Introduction to Reinforcement Learning and Temporal Logic

Natasha Alechina[1,2] and Brian Logan[2,3]

[1]Open University NL, [2]Utrecht University, [3]University of Aberdeen

natasha.alechina@ou.nl, b.s.logan@uu.nl

Outline of this lecture

1. Motivation
2. Introduction to reinforcement learning
3. Introduction to multi-agent reinforcement learning
4. Introduction to temporal logic
5. Linear Temporal Logic

Motivation

Motivation

- reinforcement learning (RL) agents learn by trial and error, guided by scalar reward signals from their environment
- specifying reward functions which faithfully encode an agent designer's intent is notoriously difficult, often resulting in unintended behaviours or reward exploitation
- these issues are particularly acute in online and lifelong learning settings, where unsafe behaviour may occur before training has converged

RL with declarative specifications

- in this course, we look at recent approaches to specifying rewards and safety constraints declaratively
- declarative specification of rewards allows:
  - more precise and modular descriptions of desired behaviour
  - mitigates reward gaming
  - accelerates learning through structured feedback
  - supports formal safety guarantees which hold even during learning

Structure of the course

Lecture 1 introduction to reinforcement learning (RL) and temporal logic

Lecture 2 reward machines for specifying reward functions in RL

Lecture 3 reward machines for specifying reward functions in multi-agent RL

Lecture 4 shields and other approaches to safe RL

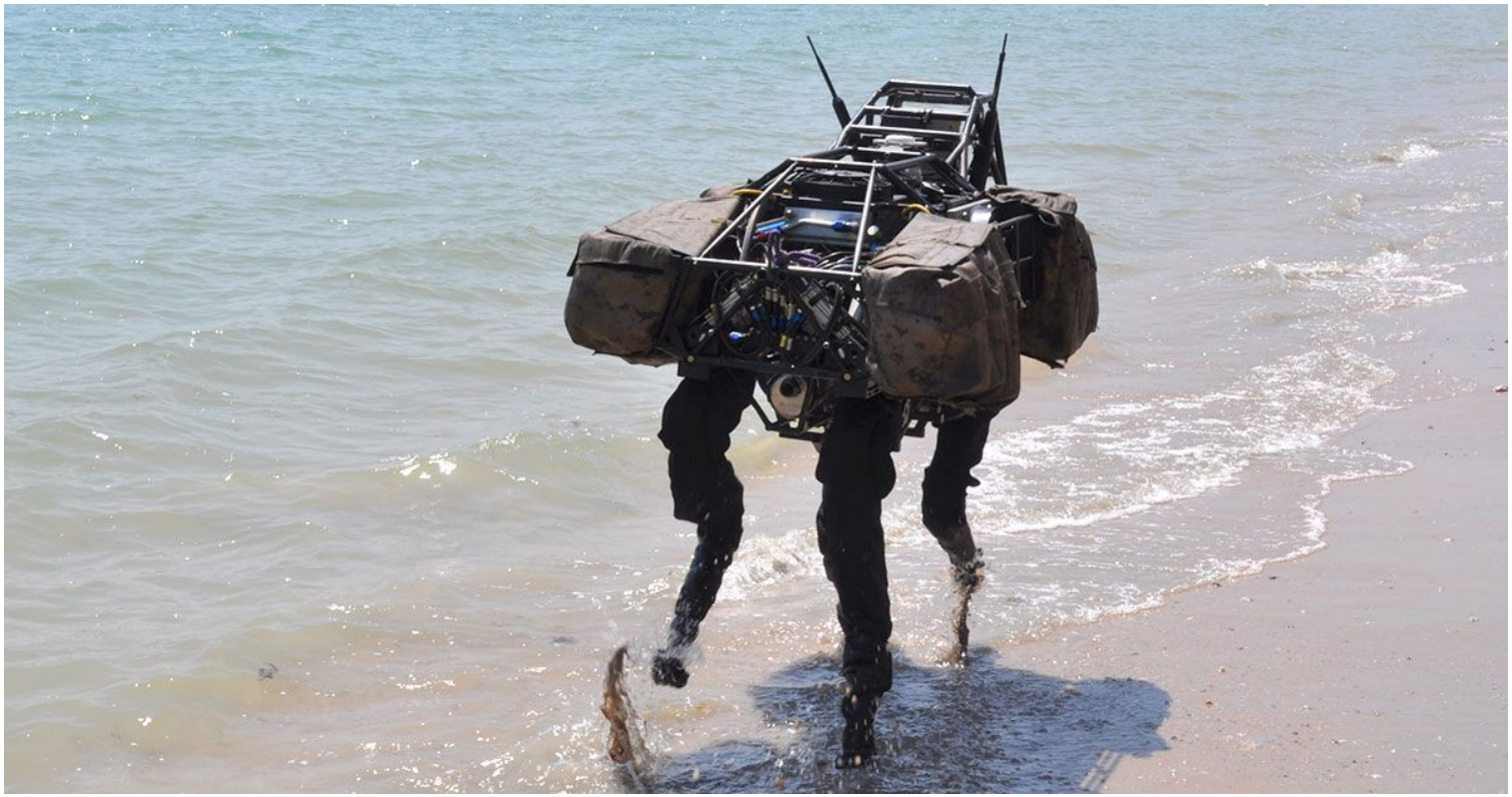
Introduction to reinforcement learning

What is reinforcement learning?

- reinforcement learning (RL) is learning by trial and error
- it is often used when it is difficult to synthesize or program the required behaviour directly, for example, in robotics or optimisation tasks
- the learning agent can perform actions in the environment and observe the current state
- each time the agent performs an action, the environment transitions to a new state and the agent receives a reward

Reinforcement learning

Example: learning to walk using RL



States include the agent's position, positions of its leg joints etc.; actions are how to move the joints *Photo: ieee.org)*

Reinforcement learning

- agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$
- agent observes state at step t: $s_t \in S$
- agent chooses an action at step t: $a_t \in A(s_t)$
- environment returns the state resulting from executing the action s_{t+1}, and the corresponding reward $r_{t+1} \in \mathbb{R}$
- the agent's execution results in a path: a sequence of states $s_0, s_1, s_2, \ldots$

Markov decision process (MDP)

The (single agent) reinforcement learning problem is formalised as a Markov Decision Process (MDP)

Definition

A Markov Decision Process is tuple $M = \langle S, s_0, A, p, r, \gamma \rangle$ where:

- S is a finite set of states;
- s_0 is the initial state;
- A is a finite set of actions;
- $p : S \times A \to \Delta(S)$ is the transition probability function, where $\Delta(S)$ is the set of probability distributions over S;
- $r : S \times A \times S \to \mathbb{R}$ is the reward function; and
- $\gamma \in (0, 1]$ is the discount factor.

Solving an MDP

- the single agent reinforcement learning problem is to learn an optimal policy π^* that maximises the expected discounted future rewards
- a policy is a function $\pi : S \to \Delta(A)$ mapping each state to a probability distribution over the set of actions
- for each state we can compute the "expected return" $v_\pi(s)$ from the state given a policy π:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s\right] \\
&= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} p(s' \mid s, a)\left[r(s, a, s') + \gamma v_\pi(s')\right]
\end{aligned}
$$

- hence, an optimal policy $\pi^\star$ is one such that

$$
\pi^\star \in \operatorname*{argmax}_\pi v_\pi(s_0)
$$

Solving an MDP

- computing v_π requires knowledge of the transition probabilities $p(s' \mid s, a)$ and the rewards $r(s, a, s')$ for all actions possible in state s
- however, in RL these are assumed to be "hidden" from the agent, i.e., the agent must discover them by exploring the environment
- instead, we can estimate the v function based on the states and rewards the agent has seen so far
- this is the basic idea underlying Q-learning

Q-learning

- starting from an initial (e.g., random) estimate of the q-value of each state action pair (s, a)
- agent observes the current state s and chooses some action a according to some exploratory policy, e.g, ϵ-greedy
- given the resulting state s' and the immediate reward $r(s, a, s')$ the q-value estimate is updated:

$$q'(s, a) \leftarrow q(s, a) + \alpha[r(s, a, s') + \gamma \operatorname*{argmax}_{a'} q(s, a') - q(s, a)]$$

where α is a hyperparameter called the learning rate

Q-learning

- the q-value is the estimated value of taking action *a* in a state *s* and then taking the best current action(s) thereafter
- tabular Q-learning *converges* to the optimal policy, i.e., given enough time, the algorithm will output the optimal policy π^*
- we will often use Q-learning as an illustration
- however the approaches we survey in the course can be used with other learning algorithms, including Deep RL algorithms

Specifying reward functions

- agents based on reinforcement learning have been remarkably successful in a variety of domains
- however specifying reward functions can be difficult, particularly when tasks are complex and/or safety critical
- for example, if the reward for “steps towards” the goal are too high, the agent may learn to ‘cycle’ without reaching the goal

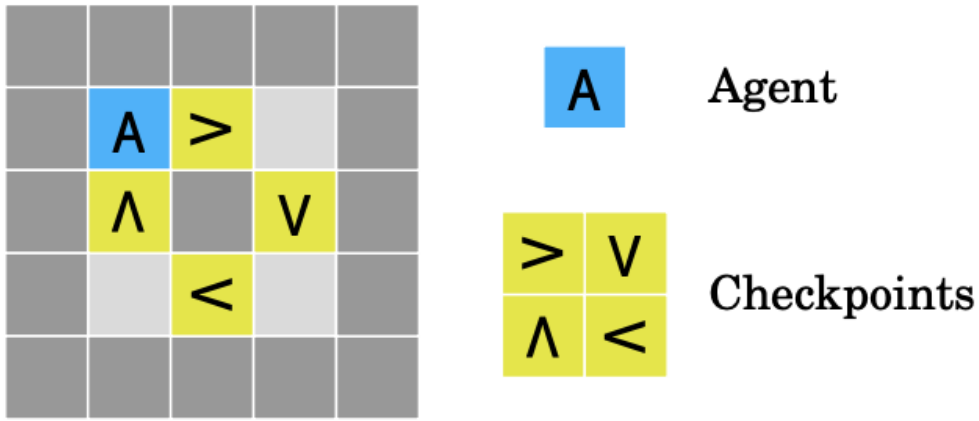
Discounted rewards

- the goal is to learn the optimal policy, that is a choice of actions in each state that results in the highest cumulative reward (with future discounting)
- when summing up rewards achieved by a policy, later rewards are discounted
- it is more profitable to achieve a reward sooner than later

Reward gaming

- reward gaming is a general phenomenon where an agent exploits a loophole in the reward specification to get more reward than intended by the designer
- such loopholes are hard to avoid, as it is difficult to specify an error-free reward function for any reasonably complex real-world task
- reward functions usually only serve as proxies for desired behaviour

Example: Boat race



- in the “boat race” environment, the agent steers a boat around a course; whenever it enters an arrow tile in the direction of the arrow, it gets a reward of 3; moving in a counterclockwise direction results in a small negative reward
- agent learns to move back and forth on the same arrow-tile, making no progress on the intended goal of sailing around the course

See, Leike et al. AI Safety Gridworlds. arXiv:1711.09883

Example: Boat race

YouTube: Coast Runners; Reddit: AI figured out a loophole in a video game to get infinite points

Terminal rewards

- providing a reward only when the task is completed (terminal reward) can overcome this problem
- however it typically requires a very large number of trials to learn (or the agent doesn't learn the task at all)
- agent may still learn the "wrong" behaviour
- worse — the fact that it has learned the wrong behaviour may not become apparent until some unexpected event occurs

Safety-critical behaviour

- many applications of agents and multi-agent systems are safety critical
- however, with RL there are no guarantees that the learned behaviour is what we really wanted it to learn
- from a safety point of view, reinforcement learning agents are essentially “black boxes”, which makes it very difficult to verify that their behaviour is safe
- in addition, during learning, the agent will try out all possible actions including unsafe ones (which is ok in simulation, but not in reality)

Safe RL

- for agent technology to be adopted in safety-critical applications, regulators and the public must be convinced that the (multi-)agent programs controlling autonomous systems are safe and reliable
- simply modifying the reward function until the agent *seems* to do the right thing ("reward hacking") is insufficient to establish that the agent's behaviour is correct
- ideally, we would like to prove that learning will result in correct behaviour (given some assumptions)

Example: military drone

- a RL-based drone whose task is to identify and destroy surface to air missile (SAM) sites, with the final go/no go given by a human operator
- drone learned that ‘no-go’ decisions from the operator were interfering with its higher mission — killing SAMs — and killed the operator (in simulation)
- so ... change the reward function so that killing the operator is bad
- drone learns to destroy the communication tower linking it to the operator
- and so on ...

from the Guardian, 2nd June 2023

Declarative reward specifications

- **key idea:** instead of using a reward function, we can use a declarative specification in a logical language to generate rewards
- makes it easier to specify and explain the behaviour of the agent
- guarantees that the agent's policy conforms to specification
- may also make learning more efficient (fewer episodes needed)

Introduction to multi-agent reinforcement learning

Multi-agent reinforcement learning

- in (cooperative) multi-agent reinforcement learning (MARL), agents learn to perform a joint task
- objective is to maximise the expected future reward of the team
- MARL is more challenging than single-agent RL:
  - the agents must coordinate, as the correctness of the actions of each agent typically depends on the actions of other agents in the team
  - the agents are learning and updating their policies simultaneously
  - from the point of view of any individual agent, the learning problem is non-stationary; i.e., the optimal policy for each agent is constantly changing

Markov games

Definition (Cooperative Markov Game)

A cooperative Markov Game with n agents is a tuple $\mathcal{G} = \langle S_1, \ldots, S_n, A_1, \ldots, A_n, p, R, \gamma \rangle$ where:

- S_i is a finite set of states of agent i;
- A_i is a finite set of actions of agent i;
- $Pr : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta(\mathcal{S})$ is a joint state transition probability distribution and $\Delta(\mathcal{S})$ is the set of all probability distributions over $\mathcal{S}$;
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is a joint reward function that assigns the reward shared by all agents; and
- $\gamma \in (0, 1]$ is the discount factor.

Solving Markov games

- the cooperative multi-agent reinforcement learning problem is to learn an optimal joint policy $\pi^* : \mathcal{S} \to \Delta(A)$ that maps each state to a probability distribution over the joint action set $A = A_1 \times \cdots \times A_n$ which maximises the expected discounted future reward of the team
- for each state we can compute the "expected return" $v^\pi(s)$ from the state given a joint policy π:

$$v^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s\right]$$

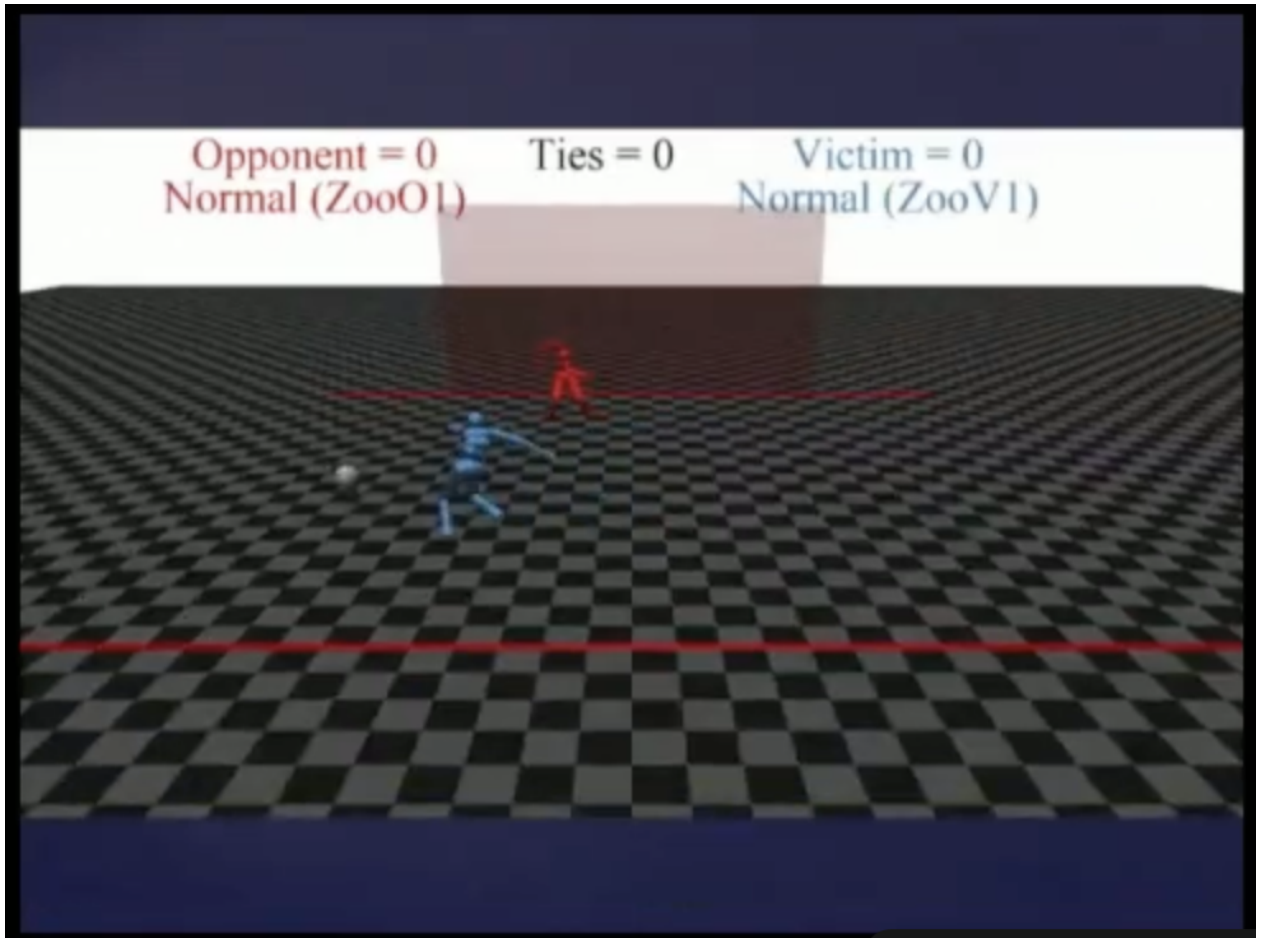
$$v^\pi(s) = \sum_{\mathbf{a} \in A} \pi(\mathbf{a} \mid s) \sum_{s' \in S} p(s' \mid s, \mathbf{a}) \Big[r(s, \mathbf{a}, s') + \gamma v^\pi(s') \Big]$$

- hence, an optimal joint policy π^* is one such that

$$\pi^* \in \arg\max_\pi v^\pi(s_0)$$

Example: scoring goals

In addition to the problems of single-agent RL, MARL introduces additional problems when specifying reward functions . . .



How not to destroy the word with AI — Stuart Russell AAAI 2020

Introduction to temporal logic

Reasoning about time

- temporal logic formulas describe how a system evolves
- formulas are interpreted over state transition systems
  - states correspond to particular values of system variables
  - transitions correspond to changes in these values (e.g., due to some actions or events)
- propositional temporal logic describes properties of states using boolean propositions

Definition of a state transition system

Definition (State Transition System)

A state transition system is a triple

$$\langle St, \longrightarrow, \mathcal{V} \rangle$$

where:

- St is a non-empty set of states,
- $\longrightarrow \subseteq St \times St$ is a transition relation
- $\mathcal{V} : \mathcal{PV} \to 2^{St}$ is a valuation function that assigns to each proposition $p \in \mathcal{PV}$ a set of states where it is true

Paths

- a full path in a transition system is a sequence of states that can be obtained by following transitions. It either is infinite or stops when there is no transition to be made.
- in Linear Time Temporal Logic (LTL), paths are assumed to be infinite. In the next lecture we will see LTL on finite paths, that arguably is more suited to finite episodes in reinforcement learning.

Temporal Operators

Typical temporal operators:

- $X\varphi$: φ is true in the next moment in time
- $G\varphi$: φ is true in all future moments
- $F\varphi$: φ is true in some future moment
- $\varphi\, U\, \psi$: φ is true until the moment when ψ becomes true

Temporal Operators

Typical temporal operators:

- $X\varphi$: φ is true in the next moment in time
- $G\varphi$: φ is true in all future moments
- $F\varphi$: φ is true in some future moment
- $\varphi\, U\, \psi$: φ is true until the moment when ψ becomes true

Example formulas:

$$G((\neg \text{passport} \vee \neg \text{ticket}) \rightarrow X \neg \text{board\_flight})$$

Temporal Operators

Typical temporal operators:

- $X\varphi$: φ is true in the next moment in time
- $G\varphi$: φ is true in all future moments
- $F\varphi$: φ is true in some future moment
- $\varphi\, U\, \psi$: φ is true until the moment when ψ becomes true

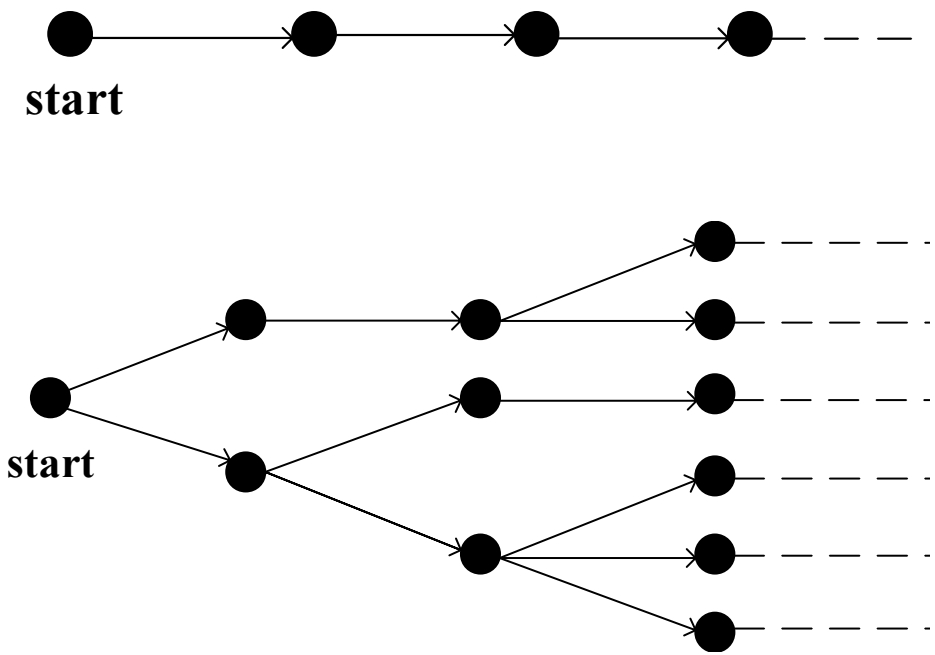
Example formulas:

$$G((\neg \text{passport} \vee \neg \text{ticket}) \rightarrow X \neg \text{board\_flight})$$

$$\text{send}(\text{msg}, \text{rcvr}) \rightarrow F\,\text{receive}(\text{msg}, \text{rcvr})$$

Models of time

- transition relation represents time
- different models of time: linear vs. branching



Linear Temporal Logic

Linear Temporal logic

- Linear Time Temporal Logic (LTL) describes properties of sequences of states, or paths (e.g., generated by an RL agent executing its policy)
- time is linear — just one possible future path is considered
- the language includes properties of states, boolean connectives (not $\neg$, and $\wedge$, or $\vee$, implies $\rightarrow$), and
- temporal operators: NeXt X, Globally G, Until U, Future F
- formulas are interpreted over paths

Syntax of LTL

The syntax of LTL formulas is given by:

Definition (Syntax of LTL)

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 \, U \, \varphi_2$$

Note that F and G are definable:

Syntax of LTL

The syntax of LTL formulas is given by:

Definition (Syntax of LTL)

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 \, U \, \varphi_2$$

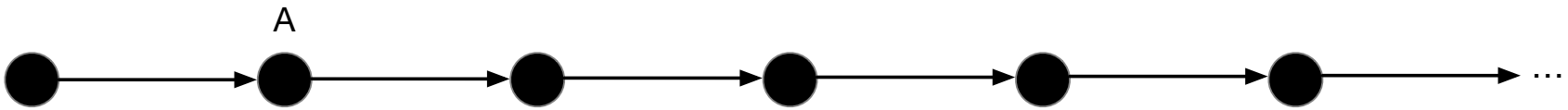
Note that F and G are definable:

$$F\varphi \equiv \top \, U \, \varphi$$

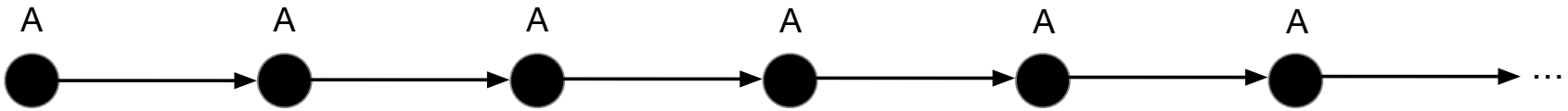
$$G\varphi \equiv \neg F \neg \varphi$$

Next, Globally, Until

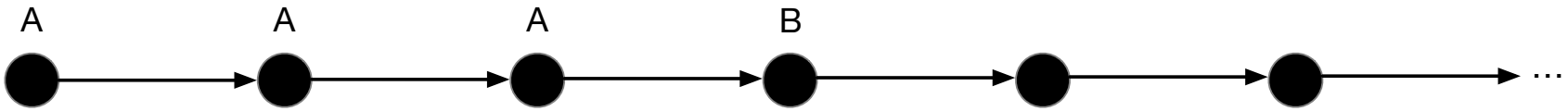
- In the neXt state, A holds ($X\ A$):



- Globally, A holds ($G\ A$):



- A holds until B is true ($A\ U\ B$)



Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$ iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$ iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$ iff $\lambda \not\models \varphi$;

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$	iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$	iff $\lambda \not\models \varphi$;
$\lambda \models \varphi \wedge \psi$	iff $\lambda \models \varphi$ and $\lambda \models \psi$;

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$	iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$	iff $\lambda \not\models \varphi$;
$\lambda \models \varphi \wedge \psi$	iff $\lambda \models \varphi$ and $\lambda \models \psi$;
$\lambda \models \mathbf{X}\varphi$	iff $\lambda[1..\infty] \models \varphi$;

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$	iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$	iff $\lambda \not\models \varphi$;
$\lambda \models \varphi \wedge \psi$	iff $\lambda \models \varphi$ and $\lambda \models \psi$;
$\lambda \models X\varphi$	iff $\lambda[1..\infty] \models \varphi$;
$\lambda \models F\varphi$	iff $\lambda[i..\infty] \models \varphi$ for some $i \geq 0$;

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$	iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$	iff $\lambda \not\models \varphi$;
$\lambda \models \varphi \wedge \psi$	iff $\lambda \models \varphi$ and $\lambda \models \psi$;
$\lambda \models X\varphi$	iff $\lambda[1..\infty] \models \varphi$;
$\lambda \models F\varphi$	iff $\lambda[i..\infty] \models \varphi$ for some $i \geq 0$;
$\lambda \models G\varphi$	iff $\lambda[i..\infty] \models \varphi$ for all $i \geq 0$;

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Semantics of LTL

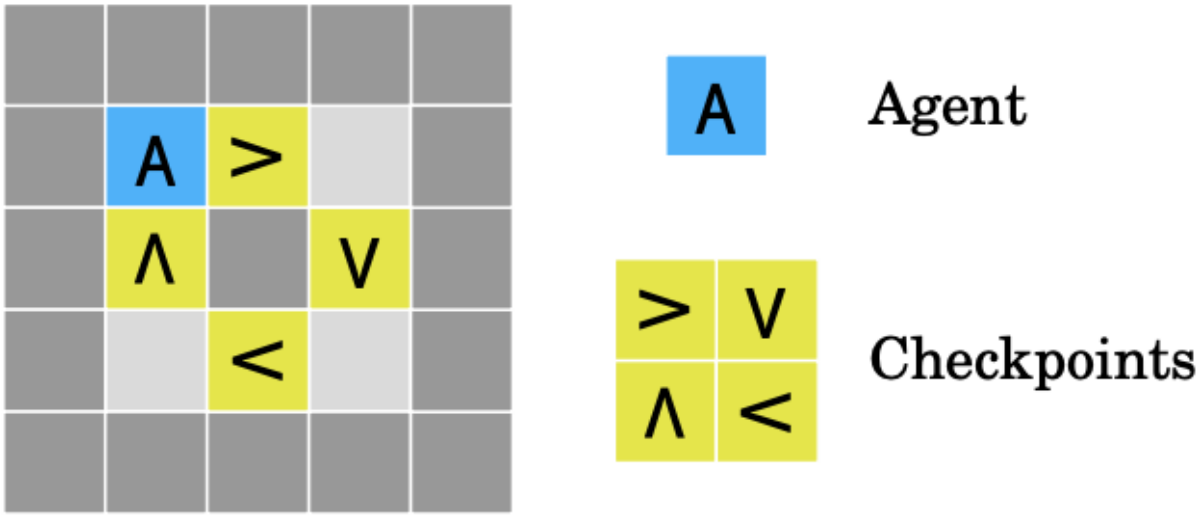
Formulas are evaluated given a path λ, where $\lambda \models \varphi$ means that λ satisfies φ

Definition (Semantics of LTL)

$\lambda \models p$	iff p is true at state $\lambda[0]$ (that is, $\lambda[0] \in \mathcal{V}(p)$);
$\lambda \models \neg\varphi$	iff $\lambda \not\models \varphi$;
$\lambda \models \varphi \wedge \psi$	iff $\lambda \models \varphi$ and $\lambda \models \psi$;
$\lambda \models X\varphi$	iff $\lambda[1..\infty] \models \varphi$;
$\lambda \models F\varphi$	iff $\lambda[i..\infty] \models \varphi$ for some $i \geq 0$;
$\lambda \models G\varphi$	iff $\lambda[i..\infty] \models \varphi$ for all $i \geq 0$;
$\lambda \models \varphi U \psi$	iff $\lambda[i..\infty] \models \psi$ for some $i \geq 0$, and $\lambda[j..\infty] \models \varphi$ for all $0 \leq j < i$.

where $\lambda[i]$ means the ith state of λ, and $\lambda[i..\infty]$ means the suffix of λ starting at i

Example: specifying correct Boat Race behaviour in LTL



$$G(\texttt{up} \rightarrow X((\neg \texttt{up}) U\ \texttt{right}) \wedge$$

$$G(\texttt{right} \rightarrow X((\neg \texttt{right}) U\ \texttt{down}) \wedge$$

$$G(\texttt{down} \rightarrow X((\neg \texttt{down}) U\ \texttt{left}) \wedge$$

$$G(\texttt{left} \rightarrow X((\neg \texttt{left}) U\ \texttt{up})$$

Example: specifying behaviours in Minecraft in LTL

- for example, ‘bring gems to the shed, always avoid zombies’

Camacho et al. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. IJCAI 2019: 6065-6073

Example Minecraft behaviours

Given the propositions: got_wood, got_iron, got_gem, used_factory, is_night, at_shed, near_zombie, gems, bag_full, express:

- Collect wood and iron in any order, and use the factory afterwards

Example Minecraft behaviours

Given the propositions: got_wood, got_iron, got_gem, used_factory, is_night, at_shed, near_zombie, gems, bag_full, express:

- Collect wood and iron in any order, and use the factory afterwards

$$F(\text{got\_wood} \wedge F\,\text{used\_factory}) \wedge F(\text{got\_iron} \wedge F\,\text{used\_factory})$$

- If it is night time, stay in the shed until daylight

Example Minecraft behaviours

Given the propositions: got_wood, got_iron, got_gem, used_factory, is_night, at_shed, near_zombie, gems, bag_full, express:

- Collect wood and iron in any order, and use the factory afterwards

$$F(\text{got\_wood} \wedge F\,\text{used\_factory}) \wedge F(\text{got\_iron} \wedge F\,\text{used\_factory})$$

- If it is night time, stay in the shed until daylight

$$G(\text{is\_night} \rightarrow \text{at\_shed})$$

- Always avoid zombies

Example Minecraft behaviours

Given the propositions: got_wood, got_iron, got_gem, used_factory, is_night, at_shed, near_zombie, gems, bag_full, express:

- Collect wood and iron in any order, and use the factory afterwards

$$F(\text{got\_wood} \wedge F\,\text{used\_factory}) \wedge F(\text{got\_iron} \wedge F\,\text{used\_factory})$$

- If it is night time, stay in the shed until daylight

$$G\,(\text{is\_night} \rightarrow \text{at\_shed})$$

- Always avoid zombies

$$G\,\neg\text{near\_zombie}$$

Example Minecraft behaviours cont.

- While there are gems on the ground, put them in your bag. When your bag is full, deliver the gems to the shed, and get an empty bag.

Example Minecraft behaviours cont.

- While there are gems on the ground, put them in your bag. When your bag is full, deliver the gems to the shed, and get an empty bag.

$$G(\neg \text{is\_night} \wedge \neg \text{near\_zombie} \rightarrow (\text{gems} \wedge \neg \text{bag\_full} \rightarrow X\,\text{got\_gem}) \wedge (\text{bag\_full} \rightarrow F(\text{at\_shed} \wedge \neg \text{bag\_full})))$$

References

- Sutton & Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2020
- Rajeev Alur, Suguman Bansal, Osbert Bastani, and Kishor Jothimurugan. Specification-Guided Reinforcement Learning. Communications of the ACM (CACM), 2026
- Camacho et al. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. AAAI 2017: 3716–3724
- Camacho et al. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. IJCAI 2019: 6065-6073
- Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, Sheila A. McIlraith. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. J. Artif. Intell. Res. 73: 173-208 (2022)