E}(\varphi) & \text{if } \Pi \models \phi \\ \mathit{true} & \text{if } \Pi \not\models \phi \end{cases} \\ \partial([\varrho?]\varphi, \Pi) & = & \partial(\mathit{nnf}(\neg\varrho), \Pi) \vee \partial(\varphi, \Pi) \\ \partial([\varrho_1 + \varrho_2]\varphi, \Pi) & = & \partial([\varrho_1]\varphi, \Pi) \wedge \partial([\varrho_2]\varphi, \Pi) \\ \partial([\varrho_1; \varrho_2]\varphi, \Pi) & = & \partial([\varrho_1][\varrho_2]\varphi, \Pi) \\ \partial([\varrho^*]\varphi, \Pi) & = & \partial(\varphi, \Pi) \wedge \partial([\varrho]\boldsymbol{T}_{\langle\varrho^*\rangle\varphi}, \Pi) \\ \partial(\boldsymbol{T}_\psi, \Pi) & = & \mathit{true} \\ \partial(\boldsymbol{F}_\psi, \Pi) & = & \mathit{false} \end{array} \tag{2.16}$$

where $\boldsymbol{E}(\varphi)$ recursively replaces in φ all occurrences of atoms of the form $\boldsymbol{T}_\psi$ and $\boldsymbol{F}_\psi$ by $\boldsymbol{E}(\psi)$.

Moreover, we define $\partial(\varphi, \epsilon)$ which is inductively defined as Equation 2.16, except for the following cases:

$$\begin{array}{rcl} \partial(\langle\phi\rangle\varphi, \epsilon) & = & \mathit{false} \\ \partial([\phi]\varphi, \epsilon) & = & \mathit{true} \end{array} \tag{2.17}$$

Note that $\partial(\varphi, \epsilon)$ is always either *true* or *false*.

2.6.3 The LDL$_f$2NFA algorithm

Algorithm 2.1 (LDL$_f$2NFA) takes in input a LDL$_f$/LTL$_f$ formula φ and outputs a NFA $\mathcal{A}_\varphi = \langle 2^\mathcal{P}, Q, q_0, \delta, F \rangle$ that accepts exactly the traces satisfying φ. It is a variant of the algorithm presented in (De Giacomo and Vardi, 2015), and its correctness relies on the fact that every LDL$_f$/LTL$_f$ formula φ can be associated a polynomial *alternating automaton on words* (AFW) accepting exactly the traces that satisfy φ and that every AFW can be transformed into an NFA (De Giacomo and Vardi, 2013). The proposed algorithm requires that φ is in *negation normal form* (NNF), i.e. with negation symbols occurring only in front of propositions.

The function ∂ used in lines 5, 12 and 15 is the one defined in sections 2.6.1 and 2.6.2; whether we are translating a LTL$_f$ or a LDL$_f$ formula, we use the function ∂ from Definition 2.13 and from Definition 2.14, respectively.

Algorithm 2.1. LDL$_f$2NFA: from LTL$_f$/LDL$_f$ formula φ to NFA $\mathcal{A}_\varphi$

1: **input** LDL$_f$/LTL$_f$ formula φ
2: **output** NFA $\mathcal{A}_\varphi = \langle 2^\mathcal{P}, Q, q_0, \delta, F \rangle$
3: $q_0 \leftarrow \{\varphi\}$
4: $F \leftarrow \{\emptyset\}$
5: **if** $(\partial(\varphi, \epsilon) = \mathit{true})$ **then**
6:     $F \leftarrow F \cup \{q_0\}$
7: **end if**
8: $Q \leftarrow \{q_0, \emptyset\}$
9: $\delta \leftarrow \emptyset$
10: **while** (Q or δ change) **do**
11:     **for** $(q \in Q)$ **do**
12:         **if** $(q' \models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi))$ **then**
13:             $Q \leftarrow Q \cup \{q'\}$
14:             $\delta \leftarrow \delta \cup \{(q, \Pi, q')\}$
15:             **if** $(\bigwedge_{(\psi \in q')} \partial(\psi, \epsilon) = \mathit{true})$ **then**
16:                 $F \leftarrow F \cup \{q'\}$
17:             **end if**
18:         **end if**
19:     **end for**
20: **end while**

How LDL$_f$2NFA works

The NFA $\mathcal{A}_\varphi$ for an LDL$_f$ formula φ is built in a forward fashion. Until convergence is reached (i.e. states and transitions do not change), the algorithm visits every state q seen until now, checks for all the possible transitions from that state and collects the results, determining the next state q', the new transition (q, Π, q') and if q' is a final state. Intuitively, the delta function ∂ emulates the semantic behavior of every LTL$_f$/LDL$_f$ subformula after seeing Π.

States of $\mathcal{A}_\varphi$ are sets of atoms (each atom is a quoted φ subformula) to be interpreted as conjunctions. The empty conjunction $\emptyset$ stands for *true*. q' is a set of quoted subformulas of φ denoting a minimal interpretation such that $q' \models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi)$ (notice that we trivially have $(\emptyset, p, \emptyset) \in \delta$ for every $p \in 2^\mathcal{P}$).

The following result holds:

Theorem 2.7 (De Giacomo and Vardi (2015))**.** *Algorithm* LDL$_f$2NFA *is correct, i.e., for every finite trace* $\pi : \pi \models \varphi$ *iff* $\pi \in \mathcal{L}(\mathcal{A}_\varphi)$. *Moreover, it terminates in at most an exponential number of steps, and generates a set of states* S *whose size is at most exponential in the size of the formula* φ.

In order to obtain a DFA, the NFA $\mathcal{A}_\varphi$ can be determinized in exponential time (Rabin and Scott, 1959). Thus, we can tranform a LTL$_f$/LDL$_f$ formula into a DFA of double exponential size.

Example 2.13. In this example we see a run of the Algorithm 2.1 with the LTL$_f$ formula $\Box A$ (A atomic).

0. Set up:

$$\begin{aligned} q_0 &= \{\Box A\} \\ Q &= \{q_0, \emptyset\} \\ F &= \{q_0, \emptyset\} \qquad \text{(because } \partial(\Box A, \epsilon) = \mathit{true}\text{)} \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset)\} \end{aligned}$$

1. Iteration: analyze $q = \{\Box A\}$

   - with $\Pi = \{A\}$ we have

$$\begin{aligned} q' &\models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi) \\ &\models \partial(\Box A, \Pi) \\ &\models \partial(A, \Pi) \wedge \partial(\bullet\Box A, \Pi) \\ &\models \mathit{true} \wedge (\text{“}\Box A\text{”} \vee \text{“}\Box \mathit{false}\text{”}) \end{aligned}$$

   Notice that $\mathit{true} \wedge (\text{“}\Box A\text{”} \vee \text{“}\Box \mathit{false}\text{”})$ is a *propositional formula* with LTL$_f$ formulas as atoms. As a minimal interpretation we have both $q' = \{\text{“}\Box A\text{”}\}$ and $q' = \{\text{“}\Box \mathit{false}\text{”}\}$. Since in both cases we have that $\partial(\psi, \epsilon) = \mathit{true}$, at the end of the iteration we have:

$$\begin{aligned} q_0 &= \{\Box A\} \\ Q &= \{q_0, \{\Box \mathit{false}\}, \emptyset\} \\ F &= \{q_0, \{\Box \mathit{false}\}, \emptyset\} \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset), \\ &\quad (q_0, \{A\}, q_0), (q_0, \{A\}, \{\Box \mathit{false}\})\} \end{aligned}$$

   - with $\Pi = \{\}$ we have

$$\begin{aligned} q' &\models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi) \\ &\models \partial(\Box A, \Pi) \end{aligned}$$

$$\models \partial(A, \Pi) \wedge \partial(\bullet\Box A, \Pi)$$

$$\models \textit{false} \wedge (\text{“}\Box A\text{”} \vee \text{“}\Box \textit{false}\text{”})$$

Which is always false. Thus we do not change nothing.

2. Iteration: we already analyzed $q = \{\Box A\}$, so we analyze $q = \{\Box \textit{false}\}$
   - Both with $\Pi = \{\}$ and $\Pi = \{A\}$ we have that:

$$q' \models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi)$$

$$\models \partial(\Box false, \Pi)$$

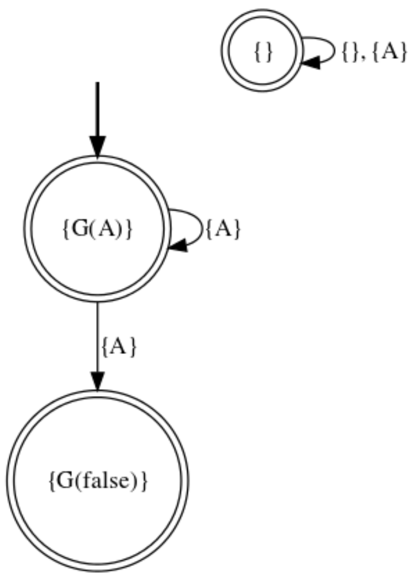
$$\models \partial(false, \Pi) \wedge \partial(\bullet\Box false, \Pi)$$

$$\models \textit{false} \wedge (\text{“}\Box false\text{”} \vee \text{“}\Box \textit{false}\text{”})$$

Which is always false. Thus we do not change nothing.

The NFA $\mathcal{A}_\varphi = \langle 2^{\{A\}}, Q, q_0, \delta, F\rangle$ is depicted in Figure 2.3, whereas the associated DFA is in Figure 2.4.

Figure 2.3. The NFA associated to $\Box A$. $G(A)$ stands for $\Box A$

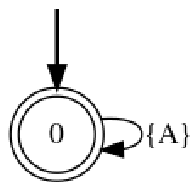


Example 2.14. Analogously to what we did in 2.13, we see a run of the Algorithm 2.1, with the LTL$_f$ formula $\Diamond A$ (A atomic).

0. Set up:

$$\begin{aligned} q_0 &= \{\Diamond A\} \\ Q &= \{q_0, \emptyset\} \\ F &= \{\emptyset\} \qquad \text{(because } \partial(\Diamond A, \epsilon) = \textit{false}) \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset)\} \end{aligned}$$

Figure 2.4. The DFA associated to $\Box A$



1. Iteration: analyze $q = \{\Diamond A\}$
   - with $\Pi = \{A\}$ we have

$$\begin{aligned} q' &\models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi) \\ &\models \partial(\Diamond A, \Pi) \\ &\models \partial(A, \Pi) \vee \partial(\bigcirc \Diamond A, \Pi) \\ &\models \mathit{true} \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond \mathit{true}\text{”}) \end{aligned}$$

   Since the propositional formula is trivially true, as a minimal interpretation we have $q' = \emptyset$. Considering that the empty conjunction is considered as *true* (as explained in Section 2.6), at the end of the iteration we have:

$$\begin{aligned} q_0 &= \{\Diamond A\} \\ Q &= \{q_0, \emptyset\} \\ F &= \{\emptyset\} \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset), (q_0, \{A\}, \emptyset)\} \end{aligned}$$

   - with $\Pi = \{\}$ we have

$$\begin{aligned} q' &\models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi) \\ &\models \partial(\Diamond A, \Pi) \\ &\models \partial(A, \Pi) \vee \partial(\bigcirc \Diamond A, \Pi) \\ &\models \mathit{false} \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond \mathit{true}\text{”}) \end{aligned}$$

   As a minimal interpretation we have $q' = \{\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond \mathit{true}\text{”}\}$. Since $\partial(\Diamond A, \epsilon) \wedge \partial(\Diamond \mathit{true}, \epsilon) = \mathit{false} \wedge \mathit{false} \neq \mathit{true}$, we do not add q' to the accepting states F. Thus we have:

$$\begin{aligned} q_0 &= \{\Diamond A\} \\ Q &= \{q_0, \emptyset, \{\Diamond A \wedge \Diamond \mathit{true}\}\} \end{aligned}$$

$$\begin{aligned} F &= \{\emptyset\} \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset), \\ &\quad (q_0, \{A\}, \emptyset), \\ &\quad (q_0, \{\}, \{\Diamond A \wedge \Diamond true\})\} \end{aligned}$$

2. Iteration: we already analyzed $q = \{\Diamond A\}$, so we analyze $q = \{\Diamond A \wedge \Diamond true\}$
   - with $\Pi = \{\}$ we have that:

$$\begin{aligned} q' &\models \bigwedge_{(\psi \in q)} \partial(\psi, \Pi) \\ &\models \partial(\Diamond A, \Pi) \wedge \partial(\Diamond true, \Pi) \\ &\models [\partial(A, \Pi) \vee \partial(\circ\Diamond A, \Pi)] \wedge [\partial(true, \Pi) \vee \partial(\circ\Diamond true, \Pi)] \\ &\models [\partial(A, \Pi) \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond true\text{”})] \wedge [true \vee (\text{“}\Diamond true\text{”} \wedge \text{“}\Diamond true\text{”})] \\ &\models \partial(A, \Pi) \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond true\text{”}) \\ &\models false \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond true\text{”}) \end{aligned}$$

     As in the previous iteration, the minimal model is $q' = \{\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond true\text{”}\}$. Hence we add a new transition $(\{\Diamond A \wedge \Diamond true\}, \{\}.\{\Diamond A \wedge \Diamond true\})$.
   - with $\Pi = \{A\}$ the delta-expansion is the same, except for the last step, where:

$$q' \models true \vee (\text{“}\Diamond A\text{”} \wedge \text{“}\Diamond true\text{”})$$

     The formula is always true, hence the minimal model is $q' = \emptyset$ and we add a new transition $(\{\Diamond A \wedge \Diamond true\}, \{\}.\emptyset)$.

   The NFA $\mathcal{A}_\varphi$ is then composed by:

$$\begin{aligned} q_0 &= \{\Diamond A\} \\ Q &= \{q_0, \emptyset, \{\Diamond A \wedge \Diamond true\}\} \\ F &= \{\emptyset\} \\ \delta &= \{(\emptyset, \{\}, \emptyset), (\emptyset, \{A\}, \emptyset), \\ &\quad (q_0, \{A\}, \emptyset), \\ &\quad (q_0, \{\}, \{\Diamond A \wedge \Diamond true\}) \\ &\quad (\{\Diamond A \wedge \Diamond true\}, \{\}.\{\Diamond A \wedge \Diamond true\}) \\ &\quad (\{\Diamond A \wedge \Diamond true\}, \{\}.\emptyset)\} \end{aligned}$$

The NFA $\mathcal{A}_\varphi = \langle 2^{\{A\}}, Q, q_0, \delta, F\rangle$ is depicted in Figure 2.5, whereas the associated DFA is in Figure 2.6.

Example 2.15. We list other examples of $\mathcal{A}_\varphi$ given a LTL$_f$/LDL$_f$ formula φ, obtained by Algorithm 2.1:

Figure 2.5. The NFA associated to $\Diamond A$. $F(A)$ stands for $\Diamond A$

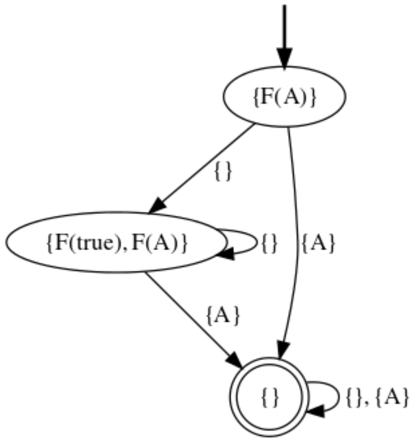
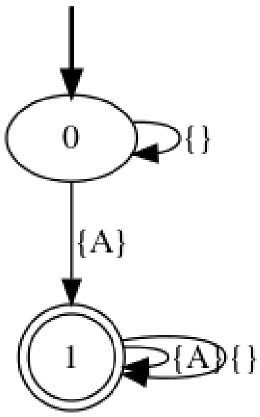


Figure 2.6. The DFA associated to $\Diamond A$



- *Conditional Response*: the LTL$_f$ formula $\varphi = \Box(A \Rightarrow \Diamond B)$ or equivalently the LDL$_f$ formula $\varphi = [true^*](\langle A\rangle tt \Rightarrow \langle true^*\rangle\langle B\rangle tt)$ translates into the automaton depicted in Figure 2.7.
- *Alternating sequence*: the LDL$_f$ formula $\varphi = \langle (A; B)^*\rangle End$ translates into the automaton depicted in Figure 2.8.

2.6.4 Complexity of LTL$_f$/LDL$_f$ reasoning

In this section we study the complexity of LTL$_f$/LDL$_f$ reasoning (i.e. complexity of problems as defined in Definition 2.8.

Theorem 2.8 (De Giacomo and Vardi (2013)). *Satisfiability, validity, and logical implication for* LDL$_f$ *formulas are* PSPACE*-complete*

Figure 2.7. The DFA associated to $\varphi = \Box(A \Rightarrow \Diamond B)$

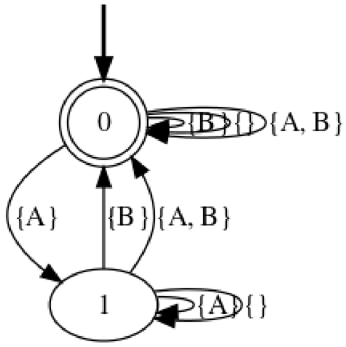
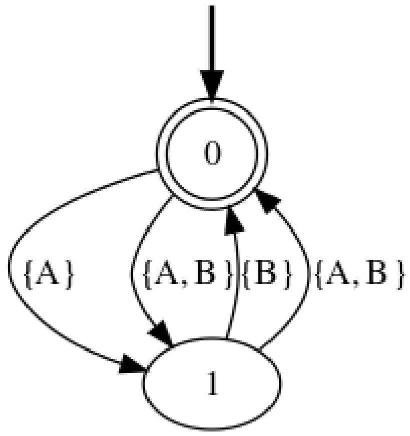


Figure 2.8. The DFA associated to $\varphi = \langle (A; B)^* \rangle End$



Proof. Given a LTL$_f$/LDL$_f$ φ, we can leverage Theorem 2.7 to solve these problems, namely:

- For LTL$_f$/LDL$_f$ satisfiability we compute the associated NFA (as explained in Section 2.6 (which is an exponential step) and then check NFA for nonemptiness (NLOGSPACE).
- For LTL$_f$/LDL$_f$ validity we compute the NFA associated to $\neg\varphi$ (which is an exponential step) and then check NFA for nonemptiness (NLOGSPACE).
- For LTL$_f$/LDL$_f$ logical implication $\psi \models \varphi$ we compute the NFA associated to $\psi \wedge \neg\varphi$ (which is an exponential step) and then check NFA for nonemptiness (NLOGSPACE).

$\square$

2.7 Conclusions

In this chapter we provided the logical tools to face other topics in later chapters. We introduced several formal languages that allowed us to introduce LTL$_f$ and LDL$_f$, focusing on their interesting properties. Moreover, we described in detail the procedure for translation from LTL$_f$/LDL$_f$ formulas to DFAs, which yields an effective way to reasoning about LTL$_f$/LDL$_f$ formulas.

Chapter 3

FLLOAT

In this chapter we describe FLLOAT (From $\text{LTL}_f/\text{LDL}_f$ tO AutomaTa), a software project written in Python. It is a porting of the homonym software project written in Java. It is the implementation of many of the topics described in Chapter 2.

3.1 Introduction

Main features: FLLOAT is a Python library that provides support for:

- Syntax, semantics and parsing of the following logic formalisms:
  - Propositional Logic;
  - Linear Temporal Logic on Finite Traces LTL_f
  - Linear Dynamic Logic on Finite Traces LDL_f;
- Conversion from $\text{LTL}_f/\text{LDL}_f$ formula to NFA, DFA and DFA On-The-Fly

Dependencies: FLLOAT requires Python>=3.5 and depends on the following packages:

- PLY, a pure-Python implementation of the popular compiler construction tools Lex and Yacc. It has been used for the parsing of PL and $\text{LTL}_f/\text{LDL}_f$ formulas;
- Pythomata, a Python package which provides support for NFA, DFA, determinization and minimization algorithms and reasoning on DFAs. It has been used for deal with $\mathcal{A}_\varphi$, the equivalent automaton of a $\text{LTL}_f/\text{LDL}_f$ formula.

Installation: You can find the package on PyPI, hence you can install it with:

```
pip install flloat
```

Please go here for further details.

3.2 Package structure

The package is structured as follows:

- `flloat.py`: the main module, it contains the implementation of the translation from $\text{LTL}_f/\text{LDL}_f$ formulas to automata. The functions implemented here are called from methods of $\text{LTL}_f/\text{LDL}_f$ formulas.
- `base`: contains the abstract definitions used in other modules. The main modules are:
  - `Symbol.py` and `Symbols.py`, where have been defined the class `Symbol` to represent the atomic propositional symbols and the operator symbols;
  - `Alphabet.py`, which is an abstraction for manage a set of `Symbol`;
  - `Interpretation.py`, an abstract class denoting the semantics used for truth evaluation. E.g. for PL the corresponding interpretation is `PLInterpretation` (a set of `Symbol`), whereas for $\text{LTL}_f/\text{LDL}_f$ we have `FiniteTrace`, which is a list of `PLInterpretation`.
  - `Formula.py`, the module containing the base class `Formula`. Every formula class extends `Formula`. In this module are defined also `AtomicFormula`, `Operator`, `BinaryOperator` etc., and how to build a syntactic tree.
  - `truths.py` and `nnf.py` that provide abstract implementations for truth evaluations of formulas and negation normal form operations.
  - other abstraction definitions that are implemented for each extending subclass.
- `syntax`: modules for each formalism (i.e. `pl.py`, `ltlf.py` and `ldlf.py`). In those modules are declared all the classes for representing formulas, implementing their truth evaluation procedure taking into account their correlation (e.g. `And` is the negative dual of `Or`, you can define `Implies` in terms of `Not` and `Or` etc.);
- `semantics`: modules providing implementations for the semantics. E.g. you can find `PLInterpretation` and `FiniteTrace` cited before;
- `parser`: modules where are defined the parsers of formulas in PL and $\text{LTL}_f/\text{LDL}_f$. They depends on PLY.

3.3 Code examples

Parse a LDL_f formula:

```
from flloat.parser.ldlf import LDLfParser

parser = LDLfParser()
formula = "<true*; A & B>tt"
# returns a LDLfFormula
parsed_formula = parser(formula)

# prints "<((true)* ; (A & B))>(tt)"
print(parsed_formula)
# prints {A, B}
print(parsed_formula.find_labels())
```

Evaluate it over finite traces:

```
from flloat.semantics.ldlf import FiniteTrace

t1 = FiniteTrace.fromStringSets([
{},
{"A"},
{"A"},
{"A", "B"},
{}
])
parsed_formula.truth(t1, 0)  # True
```

Transform it into an automaton (pythomata.DFA object):

```
dfa = parsed_formula.to_automaton(determinize=True)

# print the automaton
dfa.to_dot("./automaton.DFA")
```

Notice: `to_dot` requires Graphviz. For info about how to use a pythomata.DFA please look at the docs.

The same for a LTL$_f$ formula:

```
from flloat.parser.ltlf import LTLfParser
from flloat.base.Symbol import Symbol
from flloat.semantics.ldlf import FiniteTrace

# parse the formula
parser = LTLfParser()
formula = "F(A & !B)"
parsed_formula = parser(formula)

# evaluate over finite traces
t1 = FiniteTrace.fromStringSets([
{},
{"A"},
{"A"},
{"A", "B"}
])
assert parsed_formula.truth(t1, 0)

# from LTLf formula to DFA
dfa = parsed_formula.to_automaton(determinize=True)
assert dfa.word_acceptance(t1.trace)
```

3.4 License

The software is released under MIT license.

Chapter 4

RL for $\text{LTL}_f/\text{LDL}_f$ 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. Figure 4.1 represent the interaction between the agent and the environment in this setting.

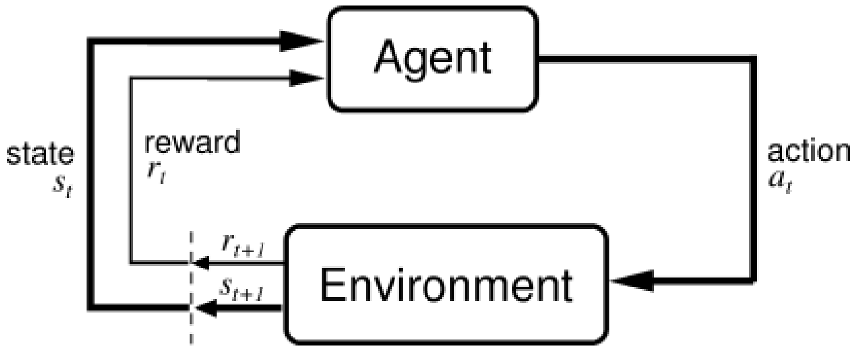


Figure 4.1. The agent and its interaction with the environment in Reinforcement Learning

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).

4.2 Markov Decision Process (MDP)

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 $0 \le \gamma \le 1$, that indicates the present value of future rewards. With $T(s, a, s')$ we denote the probability to end in state s' given the action a from s.

The discount factor γ deserves some attention. Its value highly influences the MDP, its solution, and how the agent interprets rewards. Indeed, if $\gamma = 0$, we are in the pure *greedy* setting, i.e. the agent is shortsighted and look only at the reward that it might obtain in the next step, by doing a single action. The higher γ, the longer the sight horizon, or the foresight, of the agent: the far rewards are taken into account for the current action choice. If $\gamma < 1$ we are in the *finite horizon* setting: namely, the agent is intrinsically motivated to obtain rewards as fast as possible, depending on how γ is far from 1. When $\gamma = 1$ we are in the *infinite horizon* setting, which means that the agent considers far rewards as they can be obtained in the next step. In other words, we may think the agent as *immortal*, since the time the agent spend to reach rewards does not matter anymore.

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 $R_{t+1}, R_{t+2}, \dots, R_T$, the *expected discounted return* G_t at time step t is defined as:

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

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 equation*:

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

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. $\forall s \in S. \rho \ge \rho' \iff v_\rho(s) \ge v_{\rho'}(s)$. Now we give the definition of optimal policy:

Definition 4.1. An *optimal policy* ρ^* is a policy such that $\rho^* \geq \rho$ for all ρ.

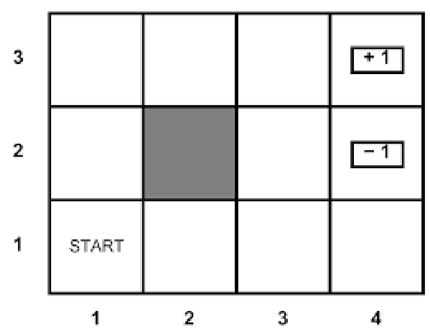
Given an MDP $\mathcal{M}$, a typical reinforcement learning problem is the following: find an optimal policy for $\mathcal{M}$, without knowing T and R. Notice that instead of explicit specification of the transition probabilities and rewards, the transition probabilities are accessed through a simulator that is restarted many times from a fixed or uniformly random initial state $s_0 \in S$. We call this way of structuring the learning process *episodic reinforcement learning*. Usually, in episodic reinforcement learning, we require the presence of one or more *goal states* where the simulation of the MDP ends and the task is considered completed, or a maximum time limit T for the number of action that can be taken by the agent in one single episode, and the overcoming of T determines the end of the episode. Optionally, *failure states* can be also defined, where the episode end similarly to goal states, but the task is considered failed.

Examples

Many dynamic systems can be modeled as Markov Decision Processes.

Example 4.1 (Gridworld). Perhaps the most simple MDP used as a toy example is *Gridworld*, depicted in Figure 4.2. There are 3×4 cells, i.e. states of the MDP $S = \{s_{11}, s_{12}, \ldots, s_{34}\} \setminus \{s_{22}\}$. The agent can do four actions: $A = \{Right, Left, Up, Down\}$. The initial state is fixed and is $s_0 = s_{11}$ and the agent can move in any of the adjacent and free cells from the current state. Assuming an episodic task, the goal is to reach s_{34}, and s_{24} represent a failure state. The state transition function T can be *deterministic*, i.e. the agent always succeed in performing actions, or non-deterministic, i.e. the effect of an action is determined by the probabilistic distribution returned by $T(s, a)$. An example of non-deterministic T is to give 90% of success (the agent moves in the chosen direction) and 10% of fail (the agent moves at either the right or left angle to the intended direction). If the move would make the agent walk into a wall (borders of the grid and s_{22}), the agent stays in the same place as before. The reward function $R(s, a, s')$ is defined as -1 if $s' = s_{24}$, as 1 if $s' = s_{34}$, and -0.01 otherwise. The small negative reward given at each transition is a popular mean for reward function design: it is called *step reward* and its purpose is to encourage the agent to finish the episode as fast as possible, with a priority proportional to the absolute value of the reward. The discount factor γ should be strictly higher than 0 because more than one step is needed to reach the goal state.

Figure 4.2. The Gridworld environment



An example of optimal policy is shown in Figure 4.3. As the reader can notice, the arrows represent the action that should be taken in a certain cell, in order to

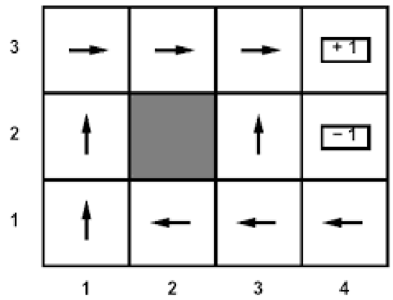


Figure 4.3. An example of optimal policy for the Gridworld environment

.

maximize the expected return. We observe that the optimal action in s_{13}, according to the policy, is not the one to take the shortest path to the goal, i.e. the Up action . This is because there is a small probability to ens in s_{24}, the failure state, and be punished with a high negative reward. In terms of expected reward, it is better to take the longer path, at the price of collect small negative rewards, but avoiding the risk to fail miserably.

Example 4.2 (**Breakout**)**.** Breakout is a well known arcade videogame developed by Atari. In this work we implemented a clone of the original Breakout. Figure 4.4 shows a screenshot of the game. On the screen there are a paddle at the bottom, many bricks at the top arranged in a grid layout with n rows and m columns (in the figure $3 \times 3 = 9$ bricks), and a ball that is free to move across the screen. The ball bounces when it hits a wall, a brick or the paddle. When the ball hits a brick, that brick is broke down and is removed from the screen. The paddle (the agent) can move left, move right or do nothing. The *goal* is to remove all the bricks, while avoiding that the paddle misses the rebound of the ball (*failure*).

The relevant features are: position of the paddle p_x, position of the ball b_x, b_y, speed of the ball v_x, v_y and status of each brick (booleans) b_{ij}. This features of the system gives all the needed information to predict the next state from the current state. Hence we can build an MDP where: S is the set of all the possible values of the sequence of features $\langle p_x, b_x, b_y, v_x, v_y, b_{11}, \dots, b_{nm} \rangle$, $A = \{Right, Left, No\text{-}op\}$, transition function T determined by the rules of the game. We give reward $R(s, a, s') = 10$ if a particular brick in s' has been removed for the first time, plus 100 if that brick was the last (i.e. goal reached).

Violation of the Markov property considering a smaller set of features: Notice that considering a strict subset of the set of features for S leads to violate the Markov property of T. Indeed, consider the case when we remove v_x and v_y from the set of features. In this setting, we removed the informations about the dynamics of the system. More precisely, we cannot predict, knowing only the current state, the value of the features b_x and b_y for the next step, because we do not know where the ball is going (up-left, down-right and so on). In order to correctly predict the next position of the ball, we should know whether earlier in the episode the ball was coming from the bottom or from the top. But this fact clearly shows that the Markovian assumption is violated. Similar arguments apply in the case where we remove the status of the bricks $b_{11}, \dots, b_{nm}$: indeed, if the ball in the next step is near to a brick, knowing about the status of the brick is determinant to predict if the ball will continue its trajectory (the case when the brick is absent) or it will

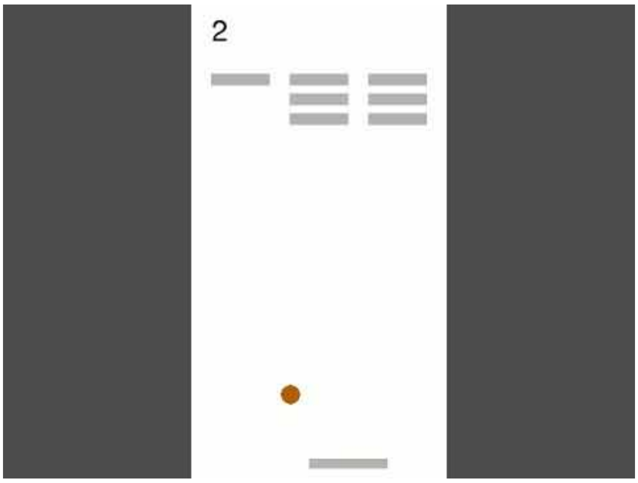


Figure 4.4. A screenshot from the videogame BREAKOUT
.

break down the brick and bounce, changing the direction of its motion (the case when the brick is present).

4.3 Temporal Difference Learning

Temporal difference learning (TD) (Sutton, 1988) 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) (Bellman, 1957). We do not discuss MC and DP methods here.

Q-Learning (Watkins, 1989; Watkins and Dayan, 1992) and SARSA are such a methods. They update $Q(s,a)$, i.e. the estimation of $q^*(s,a)$ at each transition $(s,a) \rightarrow (s',r)$. The update rule is the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta \tag{4.9}$$

where δ is the *temporal difference*. In SARSA, it is defined as:

$$\delta = r + \gamma Q(s',a') - Q(s,a) \tag{4.10}$$

whereas in Q-Learning:

$$\delta = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{4.11}$$

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) respectively. In this setting, Equation 4.9 is modified as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a) \tag{4.12}$$

Where $e(s, a) \in [0, 1]$, the *eligibility of the pair* (s, a), determines how much the temporal difference δ should be weighted. SARSA(λ) is reported in Algorithm 4.1, whereas Watkin's Q(λ) in Algorithm 4.2, both in the variants using *replacing eligibility traces* (see line 9 and line 10, respectively).

Algorithm 4.1. SARSA(λ) (Singh and Sutton, 1996)

1: Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ for all s, a
2: **repeat**{for each episode}
3:     initialize s
4:     Choose a from s using policy derived from Q (e.g. ϵ-greedy)
5:     **repeat**{for each step of episode}
6:         Take action a, observe reward r and new state s'
7:         Choose a' from s' using policy derived from Q
8:         $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
9:         $e(s, a) \leftarrow 1$ ▷ replacing traces
10:         **for all** s, a **do**
11:             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
12:             $e(s, a) \leftarrow \gamma \lambda e(s, a)$
13:         **end for**
14:         $s \leftarrow s'$, $a \leftarrow a'$
15:     **until** state s is terminal
16: **until**

4.4 Non-Markovian Reward Decision Process (NMRDP)

For some goals, it might be the case that the Markovian assumption of the reward function R – that reward depends only on the current state, and not on history – does not hold. Indeed, for many problems, it is not effective that the reward is limited to depend only on a single transition (s, a, s'); instead, it might be extended to depend on *trajectories* (i.e. $\langle s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n \rangle$), e.g. when we want to reward the agent for some (temporally extended) behaviors, opposed to simply reaching certain states.

This idea of rewarding behaviors has been proposed by (Bacchus et al., 1996) where they defined a new mathematical model, namely Non-Markovian Reward Decision Process (NMRDP), and showed how to construct optimal policies in this case.

In the next subsections, we give the main definitions to reason in this new setting. Then we show the solution proposed in (Bacchus et al., 1996).

4.4.1 Preliminaries

Now follows the definition of NMRDP, which is similiar to the MDP definition given in Section 4.2.

Definition 4.2. A Non-Markovian Reward Decision Process (NMRDP) (Bacchus et al., 1996) $\mathcal{N}$ is a tuple $\langle S, A, T, \bar{R}, \gamma \rangle$ where S, A, T and γ are defined as in the MDP, and $\bar{R} : S^* \to \mathbb{R}$ is the *non-Markovian reward function*, where $S^* = \{\langle s_0, s_1, \ldots, s_n \rangle_{n \geq 0, s_i \in S}\}$ is the set of all the possible traces, i.e. projection of trajectories $\langle s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n \rangle$

Algorithm 4.2. Watkin's Q(λ) (Watkins, 1989)

Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ for all s, a
repeat{for each episode}
  initialize s
  Choose a from s using policy derived from Q (e.g. e-greedy)
  repeat{for each step of episode}
    Take action a, observe reward r and new state s'
    Choose a' from s' using policy derived from Q (e.g. e-greedy)
    $a^* \leftarrow \arg\max_a Q(s', a)$ (if a' ties for max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
    $e(s, a) \leftarrow 1$ ▷ replacing traces
    for all s, a **do**
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
      if $a' = a^*$ **then**
        $e(s, a) \leftarrow \gamma\lambda e(s, a)$
      else
        $e(s, a) \leftarrow 0$
      end if
      $e(s, a) \leftarrow \gamma\lambda e(s, a)$
    end for
    $s \leftarrow s'$, $a \leftarrow a'$
  until state s is terminal
until

Given a trace $\pi = \langle s_0, s_1, \ldots, s_n \rangle$, the *value of* π is:

$$v(\pi) = \sum_{i=1}^{|\pi|} \gamma^{i-1} \bar{R}(\langle s_0, s_1, \ldots, s_n \rangle) \tag{4.13}$$

where $|\pi|$ denotes the number of transitions (i.e. of actions).

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:

$$v^{\bar{\rho}}(s) = \mathbb{E}_{\pi \sim \mathcal{N}, \bar{\rho}, s_0}[v(\pi)] \tag{4.14}$$

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

We are interested in two problems, that we will study in the next sections:

- Find an optimal (non-Markovian) policy $\bar{\rho}$ for an NMRDP $\mathcal{N}$ (Definition 4.2);
- Define the non-Markovian reward function for the domain of interest.

4.4.2 Find an optimal policy $\bar{\rho}$ for NMRDPs

The key difficulty with non-Markovian rewards is that standard optimization techniques, most based on Bellman's (Bellman, 1957) dynamic programming principle, cannot be used. Indeed, this requires one to resort to optimization over a policy space that maps histories (rather than states) into actions, a process that would incur great computational expense. (Bacchus et al., 1996) give the definition of a decision problem *equivalent* to an NMRDP in which the rewards are Markovian. This construction is the key element to solve our problem, i.e. find an optimal policy for an NMRDP.

Equivalent MDP

Now we give the definition of *equivalent* MDP of an NMRDP, and state an important result.

Definition 4.3 (Bacchus et al. (1996)). An NMRDP $\mathcal{N} = \langle S, A, T, \bar{R}, \gamma \rangle$ is *equivalent* to an extended MDP $\mathcal{M} = \langle S', A, T', R', \gamma \rangle$ if there exist two functions $\tau : S' \to S$ and $\sigma : S \to S'$ such that

1. $\forall s \in S : \tau(\sigma(s)) = s$;
2. $\forall s_1, s_2 \in S$ and $s'_1 \in S'$: if $T(s_1, a, s_2) > 0$ and $\tau(s'_1) = s_1$, there exists a unique $s'_2 \in S'$ such that $\tau(s'_2) = s_2$ and $T'(s'_1, a, s'_2) = T(s_1, a, s_2)$;
3. For any feasible trace $\langle s_0, s_1, \dots, s_n \rangle$ of $\mathcal{N}$ and $\langle s'_0, s'_1, \dots, s'_n \rangle$ of $\mathcal{M}$ associated to the trajectories $\langle s_0, a_0, \dots, s_{n-1}, a_{n-1}, s_n \rangle$ and $\langle s'_0, a_0, \dots, s'_{n-1}, a_{n-1}, s'_n \rangle$, such that $\tau(s'_i) = s_i$ and $\sigma(s_0) = s'_0$, we have $R(\langle s_0, s_1, \dots, s_n \rangle) = R'(s_{n-1}, a_{n-1}, s'_n)$.

Given the Definition 4.3, we give the definition of corresponding policy:

Definition 4.4 (Bacchus et al. (1996)). Let $\mathcal{N}$ be an NMRDP and let $\mathcal{M}$ be the equivalent MDP as defined in Definition 4.3. Let ρ be a policy for $\mathcal{M}$. The *corresponding policy* for $\mathcal{N}$ is defined as $\bar{\rho}(\langle s_0, \dots, s_n \rangle) = \rho(s'_n)$, where for the sequence $\langle s'_0, \dots, s'_n \rangle$ we have $\tau(s'_i) = s_i \ \forall i$ and $\sigma(s_0) = s'_0$

From definitions 4.3 and 4.4, and since that for all policy ρ of $\mathcal{M}$ the corresponding policy $\bar{\rho}$ of $\mathcal{N}$ is such that $\forall s. v_\rho(s) = v_{\bar{\rho}}(\sigma(s))$, the following theorem holds:

Theorem 4.1 (Bacchus et al. (1996)). *Let ρ be an optimal policy for MDP $\mathcal{M}$. Then the corresponding policy is optimal for NMRDP $\mathcal{N}$.*

The Theorem 4.1 allow us to learn an optimal policy $\bar{\rho}$ for NMRDP by learning a policy ρ over an equivalent MDP, which can be done by resorting on any off-the-shelf algorithm (e.g. see Section 4.3). Moreover, obtaining the corresponding policy for the original NMRDP is straightforward, although in practice is not needed, since it is enough to run the policy ρ over the MDP.

In other words, the problem of finding an optimal policy for an NMRDP reduces to find an optimal policy for an equivalent MDP such that Condition 1, 2 and 3 of Definition 4.3 hold.

4.4.3 Define the non-Markovian reward function $\bar{R}$

To reward agents for (temporally extended) behaviors, as opposed to simply reaching certain states, we need a way to specify rewards for specific trajectories through the state space. Specifying a non-Markovian reward function explicitly is quite hard and unintuitive, impossible if we are in a infinite-horizon setting. Instead, we can define *properties* over trajectories and reward only the ones which satisfy some of them, in contrast to enumerate all the possible trajectories.

Temporal logics presented in Section 2.1 gives an effective way to do this. Indeed, in order to speak about a desired behavior, i.e. fulfillment of properties that might change over time, we can define a *formula* φ (or more formulas) in some suited temporal logic formalism semantically defined over trajectories π, speaking about a set of properties $\mathcal{P}$ such that each state $s \in S$ is associated to a set of propositions ($S \subseteq 2^{\mathcal{P}}$). In this way, a trajectory $\pi = \langle s_0, a_0, \dots, s_{n-1}, a_{n-1}, s_n \rangle$ is rewarded with r_i iff $\pi \models \varphi_i$, where r_i is the reward value associated to the fulfillment of behaviors signified by φ_i.

4.4.4 Using PLTL

In (Bacchus et al., 1996) the temporal logic formalism is *Past Linear Temporal Logic* (PLTL), which is a past version of LTL (Section 2.1). As explained before, using the declarativeness of PLTL, is possible to specify the desired behavior (expressed in terms of the properties $\mathcal{P}$) that should be satisfied by the experienced trajectories and reward only them, hence obtaining a non-Markovian reward function. More formally, given a finite set Φ of PLTL *reward formulas*, and for each $\phi_i \in \Phi$ a real-valued reward r_i, the *temporally extended reward function* $\bar{R}$ is defined as:

$$\bar{R}(\langle s_0, s_1, \ldots, s_n \rangle) = \sum_{\phi_i \in \Phi : \langle s_0, s_1, \ldots, s_n \rangle \models \phi_i} r_i \tag{4.15}$$

In order to run the actual learning task, (Bacchus et al., 1996) proposed a transformation from the NMRDP to an equivalent MDP with the state space *expaneded* which allows to label each state $s \in S$. The idea is that the labels should keep track in some way the (partial) satisfaction of the temporal formulas $\phi_i \in \Phi$. A state s in the transformed state space is replicated multiple times, marking the difference between different (relevant) histories terminating in state s.

In this way, they obtained a compact representation of the required history-dependent policy by considering only relevant history, and can produce this policy using computationally-effective MDP algorithms. In other words, the states of the NMRDP can be mapped into those of the expanded MDP, in such a way that corresponding states yield same transition probabilities and corresponding traces have same rewards.

4.5 NMRDP with $\text{LTL}_f/\text{LDL}_f$ rewards

In this section we explain how to specify non-Markovian rewards with $\text{LTL}_f/\text{LDL}_f$ formulas (instead of PLTL) and how the associated MDP expansion works (Brafman et al., 2018), analogously to what we saw with PLTL (Section 4.4.4).

The temporally extended reward function $\bar{R}$ is similar to Equation 4.15, but instead of using PLTL formula we use $\text{LTL}_f/\text{LDL}_f$ formulas. Formally, given a set of pairs $\{(\varphi_i, r_i)_{i=1}^m\}$ (where φ_i denotes the $\text{LTL}_f/\text{LDL}_f$ formula for specifying a desired behavior, and r_i denotes the reward associated to the satisfaction of φ_i, and given a (partial) trace $\pi = \langle s_0, s_1, \ldots, s_n \rangle$, we define $\bar{R}$ as:

$$\bar{R}(\pi) = \sum_{1 \leq i \leq m : \pi \models \varphi_i} r_i \tag{4.16}$$

For the sake of clarity, in the following we use $\{(\varphi_i, r_i)_{i=1}^m\}$ to denote $\bar{R}$.

Now we describe the MDP expansion for doing learning in this setting, as proposed in (Brafman et al., 2018). Without loss of generality, we assume that every NMRDP $\mathcal{N}$ is reduced into another NMRDP $\mathcal{N}' = \langle S', A', T', R', \gamma \rangle$:

$$
\begin{aligned}
S' &= S \cup \{s_{init}\} \\
A' &= A \cup \{start\} \\
T'(s,a,s') &= \begin{cases} 1 & \text{if } \; s = s_{init}, a = start, s' = s_0 \\ 0 & \text{if } \; s = s_{init} \text{ and } (a \neq start \text{ or } s' \neq s_0) \\ T(s,a,s') & \text{otherwise} \end{cases} \\
R'(\langle s_{init}, s_0, \dots, s_n\rangle) &= R(\langle s_0, s_1, \dots, s_n\rangle)
\end{aligned}
\tag{4.17}
$$

and s_{init} is the new initial state. In other words, we prefix to every feasible trajectory $\mathcal{N}$ the pair $\langle s_{init}, start\rangle$, denoting the beginning of the episode. We do this for two reasons: allow to evaluate formulas in s_0 and make it compliant with the most general definition of the reward, namely $R(s,a,s')$, also when there is no true action that is done (i.e. empty trace).

Definition 4.5 (Brafman et al. (2018))**.** Given an NMRDP $\mathcal{N} = \langle S, A, T, \{(\varphi_i, r_i)_{i=1}^m, \gamma\}\rangle$ (i.e. with non-Markovian rewards specified by LTL$_f$/LDL$_f$ formulas) it is possible to build an $\mathcal{M} = \langle S', A, T', R', \gamma\rangle$ that is *equivalent* (in the sense of Definition 4.3) to $\mathcal{N}$. Denoting with $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i\rangle$ (notice that $S \subseteq 2^{\mathcal{P}}$ and δ_i is total) the DFA associated with φ_i (see Section 2.6), the equivalent MDP $\mathcal{M}$ is built as follows:

- $S' = Q_1 \times \dots \times Q_m \times S$ is the set of states;
- $T' : S' \times A \times S' \to [0,1]$ is defined as follows:

$$
\begin{aligned}
&Tr'(q_1, \dots, q_m, s, a, q_1', \dots, q_m', s') = \\
&\begin{cases} Tr(s,a,s') & \text{if } \forall i : \delta_i(q_i, s') = q_i' \\ 0 & \text{otherwise;} \end{cases}
\end{aligned}
$$

- $R' : S' \times A \times S' \to \mathbb{R}$ is defined as:

$$
R'(q_1, \dots, q_m, s, a, q_1', \dots, q_m', s') = \sum_{i: q_i' \in F_i} r_i
$$

Theorem 4.2 (Brafman et al. (2018))**.** *The NMRDP $\mathcal{N} = \langle S, A, T, \{(\varphi_i, r_i)\}_{i=1}^m, \gamma\rangle$ is equivalent to the MDP $\mathcal{M} = \langle S', A, T', R', \gamma\rangle$ defined in Definition 4.5.*

Proof. Recall that every $s' \in S'$ has the form $(q_1, \dots, q_m, s)$. Define $\tau(q_1, \dots, q_m, s) = s$. Define $\sigma(s) = (q_{10}, \dots, q_{m0}, s)$. We have $\tau(\sigma(s)) = s$, hence Condition 1 is verified. Condition 2 of Definition 4.3 is easily verifiable by inspection. For Condition 3, consider a possible trace $\pi = \langle s_0, s_1, \dots, s_n\rangle$. We use σ to obtain $s_0' = \sigma(s_0)$ and given s_i, we define s_i' (for $1 \leq i \leq n$) to be the unique state $(q_{1,i}, \dots, q_{m,i}, s_i)$ such that $q_{j,i} = \delta(q_{j,i-1}, s_i)$ for all $1 \leq j \leq m$. Moreover, we require that, without loss of generality, every trajectory in the new MDP starts from s_{init} and now have a corresponding possible trace of $\mathcal{M}$, i.e., $\pi = \langle s_0', s_1', \dots, s_n'\rangle$. This is the only feasible trajectory of $\mathcal{M}$ that satisfies Condition 3. The reward at $\pi = \langle s_0, s_1, \dots, s_n\rangle$ depends only on whether or not each formula φ_i is satisfied by π. However, by construction of the automaton $\mathcal{A}_{\varphi_i}$ and the transition function T, $\pi \models \varphi_i$ iff $s_n' = (q_1, \dots, q_m, s_n)$ and $q_i \in F_i$ ☐

Let ρ' be a (Markovian) policy for $\mathcal{M}$. It is easy to define an *corresponding* policy on $\mathcal{N}$, i.e., a policy that guarantees the same rewards, by using τ and σ mappings defined in Theorem 4.2 and the result shown in Theorem 4.4.

Obviously, typical learning techniques, such as Q-learning or SARSA, are applicable on the expanded $\mathcal{M}$ and so we can learn an optimal policy ρ for $\mathcal{M}$. Thus, an optimal policy for $\mathcal{N}$ can be learnt on $\mathcal{M}$. Of course, none of these structures is (completely) known to the learning agent, and the above transformation is never done explicitly. Rather, the agent carries out the learning process by assuming that the underlying model is $\mathcal{M}$ instead of $\mathcal{N}$ (applying the fix introduced in Definition 4.17).

Observe that the state space of $\mathcal{M}'$ is the product of the state spaces of $\mathcal{N}$ and $\mathcal{A}_{\varphi_i}$, and that the reward R' is Markovian. In other words, the (stateful) structure of the LTL$_f$/LDL$_f$ formulas φ_i used in the (non-Markovian) reward of $\mathcal{N}$ is *compiled* into the states of $\mathcal{M}$.

Why should we use LDL$_f$

LDL$_f$ formalism (introduced in Section 2.5) has the advantage of *enhanced expressive power* over other proposals, as discussed in (Brafman et al., 2018). Indeed, we move from linear-time temporal logics to LDL$_f$, paying no additional (worst-case) complexity costs. LDL$_f$ can encode in polynomial time LTL$_f$, regular expressions (RE) and the past LTL (PLTL) of (Bacchus et al., 1996). Moreover, LDL$_f$ can naturally represent "procedural constraints" (Baier et al., 2008), i.e., sequencing constraints expressed as programs, using "if" and "while", hence allowing to express more complex properties.

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

In this section we define a particular problem and propose a solution, which is the main theoretical contribution of this work. We call this problem *Reinforcement Learning for* LTL$_f$/LDL$_f$ *Goals*.

4.6.1 Problem definition

the World, the Agent and the Fluents

Let $\mathcal{W}$ be a *world* of interest (e.g. a room, an environment, a videogame). Let W be the set of *world states*, i.e. the states of the world $\mathcal{W}$. A *feature* is a function f_j that maps a world state to the values of another domain D_j, such as reals, finite enumerations, booleans, etc., i.e., $f_j : W \to D_j$. Given a set of features $F = \langle f_1, \dots, f_d \rangle$, the *feature vector* of a world state w_h is the vector $\mathbf{f}(w_h) = \langle f_1(w_h), \dots, f_d(w_h) \rangle$ of feature values corresponding to w_h.

Now consider an agent that lives in $\mathcal{W}$. The agent can interact with $\mathcal{W}$ by executing an action a taken from a *set of actions* A. Without loss of generality, we assume that such learning agent has a special action *stop* which deems the end of an episode. Moreover, the agent has its own set of features $F_{ag} = \langle f_1, \dots, f_d \rangle$, which yields its *representation of the world* S, where $S \subseteq F_{ag}(W)$ and $F_{ag}(W) = \{\mathbf{f}_{ag}(w) | w \in W\}$. We assume that the agent has a *clock* which determines the granularity of its acting. At every clock, the agent can do action a and observe both the new state s from the new world state w', namely $s = \mathbf{f}_{ag}(w)$, and a real-valued reward $R(s, a, s')$, that depends only from the transition $s \to_a s'$. Finally, we assume that the the law that determines the possible next state s' given the history of a trajectory (i.e. the state transition function T) is Markovian, hence it depends only from the current state s and the taken action a. In other words, we can define an MDP $\mathcal{M}_{ag} = \langle S, A, T, R, \gamma \rangle$.

We consider arbitrary LTL$_f$/LDL$_f$ formulas φ_i ($i = 1, \ldots, m$) over a set of fluents $\mathcal{F}$ used for provide a high-level description of the world. We denote by $\mathcal{L} = 2^{\mathcal{F}}$ the set of possible fluents configurations. Given a set of feature F_{goal}, a *configuration of fluents* $\ell_h \in \mathcal{L}$ is formed by the components that assign truth values to the fluents according to the feature vector $\mathbf{f}_{goal}(w_h)$. At every step, the features for fluents evaluations are observed, obtaining a particular configuration $\ell \in \mathcal{L}$. Notice that in general the features for the fluents and for the agent state space may differ. The formula φ_i is selecting sequences of fluents configurations $\ell_1, \cdots, \ell_n$, with $\ell_k \in \mathcal{L}$, whose relationship with the sequences of states $s_1, \ldots, s_n$, with $s_k \in S$ is unknown.

In other words, a subset of features are used to describe agent states s_h and another subset (for simplicity, assumed disjoint from the previous one) are used to evaluate the fluents in ℓ_h. Hence, given a sequence $w_1, \ldots, w_n$ of world states we get the corresponding sequence of sequences learning agent states $s_1, \ldots, s_n$ and simultaneously the sequence of fluent configurations $\ell_1, \ldots, \ell_n$. Notice that we do not have a formalization for $w_1, \ldots, w_n$ but we do have that for $s_1, \ldots, s_n$ and for $\ell_1, \ldots, \ell_n$.

Oversimplifying, we may say that S is the set of configurations of the low-level features for the learning agent, while $\mathcal{L}$ is the set of configuration of the high-level features needed for expressing φ_i.

Markovian assumption of the state transition function

Now we make the following assumption: that is, the agent actions in A induce a transition distribution over the features and fluents configuration, i.e.,

$$T^g_{ag} : S \times \mathcal{L} \times A \to Prob(S \times \mathcal{L}) \tag{4.18}$$

This means that the state transition function T^g_{ag} is *Markovian*, i.e. the probability to end in the next state s' with the next fluents configuration ℓ' depends only from s, ℓ and a (the current agent state, the current fluents configuration and the action taken, respectively).

Such a transition distribution together with the initial values of the fluents ℓ_0 and of the agent state s_0 allow us to describe a probabilistic transition system accounting for the dynamics of the fluents and agent states. In other words, in response to an agent action a_h performed in the current state w_h (in the state s_h of the agent and the configuration ℓ_h of the fluents), the world changes into w_{h+1} from which s_{h+1} and ℓ_{h+1}. This is all we need to proceed.

Notice that we do not assume nothing about the primitive state transitions function over fluents, i.e. the one induced over the set of possible fluents configurations $\mathcal{L}$. However, it might be the case that the Markovian assumption of the agent state transition function holds, i.e. the probability distribution for the next state ℓ' is fully determined by the current state ℓ and the action a. In that case, the considerations presented in this section still holds, since it is a special case of our general assumptions. We only require that the *joint* transition function in Equation 4.18 satisfies the Markov property.

We are interested in devising policies for the learning agent such that at the end of the episode, i.e., when the agent executes *stop*, the LTL$_f$/LDL$_f$ goal formulas φ_i ($i = 1, \ldots, m$) are satisfied. Now we can state our problem formally.

Definition 4.6. *We define* RL for LTL$_f$/LDL$_f$ goals*, denoted as*

$$\mathcal{M}_{ag}^{goal} = \langle S, A, R, \mathcal{L}, T_{ag}^{g}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$$

with T_{ag}^{g}, R and r_i unknown,the following problem: given a learning agent $\mathcal{M}_{ag} = \langle S, A, T, R \rangle$, with T and R unknown and a set $\{(\varphi_i, r_i)\}_{i=1}^{m}$ of LTL$_f$/LDL$_f$ *formulas with associated rewards, find a (non-Markovian) policy $\bar{\rho} : S^* \to A$ that is optimal wrt the sum of the rewards r_i and R.*

Observe that an optimal policy for our problem, although not depending on $\mathcal{L}$, is guaranteed to satisfy the LTL$_f$/LDL$_f$ goal formulas.

4.6.2 Examples

——————————————-TODO

4.6.3 Reduction to MDP

To devise a solution technique, we start by transforming $\mathcal{M}_{ag}^{goal} = \langle S, A, T_{ag}^{g}, R, \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ into an NMRDP $\mathcal{M}_{ag}^{nmr} = \langle S \times \mathcal{L}, A, T_{ag}^{g}, \{(\varphi_i', r_i)\}_{i=1}^{m} \cup \{(\varphi_s, R(s, a, s'))\}_{s \in S, a \in A, s' \in S} \rangle$ where:

- States are pairs (s, ℓ) formed by an agent configuration s and a fluents configuration ℓ.
- $\varphi_i' = \varphi_i \wedge \Diamond Done$.
- $\varphi_s = \Diamond(s \wedge a \wedge \circ(Last \wedge s'))$.
- T_{ag}^{g}, r_i and $R(s, a, s')$ are unknown and sampled from the environment.

Formulas φ_i' simply require to evaluate the corresponding goal formula φ_i after having done the action *stop*, which sets the fluent *Done* to true and ends the episode. Hence it gives the reward associated to the goal at the end of the episode. The formulas $\Diamond(s \wedge a \wedge \circ(Last \wedge s'))$, one per (s, a, s'), requires both states s and action a are followed by s' are evaluated at the end of the current (partial) trace (notice the use of *Last*). In this case, the reward $R(s, a, s')$ from $\mathcal{M}_{ag}$ associated with (s, a, s') is given.

Notice that policies for $\mathcal{M}_{ag}^{nmr}$ have the form $(S \times \mathcal{L})^* \to A$ which needs to be restricted to have the form required by our problem $\mathcal{M}_{ag}^{goal}$.

A policy $\bar{\rho} : (S \times \mathcal{L})^* \to A$ *has the form* $S^* \to A$ when for any sequence of n states $\langle s_1 \cdots s_n \rangle$, we have that for any pair of sequences of fluent configurations $\langle \ell_1' \cdots \ell_n' \rangle$, $\langle \ell_1'' \cdots \ell_n'' \rangle$ the policy returns the same action, $\bar{\rho}(\langle s_1, \ell_1' \rangle \cdots \langle s_n, \ell_n' \rangle) = \bar{\rho}(\langle s_1, \ell_1'' \rangle \cdots \langle s_n, \ell_n'' \rangle)$. In other words, a policy $\bar{\rho} : (S \times \mathcal{L})^* \to A$ has the form $\bar{\rho} : S^* \to A$ when it does not depend on the fluents $\mathcal{L}$. We can now state the following result.

Theorem 4.3. *RL for* LTL$_f$/LDL$_f$ *goals $\mathcal{M}_{ag}^{goal} = \langle S, A, Tr_{ag}^{g}, R, \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ can be reduced to RL over the NRMDP $\mathcal{M}_{ag}^{nmr} = \langle S \times \mathcal{L}, A, T_{ag}^{g}, \{(\varphi_i', r_i)\}_{i=1}^{m} \cup \{(\varphi_s, R(s, a, s'))\}_{s \in S, a \in A, s' \in S} \rangle$, restricting policies to be learned to have the form $S^* \to A$.*

Observe that by restricting $\mathcal{M}_{ag}^{nmr}$ policies to S^* in general we may discard policies that have a better reward but depend on $\mathcal{L}$. On the other hand, these policies need to change the learning agent in order to allow it to observe $\mathcal{L}$ as well. As mentioned in the introduction, we are interested in keeping the learning agent as it is, apart for additional memory.

As a second step, we apply the construction of Section 4.5 and obtain a new MDP learning agent. In such construction, however, because of the triviality of their automata, we do not need to keep track of state φ_s, but just give the reward $R(s, a, s')$ associated to (s, a, s'). Instead we do need to keep track of state of the DFAS $\mathcal{A}_{\varphi_i}$ corresponding to the formulas φ'_i. Hence, from $\mathcal{M}_{ag}^{nmr}$, we get an MDP $\mathcal{M}'_{ag} = \langle S', A', Tr'_{ag}, R' \rangle$ where:

- $S' = Q_1 \times \cdots \times Q_m \times S \times \mathcal{L}$ is the set of states;
- $Tr'_{ag} : S' \times A' \times S' \to [0, 1]$ is defined as follows:

$$Tr'_{ag}(q_1, \ldots, q_m, s, \ell, a, q'_1, \ldots, q'_m, s', \ell') = \begin{cases} Tr(s, \ell, , a, s', \ell') & \text{if } \forall i : \delta_i(q_i, \ell') = q'_i \\ 0 & \text{otherwise;} \end{cases}$$

- $R' : S' \times A \times S' \to \mathbb{R}$ is defined as:

$$R'(q_1, \ldots, q_m, s, \ell, a, q'_1, \ldots, q'_m, s', \ell') = \textstyle\sum_{i:q'_i \in F_i} r_i + R(s, a, s')$$

Finally we observe that the environment gives now both the rewards $R(s, a, s')$ of the original learning agent, and the rewards r_i associated to the formula so has to guide the agent towards the satisfaction of the goal (progressing correctly the DFAS $\mathcal{A}_{\varphi_i}$).

By applying Theorem 4.2 we get that NMRDP $\mathcal{M}_{ag}^{nmr}$ and the MDP $\mathcal{M}'_{ag}$ are equivalent, i.e., any policy of $\mathcal{M}_{ag}^{nmr}$ has an equivalent policy (hence guaranteeing the same reward) in $\mathcal{M}'_{ag}$ and vice versa. Hence we can learn policy on $\mathcal{M}'_{ag}$ instead of $\mathcal{M}_{ag}^{nmr}$.

We can refine Theorem 4.3 into the following one.

Theorem 4.4. *RL for LTL$_f$/LDL$_f$ goals $\mathcal{M}_{ag}^{goal} = \langle S, A, T_{ag}^g, R, \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$ can be reduced to RL over the MDP $\mathcal{M}'_{ag} = \langle S', A, T'_{ag}, R' \rangle$, restricting policies to be learned to have the form $Q_1 \times \ldots \times Q_n \times S \to A$.*

As before, a policy $Q_1 \times \ldots \times Q_n \times S \times \mathcal{L} \to A$ *has the form* $Q_1 \times \ldots \times Q_n \times S \to A$ when any ℓ and ℓ' the policy returns the same action, $\rho(q_1, \ldots, q_n s, \ell) = \rho(q_1, \ldots, q_n s, \ell')$.

The final step is to solve our original RL task on $\mathcal{M}_{ag}^{goal}$ by performing RL on a new MDP $\mathcal{M}_{ag}^{new} = \langle Q_1 \times \cdots \times Q_m \times S, A, T''_{ag}, R'' \rangle$ where:

- Transitions distribution T''_{ag} is the marginalization wrt $\mathcal{L}$ of T'_{ag} and is unknown;
- Rewards R'' is defined as:

$$R''(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \sum_{i:q'_i \in F_i} r_i + R(s, a, s').$$

- States q_i of DFAs $\mathcal{A}_{\varphi_i}$ are progressed correctly by the environment.

Indeed we can show the following result.

Theorem 4.5. *RL for* LTL$_f$/LDL$_f$ *goals* $\mathcal{M}_{ag}^{goal} = \langle S, A, T', R, \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$ *can be reduced to RL over the MDP* $\mathcal{M}_{ag}^{new} = \langle Q_1 \times \dots \times Q_m \times S, A, T''_{ag}, R'' \rangle$ *and the optimal policy* ρ_{ag}^{new} *learned for* $\mathcal{M}_{ag}^{new}$ *can be reduced to a corresponding optimal policy for* $\mathcal{M}_{ag}^{goal}$.

Proof. From Theorem 4.4, by the following observations. For the sake of brevity, we use $\mathbf{q}$ to denote $q_1, \dots, q_m$. Notice also that for all $\ell, \ell' \in \mathcal{L}$, $R'(\mathbf{q}, s, \ell, a, \mathbf{q}', s', \ell') = R''(\mathbf{q}, s, a, \mathbf{q}', s')$.

We show that the values of $v_{ag}^{\rho}(q_1, \dots, q_m, s, \ell)$, i.e. the state value function for $\mathcal{M}'_{ag}$(for simplicity v^{ρ}, unless otherwise stated), for some policy ρ, do not depend on ℓ or, in other words, it is necessary that $\forall \ell_1, \ell_2 . v^{\rho}(q_1, \dots, q_m, s, \ell_1) = v^{\rho}(q_1, \dots, q_m, s, \ell_2)$. Finally, we notice that $\forall \ell . v_{ag}^{\rho,new} = v_{ag}^{\rho}$

From Equation 4.4 we have:

$$
\begin{aligned}
& v_{\rho}(\mathbf{q}, s, \ell) = \\
& \sum_{\mathbf{q}', s', \ell'} P(\mathbf{q}', s', \ell' | \mathbf{q}, s, \ell, a)[R'(\mathbf{q}, s, \ell, a, \mathbf{q}', s', \ell') + \gamma v_{\rho}(\mathbf{q}', s', \ell')] = \\
& \sum_{\mathbf{q}', s', \ell'} P(\mathbf{q}', s', \ell' | \mathbf{q}, s, \ell, a)[R''(\mathbf{q}, s, a, \mathbf{q}', s') + \gamma v_{\rho}(\mathbf{q}', s', \ell')]
\end{aligned}
\tag{4.19}
$$

Using the equivalence between R' and R'', as already pointed out. Notice that we can compute $\mathbf{q}'$ from $\mathbf{q}$ and ℓ', hence we do not need ℓ. In other words:

$$P(\mathbf{q}', s', \ell' | \mathbf{q}, s, \ell, a) = P(\mathbf{q}', s', \ell' | \mathbf{q}, s, a)$$

Equation 4.19 becomes:

$$\sum_{\mathbf{q}', s', \ell'} P(\mathbf{q}', s', \ell' | \mathbf{q}, s, a)[R''(\mathbf{q}, s, a, \mathbf{q}', s') + \gamma v_{\rho}(\mathbf{q}', s', \ell')] \tag{4.20}$$

At this point, we see that v^{ρ} does not depend from ℓ, hence we can safely drop ℓ as argument for v_{ρ}, obtaining v_{ag}^{ρ}. Indeed, from 4.20:

$$
\begin{aligned}
& \sum_{\mathbf{q}', s'} [R''(\mathbf{q}, s, a, \mathbf{q}', s') + \gamma v_{ag}^{\rho,new}(\mathbf{q}, s)] \sum_{\ell'} P(\mathbf{q}', s', \ell' | \mathbf{q}, s, a) = \\
& \sum_{\mathbf{q}', s'} P(\mathbf{q}', s' | \mathbf{q}, s, a)[R''(\mathbf{q}, s, a, \mathbf{q}', s') + \gamma v_{ag}^{\rho}(\mathbf{q}', s')] = \\
& v_{ag}^{\rho}(\mathbf{q}, s)
\end{aligned}
\tag{4.21}
$$

Where in 4.21 we marginalized the distribution $P(\mathbf{q}', s', \ell' | \mathbf{q}, s, a)$ over ℓ'. From Definition 4.1 of optimal policy, we can reduce an optimal policy ρ_{ag}^{new} to a policy of the form $\rho'_{ag} : Q_1 \times \dots \times Q_m \times S \to A$ that is optimal for $\mathcal{M}'_{ag}$ (since the state value function of $\mathcal{M}_{ag}^{new}$, after dropping the argument ℓ, and of $\mathcal{M}'_{ag}$ are equivalent). From Theorem 4.4 the thesis. $\square$

It is worth to remark that in the resulting MDP $\mathcal{M}_{ag}^{new}$ the *explicit presence of the fluents configuration* ℓ *has been removed.* Rather, the dependency is compiled into the expanded state space $Q_1 \times \dots Q_m \times S$, where $Q_1, \dots, Q_m$ are the automata state spaces associated to the formulas φ_i.

4.6.4 An episodic goal-based view

Here we clarify how the actual transition model works in the final MDP $\mathcal{M}_{ag}^{new}$. We focus on the episodic view, where the learning process is organized in episodes. Recall that the state space of $\mathcal{M}_{ag}^{new}$ is $Q_1 \times \dots Q_m \times S$, where Q_i is the set of states of $\mathcal{A}_{\varphi_i}$, the automaton associated to the LTL$_f$/LDL$_f$ formula φ_i (see Section 2.6). Moreover we consider two cases: when there exists a subset of states $S_{goal} \subseteq S$ that we call *goal states*, such that every transition in those state make the task completed and hence the end of the episode; and when there are no goal states, but the task is to maximize the obtained reward, until a maximum number of steps is reached. E.g. in Gridworld presented in Example 4.1, we can defined s_{34} as a goal state. However nothing prevent us to set a maximum number of time steps, and let the agent learn how to collect reward as much as possible in a limited amount of time.

Now we give the following definition:

Definition 4.7. Let $\mathcal{M}_{ag}^{goal}$ be our problem and $\mathcal{M}_{ag}^{new}$ its transformation into and MDP, as defined in Theorem 4.5 and let $\mathbf{s} = \langle q_1, \dots, q_m, s\rangle \in Q_1 \times Q_1 \times \dots Q_m \times S$. Given an observation $s' \in S$ and $\ell' \in \mathcal{L}$, we define the *successor state of* $\mathbf{s}$ as $\mathbf{s}' = \langle q_1', \dots, q_m', s'\rangle$, where $q_i' = \delta_i(q_i, \ell')$.

A reinforcement learning episode in this setting works as follows. Assume that the MDP has a dummy initial state s_{init} and a dummy action *start*, as defined for NMRDPs in 4.17. In this construction, the dummy initial state in the expanded state space is $(q_{0,0}, q_{1,0}, \dots, q_{m,0}, s_{init})$. The first action taken by the agent is *start*, which allow the agent to observe the true initial world state w_0. From w_0, the agent extract the features to determine the state $s_0 \in S$ and the fluents configuration $\ell_0 \in \mathcal{L}$ [1]. The successor state is $(q_0', q_1', \dots, q_m', s_0)$. Then the agent might take a new action, observe another world state w', extract $s' \in S$ and $\ell' \in \mathcal{L}$, and compute the q_i as before. The sequence reiterates until the end of the episode.

Consider a generic state $\mathbf{s} = \langle q_1, \dots, q_m, s\rangle \in Q_1 \times Q_1 \times \dots Q_m \times S$ and a transition to $\mathbf{s}' = \langle q_1', \dots, q_m', s'\rangle$. For each new state q_i' of automaton $\mathcal{A}_{\varphi_i}$, the following might happen:

1. $q_i' = q_i$, i.e. the state is not changed;
2. $q_i' \neq q_i$, and from q_i' it is still possible to reach a final state;
3. $q_i' \neq q_i$, and from q_i' any final state cannot be reached;

If, for some $\mathcal{A}_{\varphi_i}$, we are in case 3, then the constraint specified by φ_i is violated, hence we call $\mathbf{s}'$ a *failure state*. We say that $\mathbf{s}$ is a *goal state* if $\forall i.q_i \in F_i$, i.e. every current state q_i is in an accepting state of the automaton $\mathcal{A}_{\varphi_i}$. Instead, if the underlying MDP is a goal-based one (e.g. $S_{goal} \neq \emptyset$) then $\mathbf{s}$ is a goal state if $s \in S_{goal}$.

Let $\mathbf{s}$ the last state of the trace of the current episode. The *stopping condition*, i.e. the condition that determines the end of the episode, depending from $\mathbf{s}$, is:

$$\text{failure\_state}(\mathbf{s}) \vee \text{goal\_state}(\mathbf{s}) \vee \text{exceeded\_time\_limit}$$

The reward $R(s, a, s')$ is collected after each taken action, and it is summed with r_i for every satisfied φ_i at the last state of the episode.

[1]Observe that the first transition is "artificial", and in many cases, both s_0 and ℓ_0 are known to the experimenter. The explanation presented here is aimed to clarify the underlying mathematical construction.

Summary

In the following we summarize the results of this section:

- We defined a new problem: *Reinforcement Learning for* LTL$_f$/LDL$_f$ *Goals* $\mathcal{M}_{ag}^{goal}$ (Definition 4.6). In a nutshell, from an existing MDP we introduced temporal goals defined by LTL$_f$/LDL$_f$ formulas about fluents $\mathcal{L}$ observed from the world. A solution for this problem is a (non-Markovian) policy that is optimal in terms of rewards and such that satisfies the LTL$_f$/LDL$_f$ specifications.

- We gave an equivalent formulation of $\mathcal{M}_{ag}^{goal}$, the NMRDP $\mathcal{M}_{ag}^{nmr}$, and observe that the original problem can be reduced to this formulation (Theorem 4.3).

- We applied the construction shown in Section 4.5, yielding $\mathcal{M}'_{ag}$, and by using Theorem 4.2 we stated Theorem 4.4, showing that $\mathcal{M}_{ag}^{goal}$ can be reduced to $\mathcal{M}'_{ag}$

- Finally, by proving Theorem 4.5, we showed that we can do reinforcement learning over the equivalent MDP $\mathcal{M}_{ag}^{new}$ by simply dropping the fluents ℓ from the state space. The optimal policy for $\mathcal{M}_{ag}^{new}$ can be transformed to a solution for the original problem $\mathcal{M}_{ag}^{goal}$.

4.7 Conclusions

In this chapter we introduced the topic of Reinforcement Learning, as well as foundational definitions (MDP, policy ρ, state-value function v_ρ) and algorithms (Q-Learning, Sarsa). Then we presented the notion of NMRDP and a technique to built an equivalent MDP, by specifying non-Markovian reward function with LTL$_f$/LDL$_f$ formalisms. Finally, we defined and studied a new problem, *Reinforcement Learning for* LTL$_f$/LDL$_f$ *goals*, and proposed a reduction that yields an equivalent MDP which can be used to solve the original problem. An episodic goal-based view of the final MDP is provided.

Chapter 5

Automata-based Reward shaping

In this chapter we discuss a method to improve exploration of the state space and improve the convergence rate in the setting studied in Section 4.6, in particular in the construction $\mathcal{M}_{ag}^{new}$. Indeed, the state space of the original MDP $\mathcal{M}_{ag}$ is expanded in order to implicitly label the states with relevant histories for the satisfaction of LTL$_f$/LDL$_f$ formulas, which in general makes harder to learn an optimal policy in $\mathcal{M}_{ag}^{new}$ wrt $\mathcal{M}_{ag}$, due to a bigger state space. Moreover, the introduction of temporal goals makes things harder, because the agent has to find a proper behavior that satisfies all the goals. In general, proper behaviors are harder to find, and require more exploration of the state space.

Reward Shaping is a general method, well-known in the literature of Reinforcement Learning, used to deal with big state spaces and *sparse* rewards, and trying to address the temporal credit assignment problem, i.e. to determine the long-term consequences of actions. It consists in provide additional reward to the learning agent. In this chapter we propose a technique to apply reward shaping in our setting.

The chapter is structured as follows: in the first section we explain the reward shaping theory, in particular the requirements for theoretical guarantees of policy invariance under reward transformation. Then we show how apply reward shaping on the automata transitions associated to LTL$_f$/LDL$_f$ formulas φ_i, both in *off-line* variant (i.e. when the automaton is built *before* the beginning of the learning process) and *on-the-fly* variant (i.e. when the automaton is built *during* the learning process).

5.1 Reward Shaping Theory

Reward shaping is a well-known technique to guide the agent during the learning process and so reduce the time needed to learn. The idea is to supply additional rewards in a proper manner such that the optimal policy is the same of the original MDP.

More formally, consider as example the temporal difference in SARSA after a transition $s \rightarrow_a s'$, presented in Equation 4.10:

$$\delta = R(s, a, s') + \gamma Q(s', a') - Q(s, a) \tag{5.1}$$

Reward shaping consists in defining the *shaping function* $F(s, a, s')$ and sum it to the environment reward $R(s, a, s')$, namely:

$$\delta = R(s, a, s') + F(s, a, s') + \gamma Q(s', a') - Q(s, a) \tag{5.2}$$

In the following sections we will discuss two way to define $F(s, a, s')$.

5.1.1 Potential-Based Reward Shaping

We give the definition of *potential-based shaping function* (PBRS).

Definition 5.1 (Ng et al. (1999)). Let any S, A, γ and any shaping function $F : S \times A \to \mathbb{R}$ be given. We say F is a potential-based shaping function if there exists a real-valued function $\Phi : S \to \mathbb{R}$ such that for all $s \in S, a \in A, s' \in S$

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{5.3}$$

Notice that in Equation 5.1 the action does not affect the value of $F(s, a, s')$, hence sometime we write $F(s, s')$.

In (Ng et al., 1999) it has been shown the following theorem:

Theorem 5.1 (Ng et al. (1999)). *Given an MDP $\mathcal{M} = \langle S, A, T, R, \gamma\rangle$ and a potential based shaping function F (Definition 5.1). Then, consider the MDP $\mathcal{M}' = \langle S, A, T, R{+}F, \gamma\rangle$, i.e. the same of $\mathcal{M}$ but applying reward shaping. Then, the fact that F is a potential-based reward shaping function is a necessary and sufficient condition to guarantee consistency with the optimal policy. In particular:*

- *(Sufficiency) if F is a potential-based shaping function, then every optimal policy in $\mathcal{M}'$ is optimal in $\mathcal{M}$.*
- *(Necessity) if F is not a potential-based shaping function (e.g. no such Φ exists satisfying 5.3), then there exists T and R such that no optimal policy in $\mathcal{M}'$ is optimal in $\mathcal{M}$.*

In poor words, potential-based reward shaping of the form $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$, for some $\Phi : S \to \mathbb{R}$, is a necessary and sufficient condition for policy invariance under this kind of reward transformation, i.e. the optimal and near-optimal solutions of $\mathcal{M}$ are preserved when considering $\mathcal{M}'$.

5.1.2 Dynamic Potential-Based Reward Shaping

A limitation of PBRS is that the potential of a state does not change dynamically during the learning. This assumption often is broken, especially if the reward-shaping function is generated automatically.

Equation 5.3 can be extended to include also the time as parameter of the potential function Φ, while guaranteeing policy invariance. Formally:

$$F(s, t, a, s', t') = \gamma\Phi(s', t') - \Phi(s, t) \tag{5.4}$$

where t and t' are respectively the time when visiting s and s'. In this case, we call this technique *dynamic potential-based reward shaping* (DPBRS).

The shaping function in the form 5.4 guarantees policy invariance, as shown in Theorem 5.1. To show why this is the case, consider the expected discounted return for an infinite sequence of states (Definition 4.1):

$$G = \sum_{k=0}^{\infty} \gamma^k R_k \tag{5.5}$$

If we apply dynamic potential-based reward shaping to Equation 5.5 we have:

$$\begin{aligned}
G_\Phi &= \sum_{k=0}^{\infty} \gamma^k (R_k + F(s_k, t_k, s_{k+1}, t_{k+1})) \\
&= \sum_{k=0}^{\infty} \gamma^k (R_k + \gamma\Phi(s_{k+1}, t_{k+1}) - \Phi(s_k, t_k)) \\
&= \sum_{k=0}^{\infty} \gamma^k R_k + \sum_{k=0}^{\infty} \gamma\Phi(s_{k+1}, t_{k+1}) - \sum_{k=0}^{\infty} \Phi(s_k, t_k) \\
&= G + \sum_{k=1}^{\infty} \gamma\Phi(s_k, t_k) - \sum_{k=1}^{\infty} \Phi(s_k, t_k)) - \Phi(s_0, t_0) \\
&= G - \Phi(s_0, t_0)
\end{aligned}$$

Hence, any expected reward with reward shaping is the same of the one without reward shaping but a negative shift equal to $\Phi(s_0, t_0)$, i.e. a constant that does not depend from the actions taken. This means that the policy cannot be affected by the shaping function.

5.1.3 Relevant considerations about PBRS

Here we talk about some issues in PBRS described in Section 5.1.1 (analogous considerations can be made for DPBRS described in Section 5.1.2), described in (Grzes and Kudenko, 2009; Grzes, 2010; Grześ, 2017). In particular, we focus on:

- the presence of the discount factor γ in Equation 5.3;
- the value of $\Phi(s)$ when s is a terminal state.

The discount factor γ

In order to guarantee policy invariance, γ in Equation 5.3 must be equal to the discount factor of the MDP. Observe that, in general, this does not imply a *speed-up in learning time*. Indeed in (Grzes, 2010; Grzes and Kudenko, 2009) several issues of PBRS have been described, when $\gamma < 1$, that worsen the learning. In particular, it might happen that for some chosen value of $\Phi(s)$, the shaping function does not give a meaningful reward signal, e.g. near to the goal state, instead of a positive reward, a negative one signal is given to the learner, which is obviously a counterproductive choice.

In (Grzes, 2010) has been proposed an alternative approach to PBRS, which simply sets $\gamma = 1$ in Equation 5.3, namely:

$$F(s, a, s') = \Phi(s') - \Phi(s) \tag{5.6}$$

It has be proven that this approach *does not guarantee policy invariance*, i.e. the policy learned over the MDP with reward shaping in general is not equivalent to the original MDP. In the same work, it has been shown experimental evidence of the goodness of the new approach, even in the pathological cases with $\gamma < 1$. So using PBRS with $\gamma_{rs} = 1$, even if the discount factor of the MDP $\gamma_{mdp} \neq 1$, "works", although the invariance of the policy is not guaranteed anymore.

The value of terminal state $\Phi(s)$

In (Pratt, 1976a) has been explained that the potential function in any terminal state (i.e. in any state where the episode terminates), must be 0 in order to guarantee policy invariance.

In Figure 5.1 is depicted a particular scenario in which the violation of this requirement over potential-based reward shaped learning leads to a different policy than non-shaped learning. In particular, without reward shaping the optimal policy from s_i would choose g_2 instead of g_1, since $r_{g_2} = 100 > r_{g_1} = 0$. However, after applying reward shaping, the reward for the transition $s_i \rightarrow_{a_1} g_1$ is $r_{g_1} = 1000$, which is higher than the one from $s_i \rightarrow_{a_1} g_2$, which is $r_{g_2} = 110$. This time, the optimal policy should prefer the transition towards g_1, although the *true* optimal policy (i.e. with no reward shaping) should prefer the transition towards g_2.

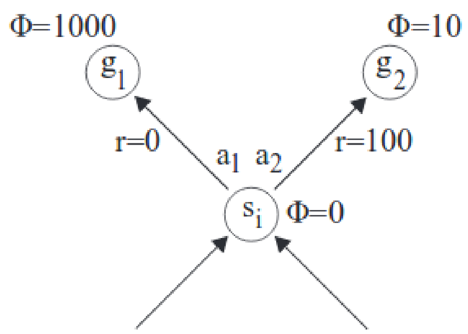


Figure 5.1. An example that shows why the potential function over terminal state must be 0.

More formally, consider the return of the sequence $\bar{s}$, similarly as Equation 4.1:

$$G(\bar{s}) \coloneqq \sum_{k=0}^{N-1} \gamma^k R(s_k, s_{k+1}) \tag{5.7}$$

If we apply PBRS, it becomes:

$$\begin{aligned} G_\Phi(\bar{s}) &= \sum_{k=0}^{N-1} \gamma^k (R(s_k, s_{k+1}) + F(s_k, s_{k+1})) \\ &= \sum_{k=0}^{N-1} \gamma^k (R(s_k, s_{k+1}) + \gamma\Phi(s_{k+1}) - \Phi(s_k)) \\ &= \sum_{k=0}^{N-1} \gamma^k R(s_k, s_{k+1}) + \sum_{k=1}^{N} \gamma^k \Phi(s_k) - \sum_{k=0}^{N-1} \gamma^k \Phi(s_k) \\ &= G(\bar{s}) + \sum_{k=1}^{N-1} \gamma^k \Phi(s_k) + \gamma^N \Phi(s_N) - \sum_{k=1}^{N-1} \gamma^k \Phi(s_k) - \Phi(s_0) \\ &= G(\bar{s}) + \gamma^N \Phi(s_N) - \Phi(s_0) \end{aligned} \tag{5.8}$$

The term $\Phi(s_0)$ cannot alter the policy since does not depend from any action executed; on the other hand, the term $\gamma^N \Phi(s_N)$ depends on actions, since the terminal state depends from the previous transitions, hence this term can modify the optimal policy. This happens whenever $\Phi(s_N) \neq 0$, where s_N is any state in which an episode ends, namely goal states, failure states and states where the episode ends due to the time limit exceeded.

A simple solution to this problem is to require that $\Phi(s_N) = 0$ whenever the reinforcement learning trajectory is terminated at state s_N. Notice that if in other trajectories the same s_N is visited, $\Phi(s_N)$ might be different from 0. The only requirement is that if s is a terminal state, then $\Phi(s) = 0$.

Example 5.1. Recalling Example 4.1, we can apply PBRS by defining a potential function that measures how far the agent is from the goal. As heuristic, we can use the *Manhattan distance* between the current position and the goal state. More formally, considering s_{34} the goal state and s_{ij} the current state, we define:

$$\Phi(s) = -[(3 - i) + (4 - j)]$$

It is easy to see that the nearer the agent to the goal state, the higher the potential function evaluated in the state of the agent. For instance, if the current state is s_{11}, $\Phi(s_{11}) = -5$, whereas in s_{33} (which is nearer to the goal), $\Phi(s_{33}) = -1$. With this definition, transition from s_{11} to s_{12} (which makes the agent closer to the goal) evaluates $F(s_{11}, s_{12}) = \Phi(s_{12}) - \Phi(s_{11}) = -4 - (-5) = +1$, whereas for transition $s_{33} \to s_{32}$ (which makes the agent more distant to the goal), $F(s_{33}, s_{32}) = (-2) - (-1) = -1$.

In the following sections, we will take into account the topics just described in designing a reward shaping strategy for our setting.

5.2 Off-line Reward shaping over $\mathcal{A}_\varphi$

In this part we propose an automatic way to apply reward shaping in the setting presented in Section 4.6. Recall that the state space of $\mathcal{M}_{ag}^{new}$ is $Q_1 \times \ldots Q_m \times S$, where Q_i is the set of states of $\mathcal{A}_{\varphi_i}$, the automaton associated to the LTL$_f$/LDL$_f$ formula φ_i (see Section 2.6).

The basic intuition about our approach is that *every step toward the satisfaction of a goal formula φ_i should be rewarded*, analogously as it is done in reward shaping for classical goals (see Example 5.1). Hence, for a given temporal specification φ_i we should assign to every $q \in Q_i$ a potential function that is, to some extent, inversely proportional to the distance from any final state.

Given a (minimal) automaton $\mathcal{A}_\varphi$ from a LTL$_f$/LDL$_f$ formula φ and its associated reward r, Algorithm 5.1 shows how the potential function is build from $\mathcal{A}_\varphi$. This operation is made *off-line*, i.e. before the learning process. Then we associate automatically to the states of the DFA a potential function $\Phi(q)$ whose value decreases proportionally with the minimum distance between the automaton state q and any accepting state. By construction, potential-based reward shaping with this definition of the potential function gives a positive reward when the agent performs an action leading to a q' that is one step closer to an accepting state, and a negative one in the opposite case. Notice that, by construction, $G(\bar{s}) = G_\Phi(\bar{s})$, where G_Φ is defined in Equation 5.8. Indeed, $\gamma^N \Phi(s_N) = 0$ because we take into account the issue explained in Section 5.1.3, and $\Phi(s_0) = 0$ by construction of the Algorithm 5.1.

In the actual implementation of the Algorithm 5.1, $\Phi(q)$ can be computed by least-fix point over the automaton $\mathcal{A}_\varphi$, i.e. starting from the accepting states and then explore the states from the nearest to the farthest ones.

Algorithm 5.1. Static Reward Shaping over $\mathcal{A}_\varphi$

input: (minimal) automaton $\mathcal{A}_\varphi$, reward r
output: potential function $\Phi : Q \to \mathbb{R}$
Let *sink* be the sink state after the completion of $\mathcal{A}_\varphi$
Let n_{q_0} be minimum number of hops to reach an accepting state from q_0
for $q \in Q$ **do**:
  Let n_q be minimum number of hops to reach an accepting state from q
  if $n_{q_0} \neq 0$ **then** ▷ i.e. if q_0 is NOT an accepting state
    $\Phi(q) \leftarrow \frac{n_{q_0} - n_q}{n_{q_0}} \cdot r$
  else
    $\Phi(q) \leftarrow (n_{q_0} - n_q) \cdot r$
  end if
end for
Let n_{max} the maximum number of hops to reach an accepting state
$\Phi(sink) \leftarrow \frac{n_{q_0} - n_{max}}{n_{q_0}} \cdot r$ **if** $n_{q_0} \neq 0$ **else** $(n_{q_0} - n_{max}) \cdot r$
return Φ

5.3 On-The-Fly Reward shaping

Reward shaping can also be used when the DFAs of the LTL$_f$/LDL$_f$ formulas are constructed *on-the-fly* (Brafman et al., 2018) so as to avoid to compute the entire automaton off-line. To do so we can rely on dynamic reward shaping (see Section 5.1.2). The idea is to build $\mathcal{A}_\varphi$ progressively while learning. During the learning process, at every step, the value of the fluents $\ell \in \mathcal{L}$ is observed and the successor state q' of the current state q of the DFA on-the-fly is computed. Then, the transition and the new state just observed are added into the built automaton at time t, $\mathcal{A}_{\varphi,t}$, yielding $\mathcal{A}_{\varphi,t'}$. The potential function Φ for $\mathcal{A}_{\varphi,t'}$ is recomputed for the new version of the automaton. In this case, the shaping function takes the following form:

$$F(q, t, a, q', t') = \Phi(q', t') - \Phi(q, t) \tag{5.9}$$

i.e. the dynamic reward shaping in Equation 5.4 but taking into account the issue presented in Section 5.1.3 where $\Phi(q, t)$ is a variant of the off-line case, but computed on the automaton $\mathcal{A}_{\varphi,t}$. In the following we explain how Φ in the on-the-fly case differs from the one shown in Section 5.2.

Details about $\Phi(q, t)$

There is an important issue which has not yet been pointed out. In the *off-line* variant described in Section 5.2, we apply reward shaping at every transition by knowing the full automaton $\mathcal{A}_\varphi$; however, in the *on-the-fly variant*, at the beginning of the learning task we have an "empty" automaton, i.e. only the initial state with no transition from it. Let assume that after an action the automaton makes a move from the initial state. How can we assign a positive/negative shaping reward on the transition if we do not know the goodness of the transition? And if during a simulation we discover an accepting state, there could be a nearer accepting state that has not yet been discovered, but in order to reach it we should first discover other intermediate states. How to allow the agent to discover it while not fixing on the only known accepting states?

It is clear that, the farther the learning task goes, the more accurate will be the shaping rewards, because every observed transition of the automaton during the activity of the agent is stored and eventually the entire automaton will be explored. However, some paths ending in an accepting state are not fully explored, so how to encourage the agent, in our setting, to explore those paths?

In order to determine $\Phi(s, t)$ for a given $A_{\varphi,t}$, we consider the same computations of Algorithm 5.1 but *considering the leaves (wrt the initial state) of the automaton as accepting state*. In this way, even if a path does not end in an accepting state, its following is still rewarded. It could be *wrongly rewarded*, since the path might lead to a failure state. However, the dynamic reward shaping theory states that until the potential-based condition is preserved, also the optimal and near-optimal solutions are preserved; moreover, eventually, once the final state of the path is discovered, the reward for that path will assume the right values.

The search for the leaves states is done through Depth-First Search, while the computation is the same of the Algorithm 5.1.

It is easy to see that:

Theorem 5.2. *Automata-based reward shaping, both in off-line and on-the-fly variants, preserves optimality and near-optimality of the MDP solutions.*

Proof. For the off-line case, the shaping-reward function Φ is, by construction, potential based, hence fulfilling the premises of theorems in (Ng et al., 1999) and (Grześ, 2017). Also for the on-the-fly variant, we observe that our construction is compliant with the requirements defined in (Devlin and Kudenko, 2012). □

Chapter 6

RLTG

In this chapter we describe RLTG (Reinforcement Learning for Temporal Goals), a software project written in Python. It is the reference implementation of many of the topics described in Chapter 4.

6.1 Introduction

Main features: FLLOAT is a Python library that provides support for:

- Syntax, semantics and parsing of the following logic formalisms:
  - Propositional Logic;
  - Linear Temporal Logic on Finite Traces LTL_f
  - Linear Dynamic Logic on Finite Traces LDL_f;
- Conversion from $\text{LTL}_f/\text{LDL}_f$ formula to NFA, DFA and DFA On-The-Fly

Dependencies: FLLOAT requires Python>=3.5 and depends on the following packages:

- FLLOAT, described in Chapter 3;
- Gym OpenAI, a toolkit for developing and comparing reinforcement learning algorithms. It offers a useful abstraction of reinforcement learning environments.

Installation: You can find the package on PyPI, hence you can install it with:

```
pip install rltg
```

6.2 Package structure

The package is structured as follows:

6.3 Code examples

6.4 License

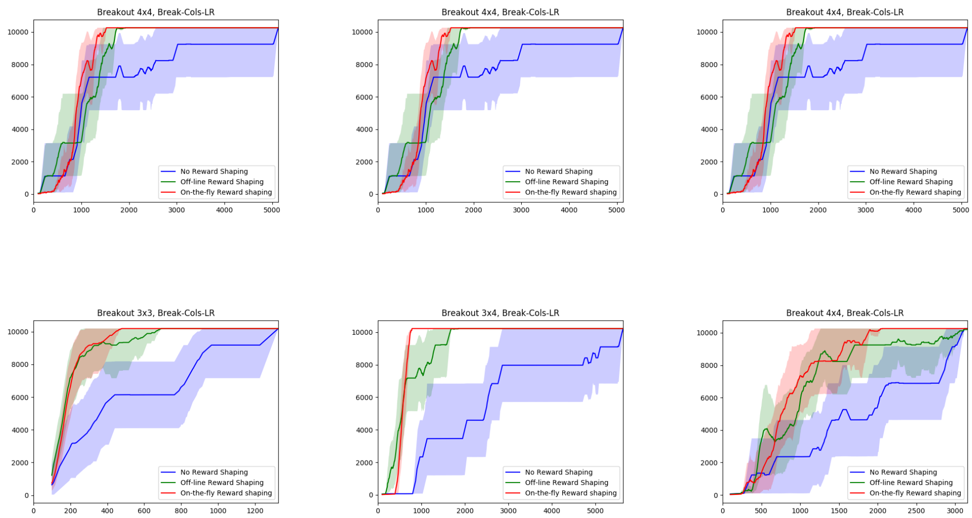
The software is released under MIT license.

Chapter 7

Experiments

look at experiment introduction in Grzes phd thesis

7.1 BREAKOUT



7.2 SAPIENTINO

7.3 MINECRAFT

Chapter 8

Conclusions

Bibliography

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 1160–1167, 1996.

Jorge A. Baier, Christian Fritz, Meghyn Bienvenu, and Sheila A McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1509–1512. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://dl.acm.org/citation.cfm?id=1620270.1620321.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. URL http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false.

Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17342.

Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Specifying non-markovian rewards in mdps using ldl on finite traces (preliminary version). *CoRR*, abs/1706.08100, 2017.

Richard J. Büchi. Weak secondâĂŘorder arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1âĂŘ6):66–92, 1960. doi: 10.1002/malq.19600060105. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/malq.19600060105.

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 Fabio Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Inf. Comput.*, 162(1/2):117–137, October 2000. ISSN 0890-5401. URL http://dl.acm.org/citation.cfm?id=359243.359271.

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.

Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for ltl and ldl on finite traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1558–1564. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL http://dl.acm.org/citation.cfm?id=2832415.2832466.

Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on ltl on finite traces: Insensitivity to infiniteness. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1027–1033. AAAI Press, 2014. URL http://dl.acm.org/citation.cfm?id=2893873.2894033.

Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 433–440, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9817381-1-7, 978-0-9817381-1-6. URL http://dl.acm.org/citation.cfm?id=2343576.2343638.

Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961. ISSN 00029947. URL http://www.jstor.org/stable/1993511.

Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194 – 211, 1979. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(79)90046-1. URL http://www.sciencedirect.com/science/article/pii/0022000079900461.

D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. Technical report, Jerusalem, Israel, Israel, 1997.

James Garson. Modal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2016 edition, 2016.

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.

M. Grzes and D. Kudenko. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *2009 International Conference on Machine Learning and Applications*, pages 337–344, Dec 2009. doi: 10.1109/ICMLA.2009.33.

Marek Grzes. *Improving exploration in reinforcement learning through domain knowledge and parameter analysis*. PhD thesis, University of York, 2010.

Marek Grześ. Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 565–573, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems. URL http://dl.acm.org/citation.cfm?id=3091125.3091208.

John E. Hopcroft, Rajeev Motwani, Rotwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000. ISBN 0201441241.

Bakhadyr Khoussainov and Anil Nerode. *Automata Theory and Its Applications*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 2001. ISBN 3764342072.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the*

Sixteenth International Conference on Machine Learning, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613.

M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 International Conference on Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-38444-8, 978-3-540-38444-1. doi: 10.1007/11837862_18. URL http://dx.doi.org/10.1007/11837862_18.

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.

V. R. Pratt. Semantical consideration on floyo-hoare logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 109–121, Oct 1976a. doi: 10.1109/SFCS.1976.27.

V. R. Pratt. Semantical considerations on floyd-hoare logic. Technical report, Cambridge, MA, USA, 1976b.

Vaughan R. Pratt. A near-optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20(2):231 – 254, 1980. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(80)90061-6. URL http://www.sciencedirect.com/science/article/pii/0022000080900616.

M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959. ISSN 0018-8646. doi: 10.1147/rd.32.0114. URL http://dx.doi.org/10.1147/rd.32.0114.

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.

Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Mach. Learn.*, 22(1-3):123–158, January 1996. ISSN 0885-6125. doi: 10.1007/BF00114726. URL http://dx.doi.org/10.1007/BF00114726.

A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985. ISSN 0004-5411. doi: 10.1145/3828.3837. URL http://doi.acm.org/10.1145/3828.3837.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988. ISSN 1573-0565. doi: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

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

Wolfgang Thomas. Star-free regular sets of ÏĽ-sequences. *Information and Control*, 42(2):148 – 156, 1979. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(79)90629-6. URL http://www.sciencedirect.com/science/article/pii/S0019995879906296.

B.A. Trakhtenbrot. Finite automata and the logic of single-place predicates. *Sov. Phys., Dokl.*, 6:753–755, 1961. ISSN 0038-5689.

Nicolas Troquard and Philippe Balbiani. Propositional dynamic logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2015 edition, 2015.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, 1989.

Pierre Wolper. Temporal logic can be more expressive. *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 340–348, 1981.