Teaching Multiple Tasks to a Reinforcement Learning Agent using Linear Temporal Logic (LTL)

Rodrigo Toro Icarte Toryn Q. Klassen
Richard Valenzano Sheila A. McIlraith

Computer Science
UNIVERSITY OF TORONTO

ELEMENT AI

VECTOR INSTITUTE | INSTITUT VECTEUR

Learning by Instruction Workshop,
NeurIPS 2018

Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

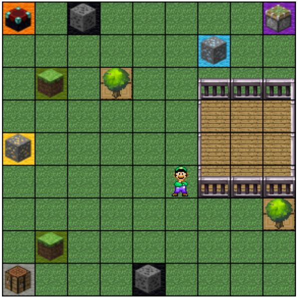
Outline

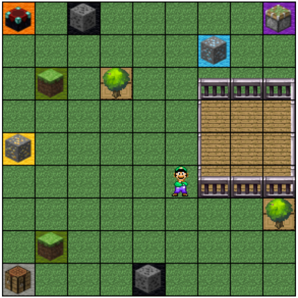
1. **Motivation**
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Motivation

How can we instruct RL agents?

Running example

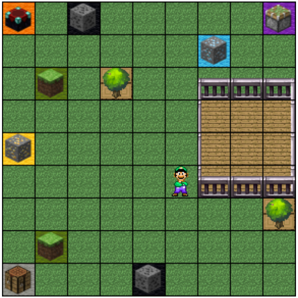




Luigi can collect raw materials:

wood	grass	iron	gold	gems

Running example



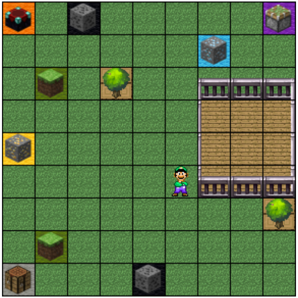
Luigi can collect raw materials:

wood grass iron gold gems

... and make new objects in:

factory toolshed workbench

Running example



Luigi can collect raw materials:

wood grass iron gold gems

... and make new objects in:

factory toolshed workbench

Make a bridge: get wood, iron, and use the factory

Running example

Task type	Example
Single goal	get wood
Sequence of goals	get wood and then use the factory
Disjunctive goals	get wood or iron
Conjunctive goals	get grass and iron
Safety constraints	do not leave the shelter at night

Running example

Task type	Example
Single goal	get wood
Sequence of goals	get wood and then use the factory
Disjunctive goals	get wood or iron
Conjunctive goals	get grass and iron
Safety constraints	do not leave the shelter at night

How can we instruct RL agents?

Motivation

Question: How can we instruct RL agents?
Proposal: Let's use language to describe tasks!

Motivation

Question: How can we instruct RL agents?
Proposal: Let's use language to describe tasks!

Desirable properties:

- It is expressive.
- RL agents understand it:
    - Task description *!* Reward function.
    - Task description *!* Learn faster.
- Humans understand it.

Outline

Outline

Related work

Hindsight Experience Replay by Andrychowicz et al. (NIPS-17)

Language: Single goal condition
Advantage: Learn to achieve goals in parallel (off-policy RL)

Task	HER
get wood	3
get wood and then use the factory	
get wood or iron	
get grass and iron	
do not leave the shelter at night	
Off-policy learning	3

Related work

Modular Multitask RL with Policy Sketches by Andreas et al. (ICML-17)

Language: Sequence of sub-goals (called sketch)
Advantage: Decompose the problem using the sketch.

Task	HER	Sketches
get wood	3	3
get wood and then use the factory		3
get wood or iron		
get grass and iron		
do not leave the shelter at night		
Off-policy learning	3	
Task decomposition		3

Related work

Teaching Multiple Tasks to an RL Agent using LTL

Language: Linear Temporal Logic (LTL)
Advantage: Decompose and use off-policy RL to learn subtasks.

Task	HER	Sketches	LTL
get wood	3	3	3
get wood and then use the factory		3	3
get wood or iron			3
get grass and iron			3
do not leave the shelter at night			3
Off-policy learning	3		3
Task decomposition		3	3

Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Outline

1. Motivation
2. Related work
3. Approach:
   - **Why (and how) we use LTL for specifying tasks in RL.**
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

De ning tasks using LTL

Idea: Let's give the RL agent a set of high-level event detectors.

De ning tasks using LTL

Idea: Let's give the RL agent a set of high-level event detectors.

Example

$$P = \{\texttt{got\_wood}, \texttt{got\_iron}, \texttt{got\_grass}, \texttt{used\_workbench}, \texttt{used\_factory}, \texttt{is\_night}, \texttt{at\_shelter}, \ldots\}$$

De ning tasks using LTL

Idea: Let's give the RL agent a set of high-level event detectors.

Example

$P =$ *f*`got_wood`, `got_iron`, `got_grass`, `used_workbench`, `used_factory`, `is_night`, `at_shelter`, ...*g*

Use LTL to define tasks by composing occurrences of events in P

Defining tasks using LTL

Linear Temporal Logic (syntax)

LTL augments propositional logic with temporal operators $\bigcirc$ (*next*), $\Diamond$ (*eventually*), and U (*until*):

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \Diamond\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \text{ with } p \in P$$

Linear Temporal Logic (syntax)

LTL augments propositional logic with temporal operators (*next*), (*eventually*), and U (*until*):

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \quad \varphi \mid \quad \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \text{ with } p \in \mathcal{P}$$

Examples

```
got_wood
(got_grass ∧  used_factory)
got_wood ∨  got_iron
got_grass ∧  got_iron
(is_night → at_shelter) U got_wood
```

Defining tasks using LTL

Linear Temporal Logic (syntax)

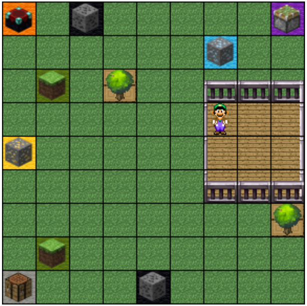
LTL augments propositional logic with temporal operators $\bigcirc$ (*next*), $\Diamond$ (*eventually*), and U (*until*):

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \Diamond\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \text{ with } p \in P$$

Examples

eventually `got_wood`
eventually (`got_grass` **and eventually** `used_factory`)
eventually `got_wood` **or eventually** `got_iron`
eventually `got_grass` **and eventually** `got_iron`
(`is_night` $\rightarrow$ `at_shelter`) **until** `got_wood`

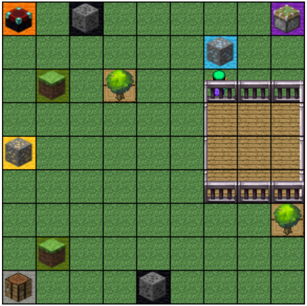
Example



Detected events
at_shelter

$$' = \textbf{eventually}(\texttt{got\_iron}\ \textbf{and eventually}\ \texttt{used\_factory})\ \textbf{and eventually}\ \texttt{got\_gold}$$

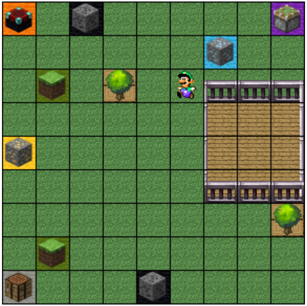
Example



Detected events

none

′ = **eventually**(`got_iron` **and eventually** `used_factory`)
and eventually `got_gold`

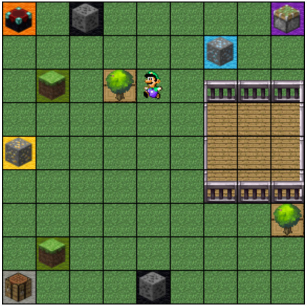


Detected events

none

′ = **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

Example

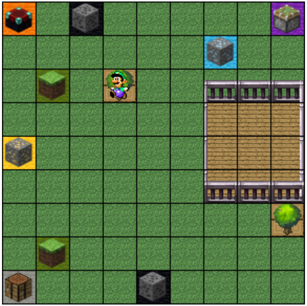


Detected events

none

$$' = \textbf{eventually}(\texttt{got\_iron}\ \textbf{and eventually}\ \texttt{used\_factory})\ \textbf{and eventually}\ \texttt{got\_gold}$$

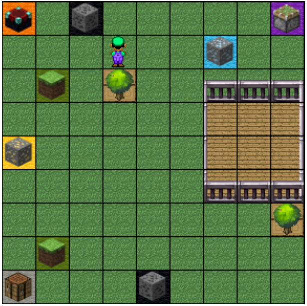
Example



Detected events
got_wood

′ = **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

Example

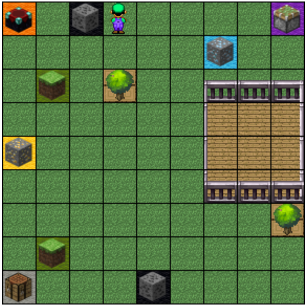


Detected events

none

$' =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

Example



Detected events
none

$' = $ **eventually**(`got_iron` **and eventually** `used_factory`)
and eventually `got_gold`

Example

Detected events
got_iron

′ = **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

Detected events
got_iron

$'$ = **eventually**(~~got_iron~~ **and eventually** used_factory)
and eventually got_gold

Detected events
got_iron

$'$ = **eventually** used_factory **and eventually** got_gold

LTL progression

LTL progression

Given an LTL formula φ and state s, we can *progress* φ using s:

- $\text{prog}(s, p) = \text{true}$ if $p \in L(s)$, where $p \in P$
- $\text{prog}(s, p) = \text{false}$ if $p \notin L(s)$, where $p \in P$
- $\text{prog}(s, \neg\varphi) = \neg\,\text{prog}(s, \varphi)$
- $\text{prog}(s, \varphi_1 \wedge \varphi_2) = \text{prog}(s, \varphi_1) \wedge \text{prog}(s, \varphi_2)$
- $\text{prog}(s, \quad \varphi) = \varphi$
- $\text{prog}(s, \quad \varphi) = \text{prog}(s, \varphi) \vee \quad \varphi$
- $\text{prog}(s, \varphi_1 \,\text{U}\, \varphi_2) = \text{prog}(s, \varphi_2) \vee (\text{prog}(s, \varphi_1) \wedge \varphi_1 \,\text{U}\, \varphi_2)$

LTL progression

LTL progression

Given an LTL formula φ and state s, we can *progress* φ using s:

- $\text{prog}(s, p) = \text{true}$ if $p \in L(s)$, where $p \in \mathcal{P}$
- $\text{prog}(s, p) = \text{false}$ if $p \notin L(s)$, where $p \in \mathcal{P}$
- $\text{prog}(s, \neg\varphi) = \neg\,\text{prog}(s, \varphi)$
- $\text{prog}(s, \varphi_1 \wedge \varphi_2) = \text{prog}(s, \varphi_1) \wedge \text{prog}(s, \varphi_2)$
- $\text{prog}(s, \bigcirc\varphi) = \varphi$
- $\text{prog}(s, \Diamond\varphi) = \text{prog}(s, \varphi) \vee \Diamond\varphi$
- $\text{prog}(s, \varphi_1 \,\mathsf{U}\, \varphi_2) = \text{prog}(s, \varphi_2) \vee (\text{prog}(s, \varphi_1) \wedge \varphi_1 \,\mathsf{U}\, \varphi_2)$

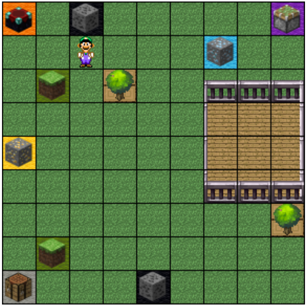
This is a **correct** and **well-defined** procedure!

Example

Detected events
got_iron

′ = **eventually** used_factory **and eventually** got_gold

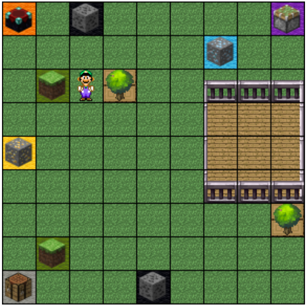
Example



Detected events
none

$\varphi' =$ **eventually** `used_factory` **and eventually** `got_gold`

Example

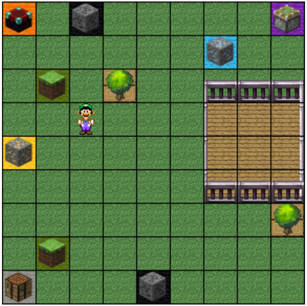


Detected events

none

$' =$ **eventually** `used_factory` **and eventually** `got_gold`

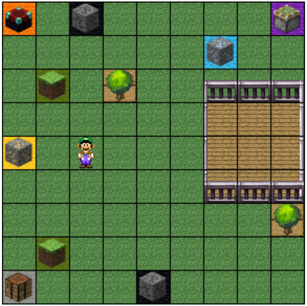
Example



Detected events
none

$' =$ **eventually** `used_factory` **and eventually** `got_gold`

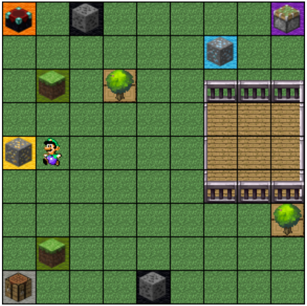
Example



Detected events
none

′ = **eventually** `used_factory` **and eventually** `got_gold`

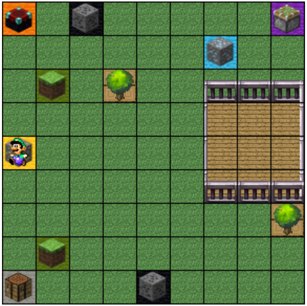
Example



Detected events
none

$'$ = **eventually** `used_factory` **and eventually** `got_gold`

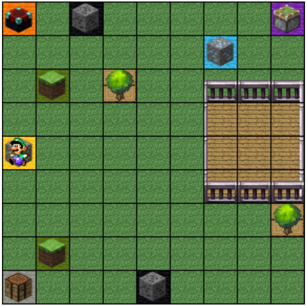
Example



Detected events
got_gold

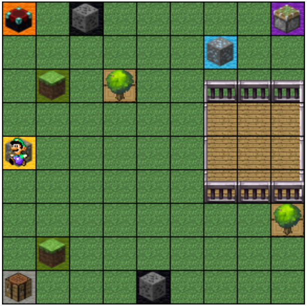
$' =$ **eventually** used_factory **and eventually** got_gold

Example



Detected events
got_gold

′ = **eventually** used_factory **and eventually** ~~got_gold~~

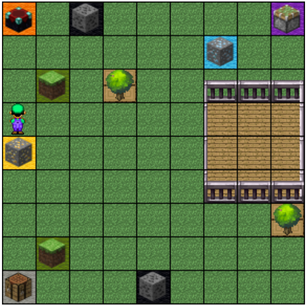


Detected events

got_gold

′ = **eventually** used_factory

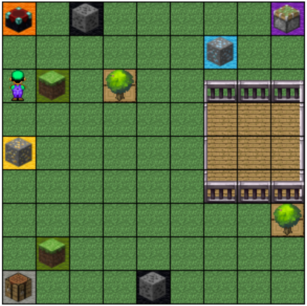
Example



Detected events
none

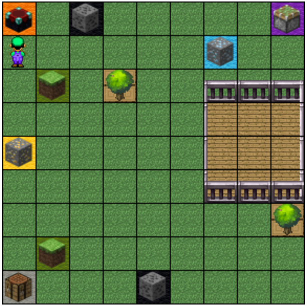
$\varphi' =$ **eventually** used_factory

Example



Detected events
none

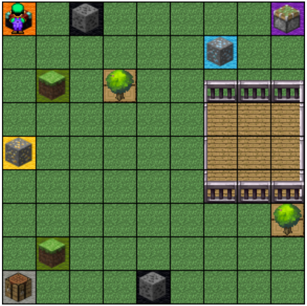
$\varphi' =$ **eventually** `used_factory`



Detected events
none

$\varphi' =$ **eventually** `used_factory`

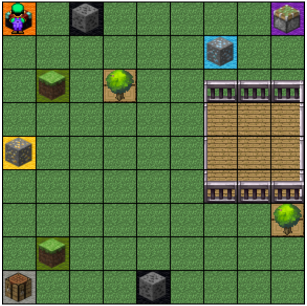
Example



Detected events
`used_factory`

′ = **eventually** `used_factory`

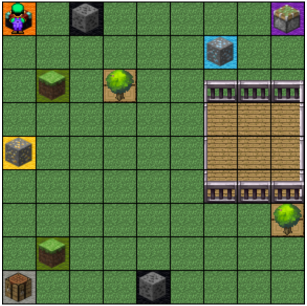
Example



Detected events
`used_factory`

$' =$ **eventually** ~~`used_factory`~~

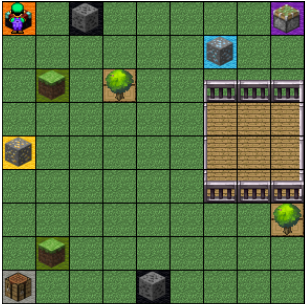
Example



Detected events
`used_factory`

$' = \text{true}$ (+1 reward)

Example

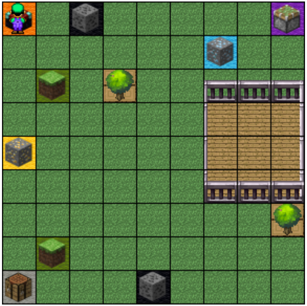


Detected events

`used_factory`

LTL formulas ! Rewards

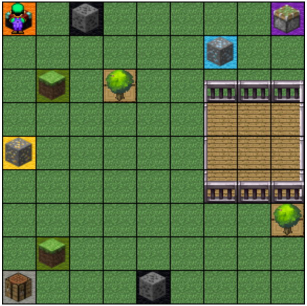
Example



Detected events
`used_factory`

(′ can be learned using standard RL)

Example



Detected events
used_factory

We can do better than this!

Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - **Why (and how) LPOPL speeds up learning of multiple tasks.**
4. Results
5. Concluding remarks

LPOPL: An example

Suppose Luigi has to learn two tasks:

LPOPL: An example

Suppose Luigi has to learn two tasks:

$'_1 =$ **eventually**(`got_iron` **and eventually** `used_factory`) **and eventually** `got_gold`

$'_2 =$ **eventually**([`got_grass` **or** `got_wood`] **and eventually** `used_factory`)

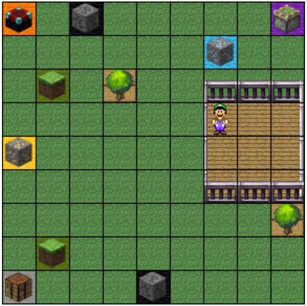
Suppose Luigi has to learn two tasks:

$'_1 =$ **eventually**(`got_iron` **and eventually** `used_factory`) **and eventually** `got_gold`

$'_2 =$ **eventually**([`got_grass` **or** `got_wood`] **and eventually** `used_factory`)

... and begins by trying to solve $'_1$

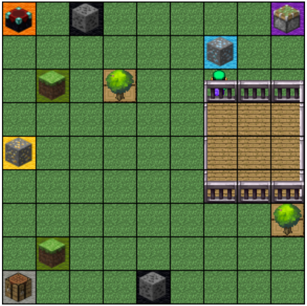
LPOPL: An example



Detected events
at_shelter

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood] **and eventually** used_factory)



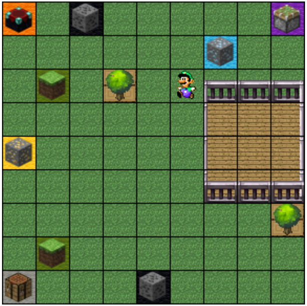
Detected events

none

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood] **and eventually** used_factory)

LPOPL: An example



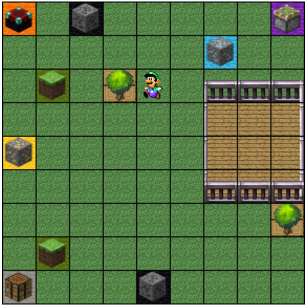
Detected events

none

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood] **and eventually** used_factory)

LPOPL: An example

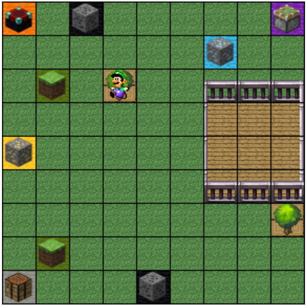


Detected events
none

$'_1 = \textbf{eventually}(\texttt{got\_iron}\ \textbf{and eventually}\ \texttt{used\_factory})$
$\textbf{and eventually}\ \texttt{got\_gold}$

$'_2 = \textbf{eventually}([\texttt{got\_grass}\ \textbf{or}\ \texttt{got\_wood}]\ \textbf{and eventually}$
$\texttt{used\_factory})$

LPOPL: An example

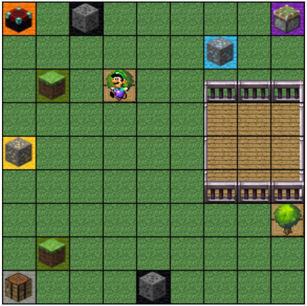


Detected events
got_wood

$'_1$ = **eventually**(got_iron **and eventually** used_factory) **and eventually** got_gold

$'_2$ = **eventually**([got_grass **or** got_wood] **and eventually** used_factory)

LPOPL: An example



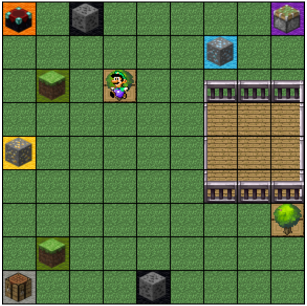
Detected events

got_wood

$'_1 =$ **eventually**(got_iron **and eventually** used_factory) **and eventually** got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood] **and eventually** used_factory)

LPOPL: An example



Detected events
got_wood

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood] **and eventually**
used_factory)

LPOPL learns all the tasks in parallel!

LPOPL overview

Step 1: Decompose the tasks into subtasks using LTL progression.

LPOPL overview

Step 1: Decompose the tasks into subtasks using LTL progression.

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold

$'_2 =$ **eventually**([got_grass **or** got_wood]
and eventually used_factory)

LPOPL overview

Step 1: Decompose the tasks into subtasks using LTL progression.

$'_1 =$ **eventually**(got_iron **and eventually** used_factory)
and eventually got_gold
$'_2 =$ **eventually**([got_grass **or** got_wood]
and eventually used_factory)
$'_3 =$ **eventually**(got_iron **and eventually** used_factory)
$'_4 =$ **eventually** used_factory **and eventually** got_gold
$'_5 =$ **eventually** used_factory
$'_6 =$ **eventually** got_gold
$'_7 =$ true

LPOPL overview

Step 1: Decompose the tasks into subtasks using LTL progression.

$'_1 =$ (got_iron ^ used_factory)
^ got_gold

$'_2 =$ ([got_grass _ got_wood]
^ used_factory)

$'_3 =$ (got_iron ^ used_factory)

$'_4 =$ used_factory ^ got_gold

$'_5 =$ used_factory

$'_6 =$ got_gold

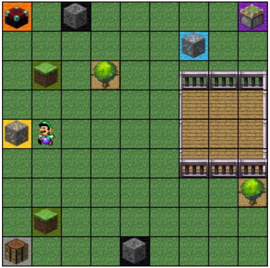
$'_7 =$ true

Step 1: Decompose the tasks into subtasks using LTL progression.

$'_1 =$ (got_iron ^ used_factory)
^ got_gold
$'_2 =$ ([got_grass _ got_wood]
^ used_factory)
$'_3 =$ (got_iron ^ used_factory)
$'_4 =$ used_factory ^ got_gold
$'_5 =$ used_factory
$'_6 =$ got_gold
$'_7 =$ true

Step 2: Learn one policy per subtask using off-policy learning.

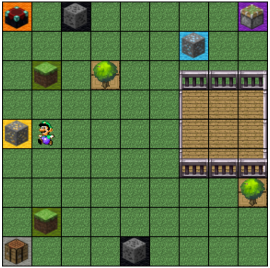
Example



Subtasks

'$_1$ = (got_iron ∧ used_factory) ∧ got_gold
'$_2$ = ([got_grass _ got_wood] ∧ used_factory)
'$_3$ = (got_iron ∧ used_factory)
'$_4$ = used_factory ∧ got_gold
'$_5$ = used_factory
'$_6$ = got_gold

Detected events: none



Subtasks

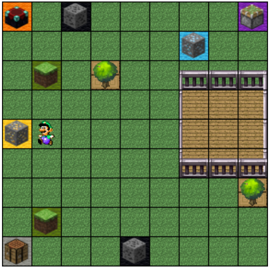
$'_1$ = (got_iron ∧ used_factory) ∧ got_gold
$'_2$ = ([got_grass _ got_wood] ∧ used_factory)
$'_3$ = (got_iron ∧ used_factory)
$'_4$ = used_factory ∧ got_gold
$'_5$ = used_factory
$'_6$ = got_gold

Detected events: none

Q-updates

$Q_{'_1}(s; a)$	$Q_{'_4}(s; a)$
$Q_{'_2}(s; a)$	$Q_{'_5}(s; a)$
$Q_{'_3}(s; a)$	$Q_{'_6}(s; a)$

Example



Subtasks

$'_1$ = (got_iron ^ used_factory) ^ got_gold
$'_2$ = ([got_grass _ got_wood] ^ used_factory)
$'_3$ = (got_iron ^ used_factory)
$'_4$ = used_factory ^ got_gold
$'_5$ = used_factory
$'_6$ = got_gold

Detected events: none

Q-updates

$Q_{'_1}(s; a)$	$Q_{'_4}(s; a)$
$Q_{'_2}(s; a)$	$Q_{'_5}(s; a)$
$Q_{'_3}(s; a)$	$Q_{'_6}(s; a)$

Subtasks

$\varphi_1 = (\texttt{got\_iron} \wedge \texttt{used\_factory}) \wedge \texttt{got\_gold}$

$\varphi_2 = ([\texttt{got\_grass} \vee \texttt{got\_wood}] \wedge \texttt{used\_factory})$

$\varphi_3 = (\texttt{got\_iron} \wedge \texttt{used\_factory})$

$\varphi_4 = \texttt{used\_factory} \wedge \texttt{got\_gold}$

$\varphi_5 = \texttt{used\_factory}$

$\varphi_6 = \texttt{got\_gold}$

Detected events: got_gold

Q-updates

$Q_{\varphi_1}(s;a)$	$Q_{\varphi_4}(s;a)$
$Q_{\varphi_2}(s;a)$	$Q_{\varphi_5}(s;a)$
$Q_{\varphi_3}(s;a)$	$Q_{\varphi_6}(s;a)$

Example

Subtasks

$\varphi_1 =$	$(\texttt{got\_iron} \wedge \texttt{used\_factory}) \wedge \texttt{got\_gold}$
$\varphi_2 =$	$([\texttt{got\_grass} \vee \texttt{got\_wood}] \wedge \texttt{used\_factory})$
$\varphi_3 =$	$(\texttt{got\_iron} \wedge \texttt{used\_factory})$
$\varphi_4 =$	$\texttt{used\_factory} \wedge \texttt{got\_gold}$
$\varphi_5 =$	$\texttt{used\_factory}$
$\varphi_6 =$	$\texttt{got\_gold}$

Detected events: `got_gold`

Q-updates

$Q_{\varphi_1}(s,a)$ $\quad$ $Q_{\varphi_4}(s,a)$

$Q_{\varphi_2}(s,a)$ $\quad$ $Q_{\varphi_5}(s,a)$

$Q_{\varphi_3}(s,a)$ $\quad$ $Q_{\varphi_6}(s,a)$

Example

Subtasks

$'_1 =$ (got_iron $\wedge$ used_factory) $\wedge$ got_gold
$'_2 =$ ([got_grass _ got_wood] $\wedge$ used_factory)
$'_3 =$ (got_iron $\wedge$ used_factory)
$'_4 =$ used_factory $\wedge$ got_gold
$'_5 =$ used_factory
$'_6 =$ got_gold

Detected events: got_gold

Q-updates

$Q_{'_1}(s; a)$	$Q_{'_4}(s; a)$
$Q_{'_2}(s; a)$	$Q_{'_5}(s; a)$
$Q_{'_3}(s; a)$	$Q_{'_6}(s; a)$

Example

Subtasks

$\varphi_1 =$ (got_iron $\wedge$ used_factory) $\wedge$ got_gold
$\varphi_2 =$ ([got_grass $\vee$ got_wood] $\wedge$ used_factory)
$\varphi_3 =$ (got_iron $\wedge$ used_factory)
$\varphi_4 =$ used_factory $\wedge$ got_gold
$\varphi_5 =$ used_factory
$\varphi_6 =$ got_gold

Detected events: got_gold

Q-updates

$Q_{\varphi_1}(s, a)$
$Q_{\varphi_2}(s, a)$
$Q_{\varphi_3}(s, a)$
$Q_{\varphi_4}(s, a) \quad \max_{a'} Q_{\varphi_5}(s', a')$
$Q_{\varphi_5}(s, a)$
$Q_{\varphi_6}(s, a)$

Example

Subtasks

$\varphi_1 =$ (got_iron ∧ used_factory) ∧ got_gold
$\varphi_2 =$ ([got_grass ∨ got_wood] ∧ used_factory)
$\varphi_3 =$ (got_iron ∧ used_factory)
$\varphi_4 =$ used_factory ∧ got_gold
$\varphi_5 =$ used_factory
$\varphi_6 =$ got_gold

Detected events: got_gold

Q-updates

$Q_{\varphi_1}(s;a)$	$Q_{\varphi_4}(s;a) \quad \max_{a'} Q_{\varphi_5}(s';a')$
$Q_{\varphi_2}(s;a)$	$Q_{\varphi_5}(s;a)$
$Q_{\varphi_3}(s;a)$	$Q_{\varphi_6}(s;a)$

Example

Subtasks

$\varphi_1 = (\texttt{got\_iron} \wedge \texttt{used\_factory}) \wedge \texttt{got\_gold}$
$\varphi_2 = ([\texttt{got\_grass} \vee \texttt{got\_wood}] \wedge \texttt{used\_factory})$
$\varphi_3 = (\texttt{got\_iron} \wedge \texttt{used\_factory})$
$\varphi_4 = \texttt{used\_factory} \wedge \texttt{got\_gold}$
$\varphi_5 = \texttt{used\_factory}$
$\varphi_6 = \texttt{got\_gold}$

Detected events: got_gold

Q-updates

$Q_{\varphi_1}(s;a)$		$Q_{\varphi_4}(s;a)$	$\max_{a'} Q_{\varphi_5}(s';a')$
$Q_{\varphi_2}(s;a)$		$Q_{\varphi_5}(s;a)$	
$Q_{\varphi_3}(s;a)$		$Q_{\varphi_6}(s;a)$	1

Example

Subtasks

$'_1 =$ (got_iron $\wedge$ used_factory) $\wedge$ got_gold
$'_2 =$ ([got_grass _ got_wood] $\wedge$ used_factory)
$'_3 =$ (got_iron $\wedge$ used_factory)
$'_4 =$ used_factory $\wedge$ got_gold
$'_5 =$ used_factory
$'_6 =$ got_gold

Detected events: got_gold

Q-updates

$Q_{'_1}(s; a)$	$\max_{a^{\theta}} Q_{'_3}(s^{\theta}; a^{\theta})$	$Q_{'_4}(s; a)$	$\max_{a^{\theta}} Q_{'_5}(s^{\theta}; a^{\theta})$
$Q_{'_2}(s; a)$	$\max_{a^{\theta}} Q_{'_2}(s^{\theta}; a^{\theta})$	$Q_{'_5}(s; a)$	$\max_{a^{\theta}} Q_{'_5}(s^{\theta}; a^{\theta})$
$Q_{'_3}(s; a)$	$\max_{a^{\theta}} Q_{'_3}(s^{\theta}; a^{\theta})$	$Q_{'_6}(s; a)$	1

Example

Subtasks

$'_1 =$ (got_iron ^ used_factory) ^ got_gold
$'_2 =$ ([got_grass _ got_wood] ^ used_factory)
$'_3 =$ (got_iron ^ used_factory)
$'_4 =$ used_factory ^ got_gold
$'_5 =$ used_factory
$'_6 =$ got_gold

Detected events: got_gold

Theorem

LPOPL using tabular q-learning converges to an optimal policy.

Outline

1. Motivation
2. Related work
3. Approach:
    - Why (and how) we use LTL for specifying tasks in RL.
    - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Outline

Experiments

Goal

- Study LPOPL + DQN
- Compare with standard RL
- Compare with alternative decomposition methods for RL

Experiments

Goal

- Study LPOPL + DQN
- Compare with standard RL
- Compare with alternative decomposition methods for RL

Baselines

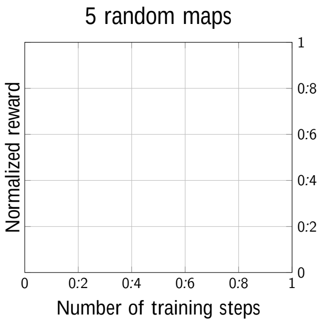
- DQN-L: Standard DQN[1] (no decomposition).
- HRL-E: Hierarchical Deep RL proposed by Kulkarni et al.[2]
- HRL-L: HRL-E but exploiting LTL to prune *useless* options.

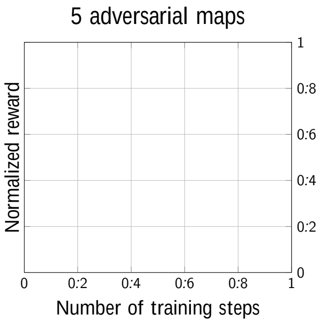
[1] Human-level control through deep reinforcement learning (Nature-15)
[2] HDRL: Integrating Temporal Abstraction and Intrinsic Motivation (NIPS-16)

Experiment 1: Interleaving subtasks

Description

Tasks from (Andreas et al., 2017) w/o unneeded order constraints.
e.g. (get iron and get wood), then use factory.

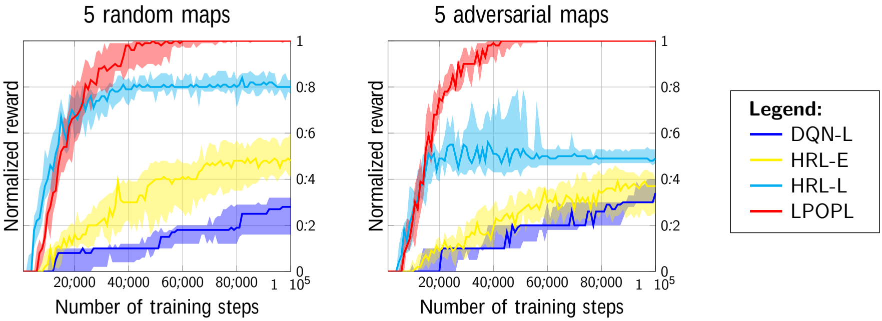




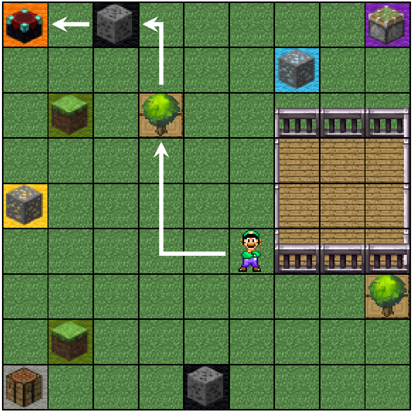
Experiment 1: Interleaving subtasks

Description

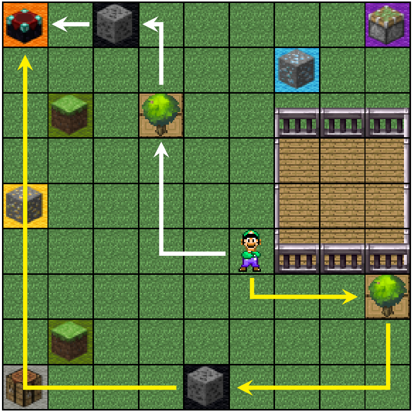
Tasks from (Andreas et al., 2017) w/o unneeded order constraints. e.g. (get iron and get wood), then use factory.



Hierarchical RL might converge to suboptimal policies



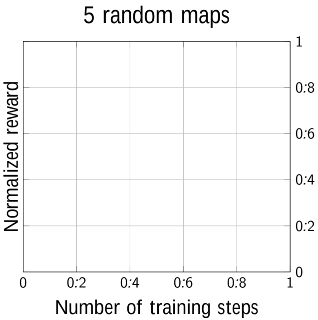
Hierarchical RL might converge to suboptimal policies

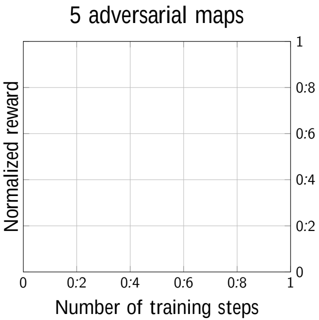


Experiment 2: Safety constraints

Description

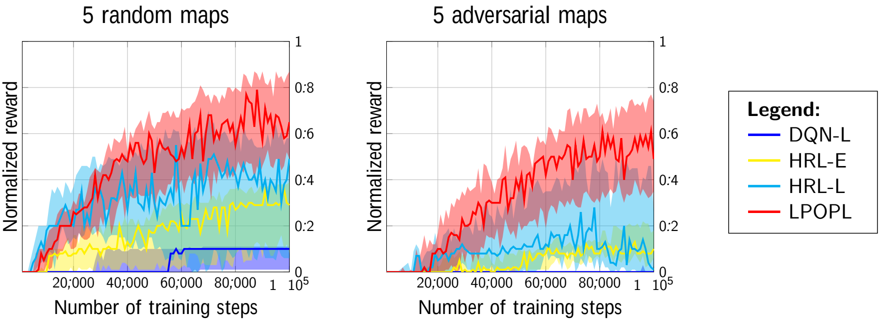
Same set of 10 tasks but including the safety constraint of being at the shelter during the night.





Description

Same set of 10 tasks but including the safety constraint of being at the shelter during the night.



Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Outline

1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. **Concluding remarks**

Concluding remarks

Question: How can we instruct RL agents?

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Desirable properties:

- It is expressive.

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Desirable properties:

- It is expressive.
- RL agents understand it:
  - Task description *!* Reward function.
  - Task description *!* Learn faster.

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Desirable properties:

- It is expressive.
- RL agents understand it:
  - Task description ! Reward function.
  - Task description ! Learn faster.
- Humans understand it.

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Desirable properties:

- It is expressive.
- RL agents understand it:
  - Task description ! Reward function.
  - Task description ! Learn faster.
- Humans understand it.
  - NL ! LTL by Dzifcak et al. (ICRA-09).

Concluding remarks

Question: How can we instruct RL agents?
Proposal: Define tasks using Linear Temporal Logic.

Desirable properties:

- It is expressive.
- RL agents understand it:
  - Task description ! Reward function.
  - Task description ! Learn faster.
- Humans understand it.
  - NL ! LTL by Dzifcak et al. (ICRA-09).

Code: `https://bitbucket.org/RToroIcarte/lpopl`