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
Marrco Favorito
ID number 1609890

Thesis Advisor
Prof. Giuseppe De Giacomo

Co-Advisor
Dr. co-advisor

Academic Year 2017/2018

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

Prof. ... (chairman)

Prof. ...

Prof. ...

Prof. ...

Prof. ...

Prof. ...

Prof. ...

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

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

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., $\Diamond rich$ (eventually I will become rich), $Request \implies \Diamond Response$ (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., $\Box happy$ (I'm always happy), $\Box\neg(temperature > 30)$ (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. $\Diamond\Box dead$ (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., $\Box\Diamond ready \implies \Box\Diamond run$ (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 $\langle s_1, s_2, \ldots\rangle$ such that $\forall i. R(s_i, s_{i+1})$. From a path we can build a *word* w on the path ρ by mapping each state of the sequence with L, namely:

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

In simpler words, a trace of propositional symbols $\mathcal{P}$ is a infinite sequence of combinations of propositional symbols in $\mathcal{P}$. 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 $\langle s_1, s_2, s_3, s_3, s_3 \ldots\rangle$ yields the following trace π:

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

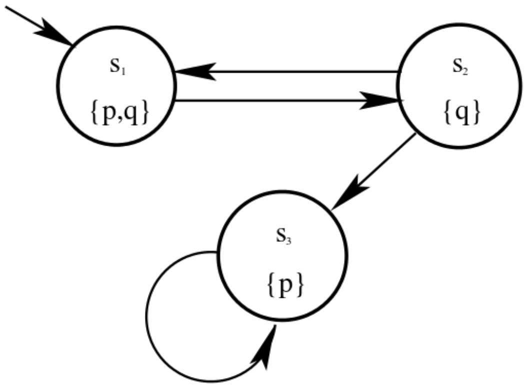


Figure 2.1. An example of Kripke structure.

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:

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

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 $zR_p(\beta)y$

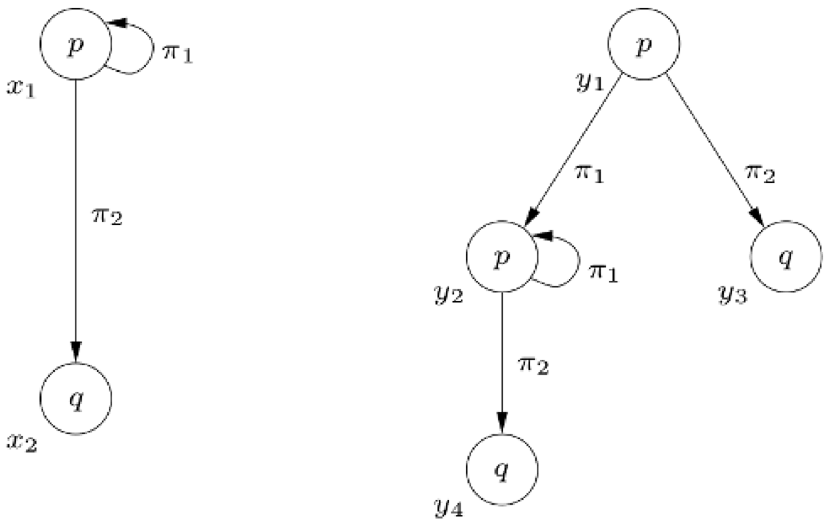


Figure 2.2. Two examples of LTS

- $xR_p(\alpha \cup \beta)y$ 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:

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

And for temporal formulas:

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

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

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

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

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

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:

$$\begin{aligned}
\varphi_1 \lor \varphi_2 &\doteq \neg(\neg\varphi_1 \land \neg\varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 &\doteq \neg\varphi_1 \lor \varphi_2 \\
\varphi_1 \Leftrightarrow \varphi_2 &\doteq \varphi_1 \Rightarrow \varphi_2 \land \varphi_2 \Rightarrow \varphi_1 \\
ff &\doteq \neg tt
\end{aligned}$$

And for temporal formulas:

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

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

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

$[\varrho]\varphi$ 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 $\pi, i \not\models \varphi$
$\pi, i \models \varphi_1 \land \varphi_2$ iff $\pi, i \models \varphi_1 \land \pi, i \models \varphi_2$
$\pi, i \models \langle\phi\rangle\varphi$ iff $i < last \land \pi(i) \models \phi \land \pi, i+1 \models \varphi$
$\pi, i \models \langle\psi?\rangle\varphi$ iff $\pi, i \models \psi \land \pi, i \models \varphi$
$\pi, i \models \langle\varrho_1 + \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\varphi \lor \langle\varrho_2\rangle\varphi$
$\pi, i \models \langle\varrho_1; \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\langle\varrho_2\rangle\varphi$
$\pi, i \models \langle\varrho^*\rangle\varphi$ iff $\pi, i \models \varphi \lor i < last \land \pi, i \models \langle\varrho\rangle\langle\varrho^*\rangle\varphi$ 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	$\langle A\rangle tt$

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

Notice also that every RE$_f$ expression ϱ is captured in LDL$_f$ by $\langle\varrho\rangle \mathit{End}$. 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*: $[\mathit{true}^*]\varphi$, equivalent to LTL$_f$ formula $\Box\varphi$
- *Liveness*: $\langle \mathit{true}^* \rangle\varphi$, equivalent to LTL$_f$ formula $\Diamond\varphi$
- *Conditional Response*: $[\mathit{true}^*](\varphi_1 \Rightarrow \langle \mathit{true}^* \rangle\varphi_2)$, equivalent to LTL$_f$ formula $\Box(\varphi_1 \Rightarrow \Diamond\varphi_2)$
- *Ordered occurrence*: $\langle \mathit{true}^*; \varphi_1; \mathit{true}^*; \varphi_2; \mathit{true}^* \rangle \mathit{End}$ equivalent to the RE$_f$ expression $\mathit{true}^*; \varphi_1; \mathit{true}^*; \varphi_2; \mathit{true}^*$
- *Alternating occurrence*: $\langle (\psi; \varphi)^* \rangle \mathit{End}$ 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 $\langle\varrho_1\rangle \mathit{End}$ and $\langle\varrho_2\rangle \mathit{End}$ 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 $\rho = s_1, s_2, (s_1, s_2)^\omega, s_3, (s_3)^\omega$
- $[\mathit{true}^*]\langle p \vee q \rangle tt$ 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:

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

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

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

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

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

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

2. Iteration: we already analyzed $q = \{\Box A\}$, so we analyze $q = \{\Box \mathit{false}\}$

   - Both with $\Pi = \{\}$ and $\Pi = \{A\}$ we have that:

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

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

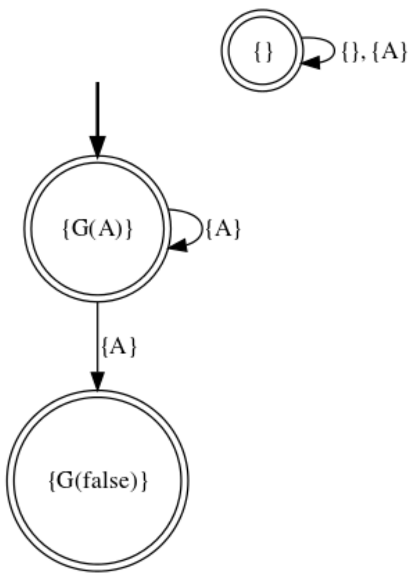
$$\models \partial(false, \Pi) \wedge \partial(\bullet\Box false, \Pi)$$

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

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 $\Box A$

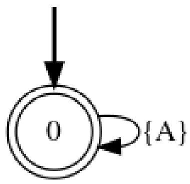


Example 2.14. Analogously to what we did in 2.13, we see a run of the Algorithm 2.1, with the LTL$_f$ formula $\Diamond A$ (A atomic).

0. Set up:

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

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 $\text{LTL}_f/\text{LDL}_f$ formula φ, obtained by Algorithm 2.1:

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

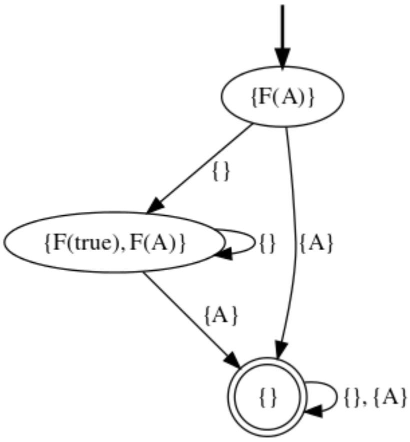
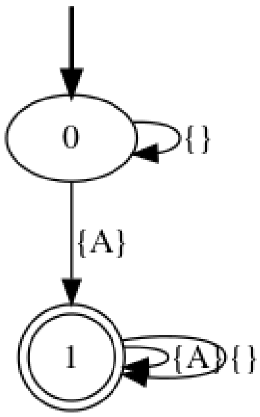


Figure 2.6. The DFA associated to $\Diamond A$



- *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 $\varphi = \Box(A \Rightarrow \Diamond B)$

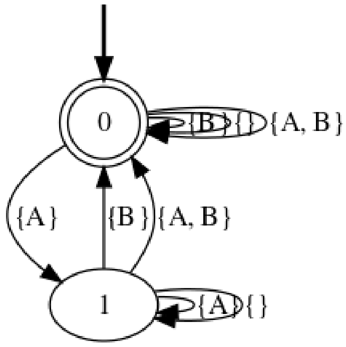
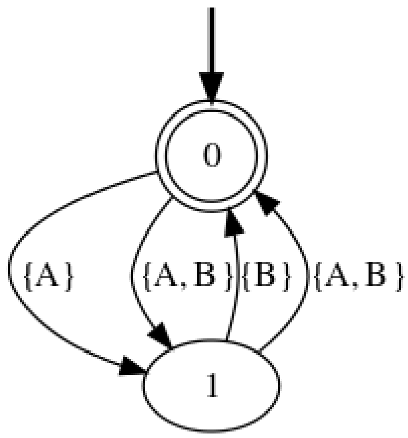


Figure 2.8. The DFA associated to $\varphi = \langle (A; B)^* \rangle End$



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).

□

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

3.1 Main features

3.2 Package structure

3.3 Code examples

3.4 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.

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.

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 return* G_t at time step t is defined as:

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

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

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

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

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

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

Notice that we can rewrite 4.2 and 4.3 recursively, yielding the *Bellman 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. $\forall s \in S. \rho \geq \rho' \iff v_\rho(s) \geq v_{\rho'}(s)$. An *optimal policy* ρ^* is a policy such that $\rho^* \geq \rho$ for all ρ.

4.3 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:         $e(s,a) \leftarrow 1$ $\triangleright$ 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. $\langle s_0, a_0, s_1, a_1, s_2, \ldots, s_n, a_n \rangle$), 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.

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

Definition 4.1. 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 $\bar{R} : (S \times A)^* \to \mathbb{R}$ is the *non-Markovian reward function*, i.e. is defined over trajectories $\langle s_0, a_0, \dots, s_n, a_n\rangle$.

Given a trace $\pi = \langle s_0, a_0, s_1, a_1, \dots, s_n, a_n\rangle$, the *value of* π is:

$$v(\pi) = \sum_{i=1}^{|\pi|} \gamma^{i-1} \bar{R}(\langle \pi(1), \pi(2), \dots, \pi(i)\rangle) \tag{4.13}$$

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

The policy $\bar{\rho}$ in this setting is defined over sequences of states and actions, 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 4.1);
- 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 4.2 (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 $T(s1, a, s2) > 0$ 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 trajectory $\langle s_0, a_0, \dots, s_{n-1}, a_n \rangle$ of $\mathcal{N}$ and $\langle s'_0, a_0, \dots, s'_n, a_n \rangle$ of $\mathcal{M}$, such that $\tau(s'_i) = s_i$ and $\sigma(s_0) = s'_0$, we have $R(\langle s_0, a_0, \dots, s_n, a_n \rangle) = R'(\langle s'_0, a_0, \dots, s'_n, a_n \rangle)$.

Given the Definition 4.2, we give the definition of corresponding policiy:

Definition 4.3 (Bacchus et al. (1996))**.** Let $\mathcal{N}$ be an NMRDP and let $\mathcal{M}$ be the equivalent MDP as defined in Definition 4.2. 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 4.2 and 4.3, 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 4.2 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 $\pi = \langle s_0, a_0, \ldots, s_n, a_n \rangle$ 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.

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:

$$\bar{R}(\langle s_0, a_0, s_1, a_1 \ldots, s_n, a_n \rangle) = \sum_{\phi_i \in \Phi : \langle s_0, s_1, \ldots, s_n \rangle \models \phi_i} r_i \tag{4.15}$$

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, we obtain a compact representation of the required history-dependent policy by considering only relevant history, and can produce this policy using computationally-effective MDP algorithms.

4.5 NMRDP with LTL$_f$/LDL$_f$ rewards

In this section we explain how to specify non-Markovian rewards with LTL$_f$/LDL$_f$ formulas (instead of PLTL) and how the associated MDP expansion works (Brafman et al., 2017), analogously to what we saw with PLTL (Section 4.4).

The temporally extended reward function $\bar{R}$ is similar to Equation 4.15, but instead of using PLTL formula we use LTL$_f$/LDL$_f$ formulas. Formally, given a set of pairs $\{(\varphi_i, r_i)_{i=1}^m\}$ (where φ_i denotes the LTL$_f$/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 $\pi = \langle s_0, a_0, \ldots, s_n, a_n \rangle$, we define $\bar{R}$ as:

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

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

We are interested in doing reinforcement learning in the setting of **?**. That is we want to learn a (possibly optimal) policy for an NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$, whose rewards r_i are given on traces specified by LTL$_f$/LDL$_f$ formulas φ_i, where state space S, action set A and LTL$_f$/LDL$_f$ reward formulas φ_i are known, while the transitions Tr and the rewards r_i are not.

Formally, given the NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$, with Tr and r_i unknown to the learning agent, but sampled during learning, and an initial state $s_0 \in S$, the RL problem over M consists in learning an optimal policy $\bar{\rho}$. Note that, since NMRDP rewards are based on traces, instead of on state-action pairs, typical learning algorithms, such as Q-learning or SARSA **?**, which are based on MDPs, are not applicable.

However in **?**, it has been shown that for any NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$, there exists an MDP $M' = \langle S', A', Tr', R' \rangle$ that is *equivalent* to M in the sense that the states of M can be (injectively) mapped into those of M', in such a way that corresponding (under the mapping) states yield same transition probabilities and corresponding traces have same rewards **?**. Denoting with $\AA_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i \rangle$ (notice that $S \subseteq 2^{\mathcal{P}}$ and δ_i is total) the DFA associated with φ_i, the equivalent MDP $M' = \langle S', A', Tr', R' \rangle$ is built as follows **?**:

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

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

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

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

Theorem 4.2 (**?**). *The NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$ is equivalent to the MDP $M' = \langle S', A', Tr', R' \rangle$ defined above.*

Let ρ' be a (Markovian) policy for M'. It is easy to define an *equivalent* policy on M, i.e., a policy that guarantees the same rewards. To this end, consider a trace $\pi = \langle s_0, a_1, s_1, \ldots, s_n, a_n \rangle$ of M, and assume it leads to state s_n. Moreover, let q_i be the state of A_{φ_i} on the input π. We define the (non-Markovian) policy $\bar{\rho}$ equivalent to ρ' as $\bar{\rho}(\pi) = \rho'(q_1, \ldots, q_m, s_n)$. In particular we have:

Theorem 4.3 (**?**). *Given an NMRDP M, let ρ' be an optimal policy for an equivalent MDP M'. Then, the policy $\bar{\rho}$ for M that is equivalent to ρ' is optimal for M.*

Obviously, typical learning techniques, such as Q-learning or SARSA, are applicable on (the state space of) M' and we can learn an optimal policy ρ' for M'. Thus, an optimal policy for M can be learnt on 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 M' instead of M.

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

Theorem 4.4. *RL for $\text{LTL}_f/\text{LDL}_f$ rewards φ over an NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\{_{i=1}^m \rangle$, with Tr and r_i unknown to the learning agent can be reduced to RL over the MDP M' defined above.*

Why $\text{LTL}_f/\text{LDL}_f$

Our formalism has three important advantages: 1. En- hanced expressive power. We move from linear-time tempo- ral logics used by past authors to LDL f , paying no additional (worst-case) complexity costs. LDL f can encode in polyno- mial time LTL f , regular expressions (RE), the past LTL (PLTL) of (Bacchus, Boutilier, and Grove 1996), and all examples of (ThiÃľbaux et al. 2006). Moreover, LDL f can naturally represent âĂIJprocedural constraintsâĂİ (Baier et al. 2008), i.e., sequencing constraints expressed as programs, using âĂIJifâĂİ and âĂIJwhileâĂİ. In fact, future logics are more commonly used in the model checking community, as they are considered more natural for expressing desirable properties. This is especially true with complex properties that require the power of LDL f

4.6 RL for $\text{LTL}_f/\text{LDL}_f$ Goals

Chapter 5

Automata-based Reward shaping

5.1 Reward Shaping Theory

5.1.1 Classic Reward Shaping

5.1.2 Dynamic Reward Shaping

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

5.3 Reward shaping on-the-fly

Chapter 6

RLTG

6.1 Main features

6.2 Package structure

6.3 Code examples

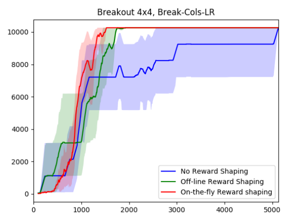
6.4 License

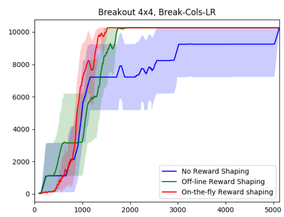
Chapter 7

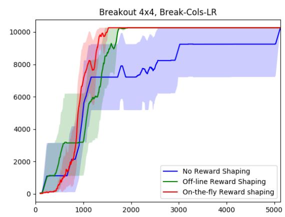
Experiments

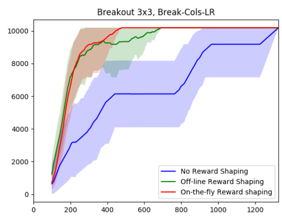
look at experiment introduction in Grzes phd thesis

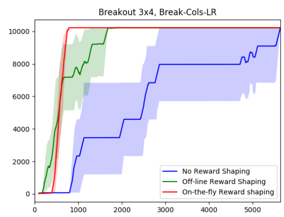
7.1 BREAKOUT

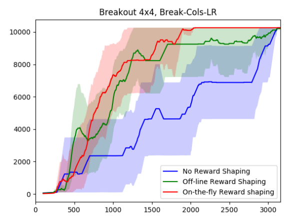












7.2 SAPIENTINO

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.

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 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.

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.

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.

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.