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

AAMAS 2018

July 11

Outline

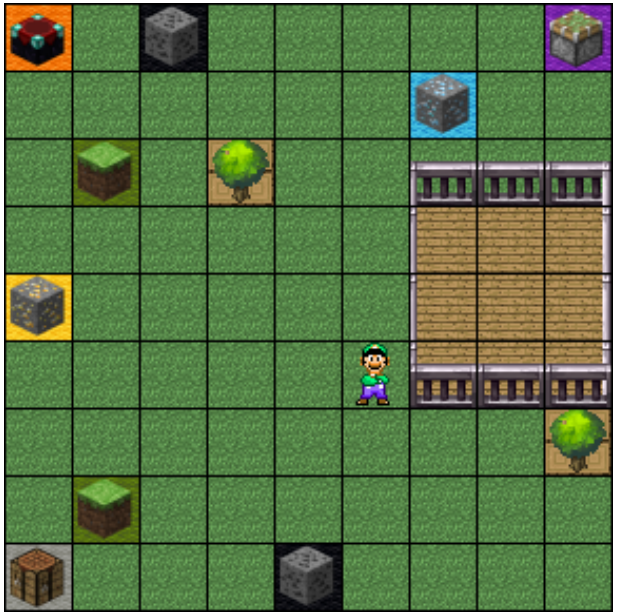
1. Motivation
2. Related work
3. Approach:
   - Why (and how) we use LTL for specifying tasks in RL.
   - Why (and how) LPOPL speeds up learning of multiple tasks.
4. Results
5. Concluding remarks

Outline

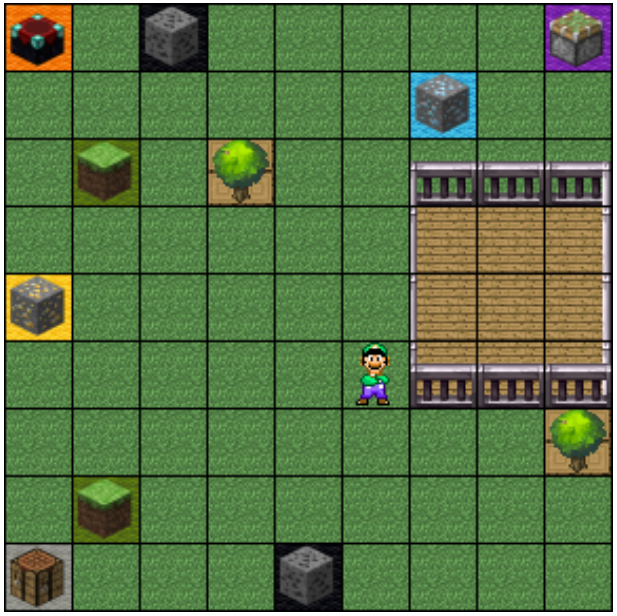
Motivation

How do you describe to an RL agent what to do?

Running example



Running example



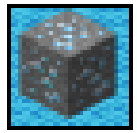
Luigi can collect raw materials:


wood

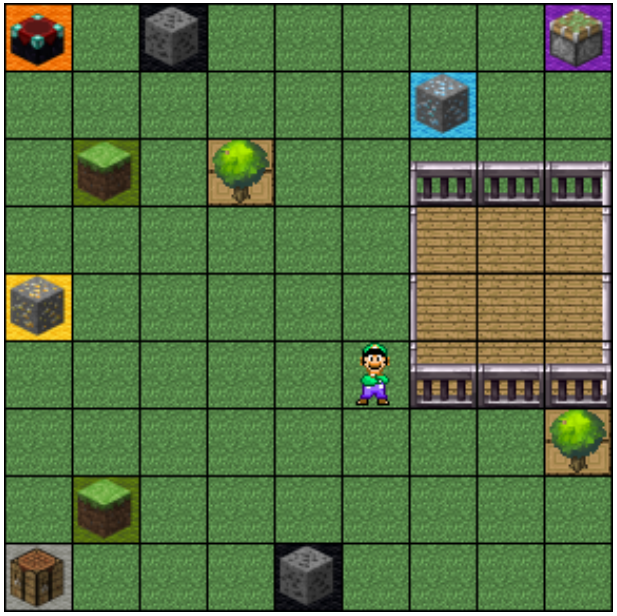

grass


iron


gold


gems

Running example



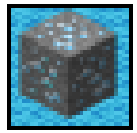
Luigi can collect raw materials:


wood


grass


iron


gold


gems

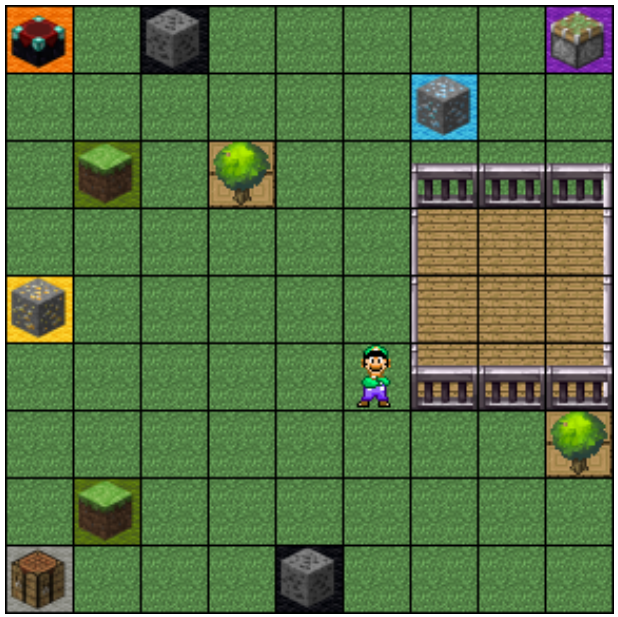
... and make new objects in:


factory


toolshed


workbench

Running example



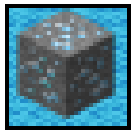
Luigi can collect raw materials:


wood


grass

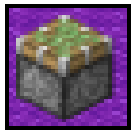

iron


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 do you describe to an RL agent what to do?

Motivation

How do you describe to an RL agent what to do?

Description $\neq$ Reward function

Motivation

How do you describe to an RL agent what to do?

Description $\not\Rightarrow$ Reward function

Language $\not\Rightarrow$ Reward function

Motivation

How do you describe to an RL agent what to do?

Description $\not\Rightarrow$ Reward function

Language $\not!$ Reward function

Why would we want such a language?

Motivation

How do you describe to an RL agent what to do?

Description $\not\Rightarrow$ Reward function

Language $\not!$ Reward function

Why would we want such a language?

To define new task faster.

To transfer learning between tasks.

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

Related works

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

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

Related works

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	

Related works

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.

Related works

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		

This 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.

This 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

This 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

Related works

Other works using variants of LTL in RL:

- Reinforcement learning with temporal logic rewards
  by Li et al. (IROS, 2017)
- Environment-independent task specifications via GLTL
  by Littman et al. (arXiv, 2017)
- Logically constrained reinforcement learning
  by Hasanbeig et al. (arXiv, 2018)

Related works

Other works using variants of LTL in RL:

- Reinforcement learning with temporal logic rewards
  by Li et al. (IROS, 2017)
- Environment-independent task specifications via GLTL
  by Littman et al. (arXiv, 2017)
- Logically constrained reinforcement learning
  by Hasanbeig et al. (arXiv, 2018)

We exploit task decomposition and off-policy RL.

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

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

De ning tasks using LTL

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

Example

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

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

De ning tasks using LTL

Linear Temporal Logic (syntax)

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

$$' ::= p \; j \; : ' \; j \; ' _1 \; \hat{} \; ' _2 \; j \qquad ' \; j \qquad ' \; j \; ' _1 \, \mathrm{U} \; ' _2 \text{ with } p \; 2 \; P$$

De ning tasks using LTL

Linear Temporal Logic (syntax)

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

$$' ::= p \; j \; : ' \; j \; ' _1 \; \hat{} \; ' _2 \; j \qquad ' \; j \qquad ' \; j \; ' _1 \, \mathrm{U} \, ' _2 \text{ with } p \; 2 \; P$$

Examples

got_wood
(got_grass ^ used_factory)
got_wood _ got_iron
got_grass ^ got_iron
(is_night ! at_shelter) U got_wood

De ning tasks using LTL

Linear Temporal Logic (syntax)

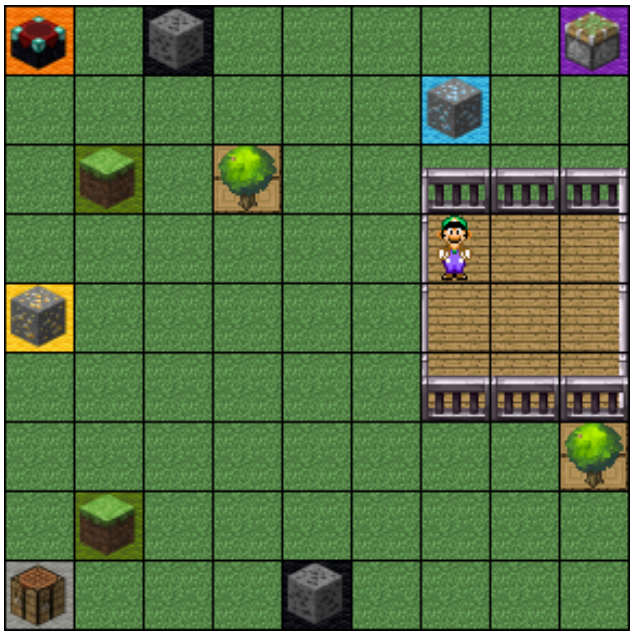
LTL augments propositional logic with temporal operators (*next*), (*eventually*), and U (*until*):

$$' ::= p \; j \; : ' \; j \; ' _1 \; \hat{} \; ' _2 \; j \qquad ' \; j \qquad ' \; j \; ' _1 \, \mathrm{U} \, ' _2 \text{ with } p \; 2 \; P$$

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 *!* at_shelter) **until** got_wood

Example

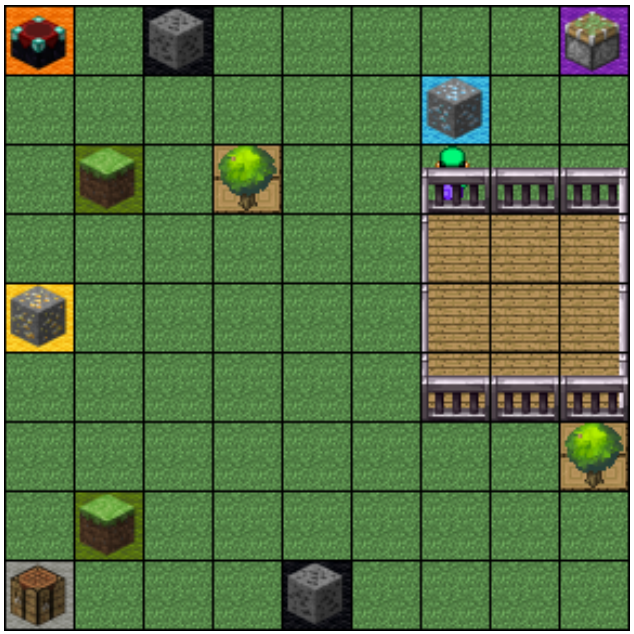


Detected events

at_shelter

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

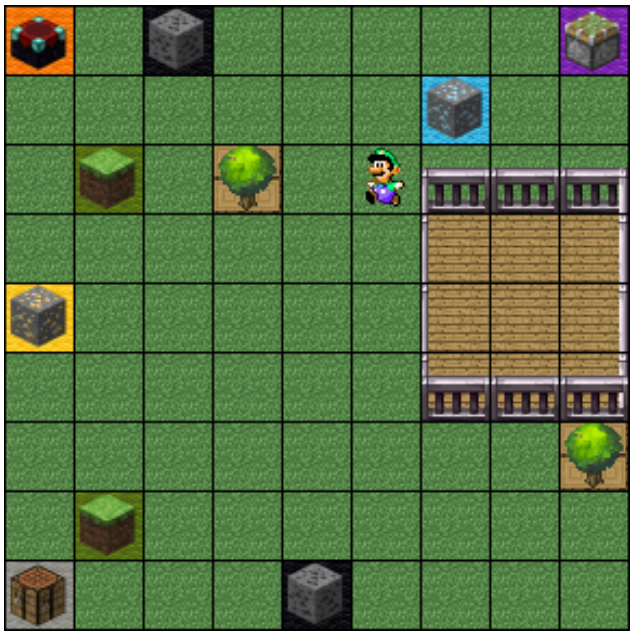
Example



Detected events
none

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

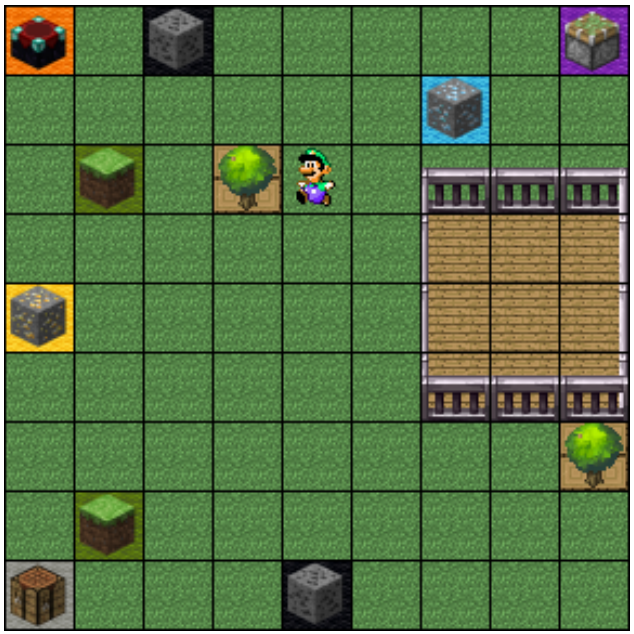
Example



Detected events
none

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

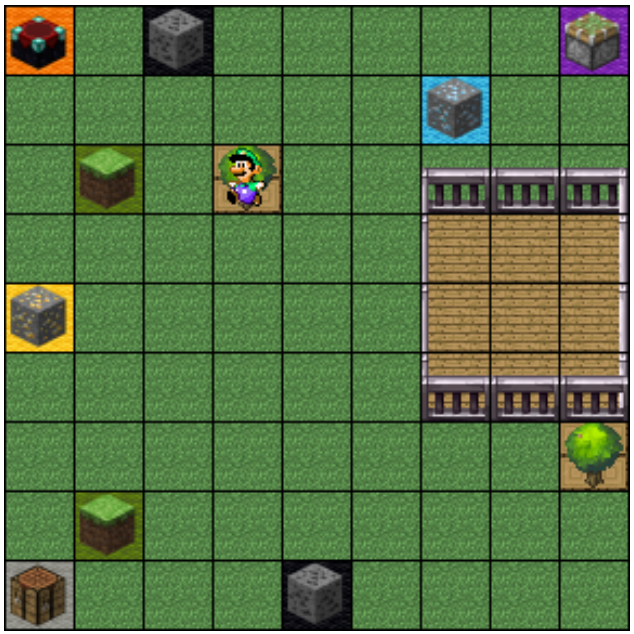
Example



Detected events
none

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

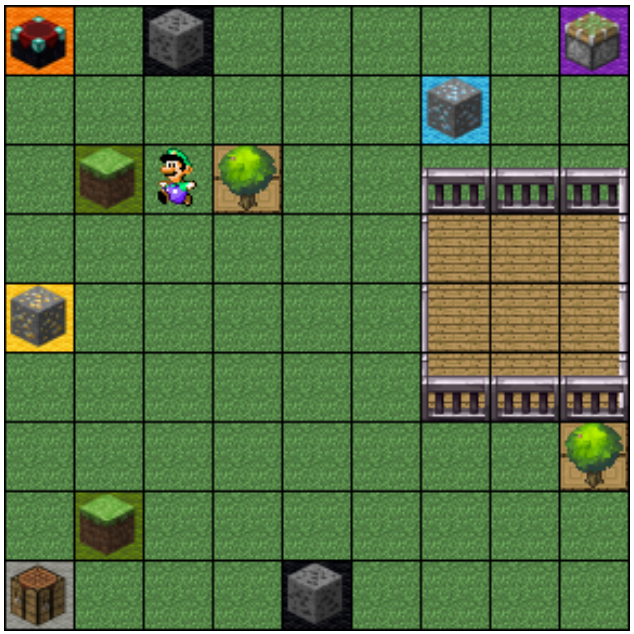
Example



Detected events
got_wood

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

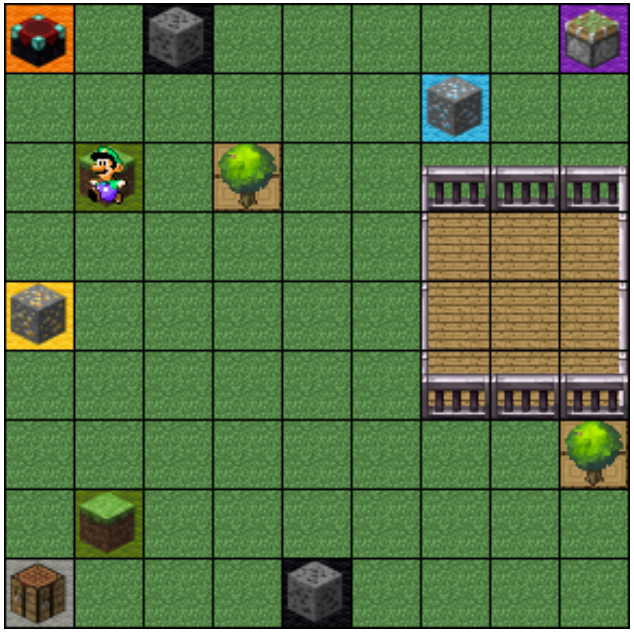
Example



Detected events
none

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

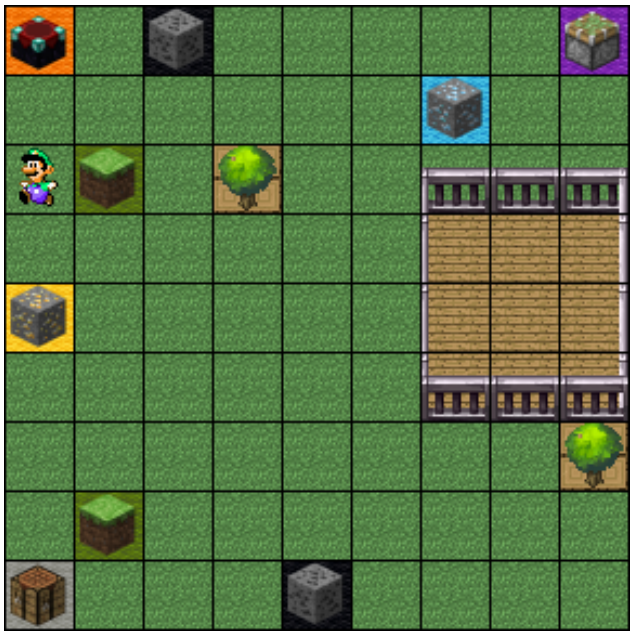
Example



Detected events
got_grass

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

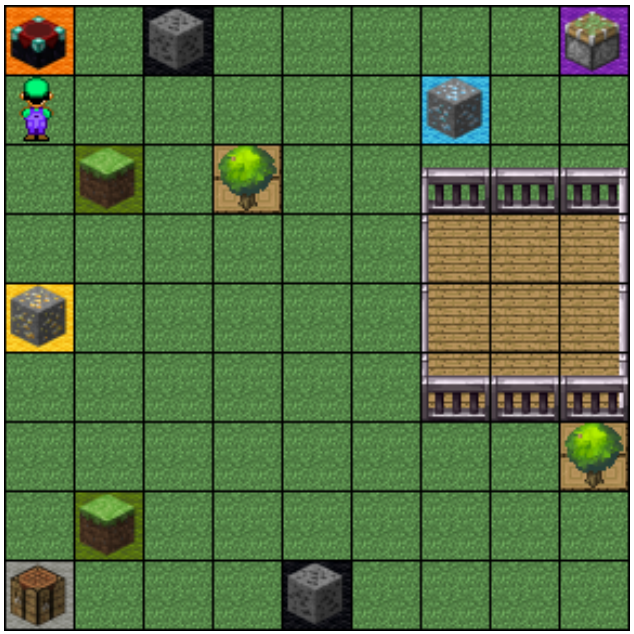
Example



Detected events
none

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

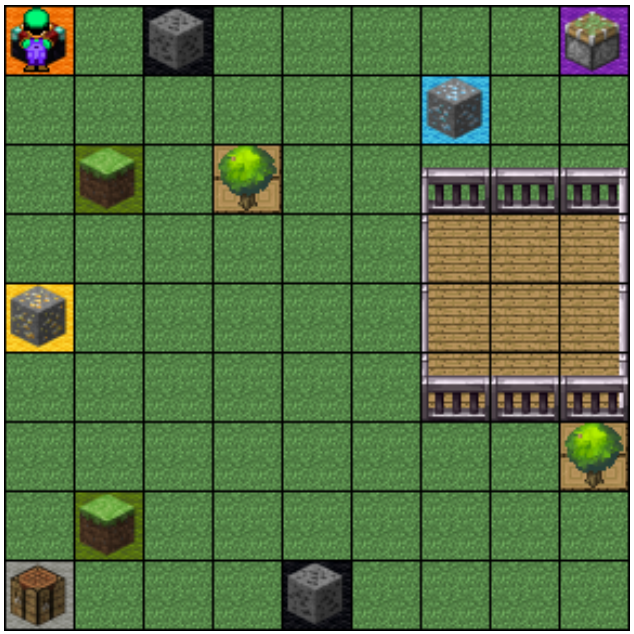
Example



Detected events
none

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

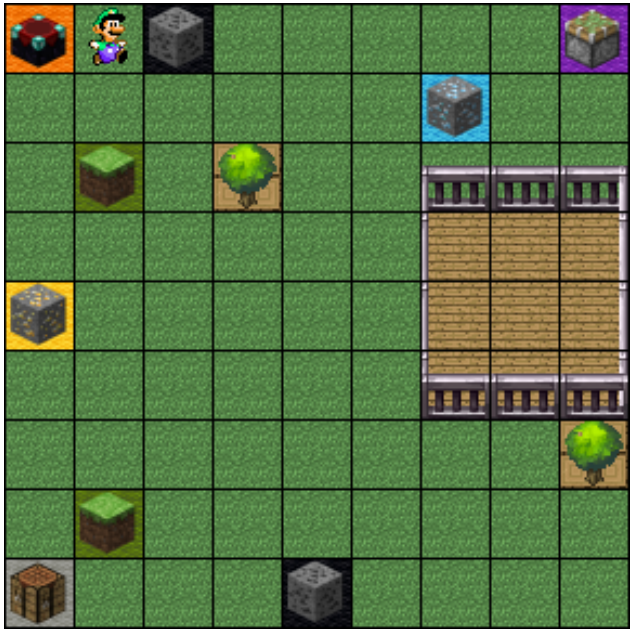
Example



Detected events
used_factory

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

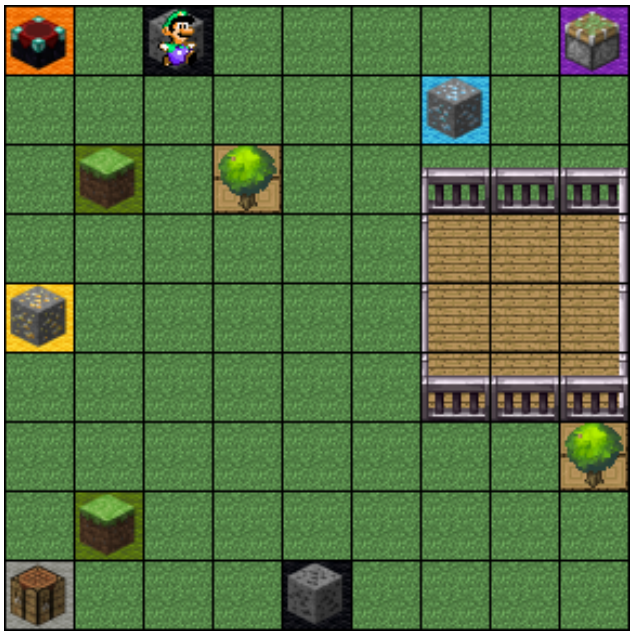
Example



Detected events
none

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

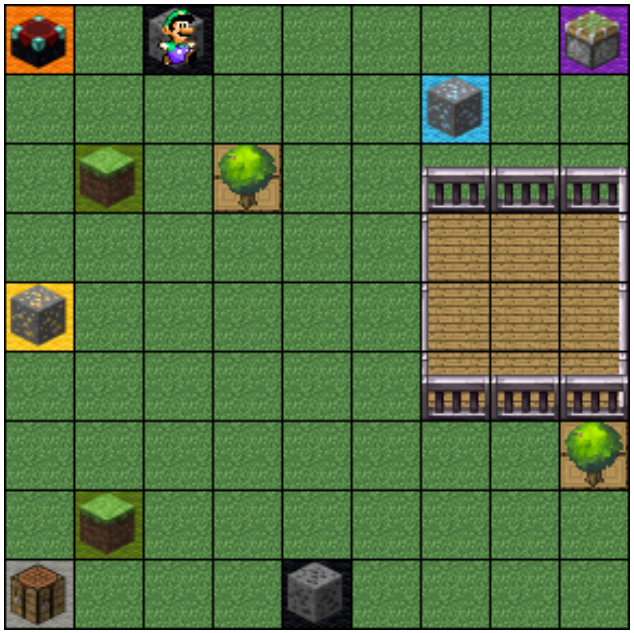
Example



Detected events
got_iron

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

Example



Detected events

got_iron

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

LTL progression

LTL progression

Given an LTL formula $'$ and state s, we can *progress* $'$ 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 $p \in P$
- $\text{prog}(s; :') = : \text{prog}(s; ')$
- $\text{prog}(s; '_1 \wedge '_2) = \text{prog}(s; '_1) \wedge \text{prog}(s; '_2)$
- $\text{prog}(s; \bigcirc') = '$
- $\text{prog}(s; \square') = \text{prog}(s; ') \_ \square'$
- $\text{prog}(s; '_1 \,\mathrm{U}\, '_2) = \text{prog}(s; '_2) \_ (\text{prog}(s; '_1) \wedge '_1 \,\mathrm{U}\, '_2)$

LTL progression

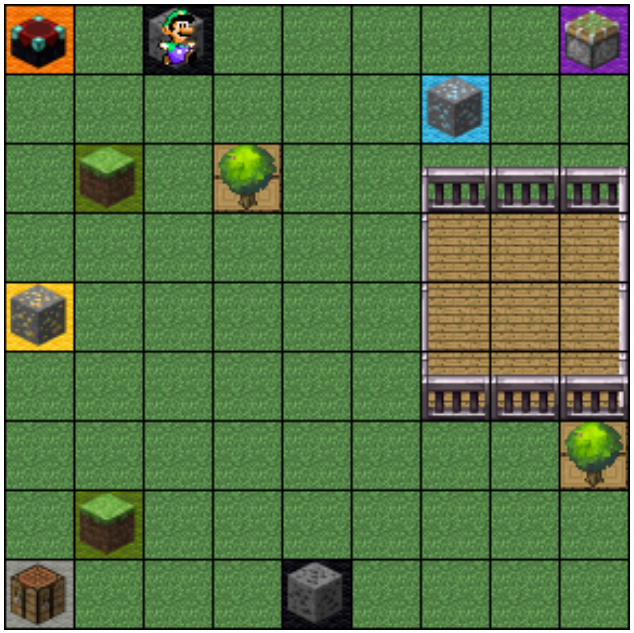
LTL progression

Given an LTL formula $'$ and state s, we can *progress* $'$ 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 $p \in P$
- $\text{prog}(s; :') = : \text{prog}(s; ')$
- $\text{prog}(s; '_1 \wedge '_2) = \text{prog}(s; '_1) \wedge \text{prog}(s; '_2)$
- $\text{prog}(s; \bigcirc') = '$
- $\text{prog}(s; \square') = \text{prog}(s; ') \_ \square'$
- $\text{prog}(s; '_1 \,\mathsf{U}\, '_2) = \text{prog}(s; '_2) \_ (\text{prog}(s; '_1) \wedge '_1 \,\mathsf{U}\, '_2)$

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

Example



Detected events

got_iron

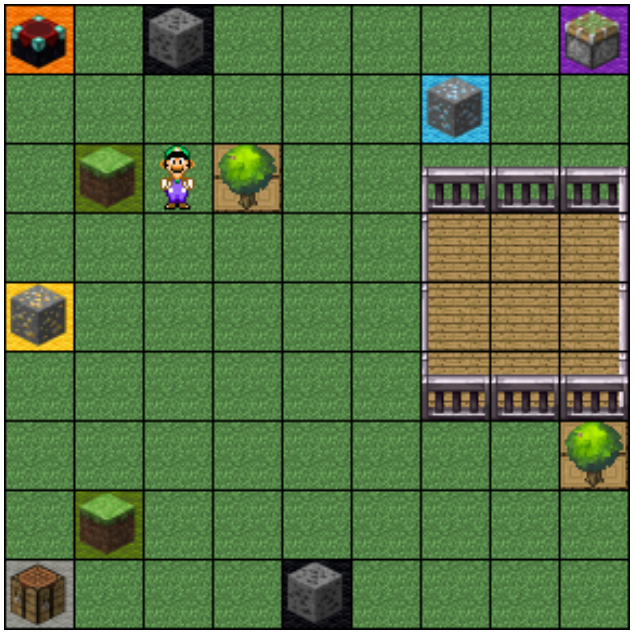
$' =$ **eventually** used_factory and **eventually** got_gold

Example

Detected events
none

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

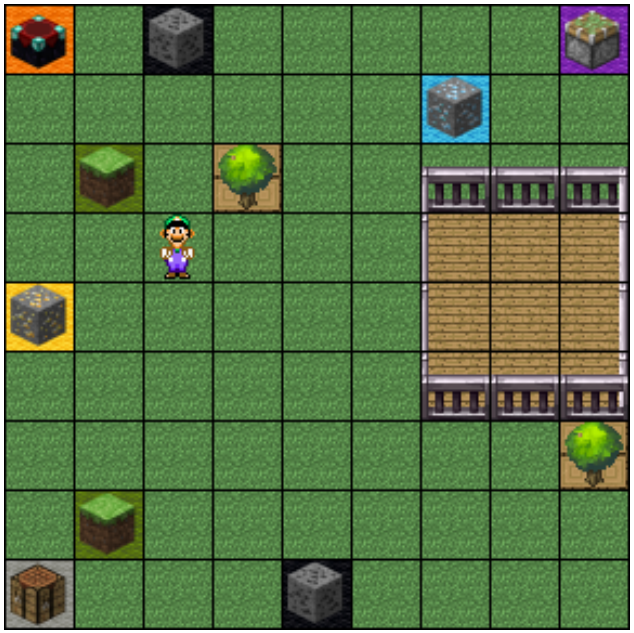
Example



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

$$' = \text{eventually used\_factory} \quad \text{and} \quad \text{eventually got\_gold}$$

Example

Detected events
none

$$' = \text{eventually used\_factory} \quad \text{and} \quad \text{eventually got\_gold}$$

Example

Detected events
none

$$' = \text{eventually used\_factory} \quad \text{and} \quad \text{eventually got\_gold}$$

Example

Detected events
none

$$' = \text{eventually used\_factory} \quad \text{and} \quad \text{eventually got\_gold}$$

Example

$$\frac{\text{Detected events}}{\text{got\_gold}}$$

$$' = \text{eventually used\_factory} \quad \text{and} \quad \text{eventually got\_gold}$$

Example

$$\frac{\text{Detected events}}{\text{got\_gold}}$$

$$' = \text{eventually used\_factory}$$

Example

Detected events
none

' = eventually used_factory

Example

Detected events
none

' = eventually used_factory

Example

$$\frac{\text{Detected events}}{\text{got\_grass}}$$

$$' = \text{eventually used\_factory}$$

Example

Detected events
none

' = eventually used_factory

Example

Detected events
none

' = eventually used_factory

Example

$$\frac{\text{Detected events}}{\text{used\_factory}}$$

$$' = \text{eventually used\_factory}$$

Example

$$\frac{\text{Detected events}}{\text{used\_factory}}$$

' = true (+1 reward)

Example

Detected events

used_factory

LTL formulas! Rewards

Example

$$\frac{\text{Detected events}}{\text{used\_factory}}$$

(' 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:

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass } \text{or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Suppose Luigi has to learn two tasks:

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

... and begins by trying to solve φ_1

LPOPL: An example

Detected events

at_shelter

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
none

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
none

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
none

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
got_wood

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
got_wood

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

LPOPL: An example

Detected events
got_wood

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

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.

$$\varphi_1 = \text{eventually}(\text{got\_iron} \text{ and } \text{eventually } \text{used\_factory}) \text{ and } \text{eventually } \text{got\_gold}$$

$$\varphi_2 = \text{eventually}([\text{got\_grass} \text{ or } \text{got\_wood}] \text{ and } \text{eventually } \text{used\_factory})$$

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.

$$
\begin{aligned}
\varphi_1 &= (\text{got\_iron} \ \hat{} \ \text{used\_factory}) \\
&\quad \hat{} \ \text{got\_gold} \\
\varphi_2 &= ([\text{got\_grass} \ \_ \ \text{got\_wood}] \\
&\quad \hat{} \ \text{used\_factory}) \\
\varphi_3 &= (\text{got\_iron} \ \hat{} \ \text{used\_factory}) \\
\varphi_4 &= \text{used\_factory} \ \hat{} \ \text{got\_gold} \\
\varphi_5 &= \text{used\_factory} \\
\varphi_6 &= \text{got\_gold} \\
\varphi_7 &= \text{true}
\end{aligned}
$$

LPOPL overview

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

$$
\begin{aligned}
\varphi_1 &= (\text{got\_iron} \wedge \text{used\_factory}) \\
&\quad \wedge \text{got\_gold} \\
\varphi_2 &= ([\text{got\_grass} \vee \text{got\_wood}] \\
&\quad \wedge \text{used\_factory}) \\
\varphi_3 &= (\text{got\_iron} \wedge \text{used\_factory}) \\
\varphi_4 &= \text{used\_factory} \wedge \text{got\_gold} \\
\varphi_5 &= \text{used\_factory} \\
\varphi_6 &= \text{got\_gold} \\
\varphi_7 &= \text{true}
\end{aligned}
$$

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

LPOPL using tabular q-learning

Standard q-learning update given experience (s, a, r, s'):

$$Q(s, a) \leftarrow r + \max_{a'} Q(s', a')$$

... where $x \leftarrow y$ is a shorthand notation for $x \leftarrow x + \alpha(y - x)$.

LPOPL using tabular q-learning

Standard q-learning update given experience (s, a, r, s'):

$$Q(s, a) \leftarrow r + \max_{a'} Q(s', a')$$

... where $x \leftarrow y$ is a shorthand notation for $x \leftarrow x + \alpha(y - x)$.

LPOPL update using q-learning given experience (s, a, r, s'):

$$Q_{\varphi}(s, a) \leftarrow r_{\varphi} + \max_{a'} Q_{\varphi'}(s', a')$$

... where $\varphi' = \text{prog}(s', \varphi)$ and $r_{\varphi} = 1$ if $\varphi \neq \text{false}$ and $\varphi' = \text{true}$.

LPOPL using tabular q-learning

Standard q-learning update given experience (s, a, r, s'):

$$Q(s, a) \leftarrow r + \max_{a'} Q(s', a')$$

... where $x \leftarrow y$ is a shorthand notation for $x \leftarrow x + \alpha(y - x)$.

LPOPL update using q-learning given experience (s, a, r, s'):

$$Q_{\varphi}(s, a) \leftarrow r_{\varphi} + \max_{a'} Q_{\varphi'}(s', a')$$

... where $\varphi' = \text{prog}(s', \varphi)$ and $r_{\varphi} = 1$ if $\varphi \neq \varphi' = \text{true}$.

Theorem

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

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: at_shelter

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: got_wood

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: got_wood

Q-updates

$Q_{\phi_1}(s;a)$	$\max_{a'} Q_{\phi_1}(s';a')$	$Q_{\phi_4}(s;a)$	$\max_{a'} Q_{\phi_4}(s';a')$
$Q_{\phi_2}(s;a)$	$\max_{a'} Q_{\phi_5}(s';a')$	$Q_{\phi_5}(s;a)$	$\max_{a'} Q_{\phi_5}(s';a')$
$Q_{\phi_3}(s;a)$	$\max_{a'} Q_{\phi_3}(s';a')$	$Q_{\phi_6}(s;a)$	$\max_{a'} Q_{\phi_6}(s';a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: got_grass

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: got_grass

Q-updates

$Q_{\phi_1}(s;a)$	$\max_{a'} Q_{\phi_1}(s';a')$	$Q_{\phi_4}(s;a)$	$\max_{a'} Q_{\phi_4}(s';a')$
$Q_{\phi_2}(s;a)$	$\max_{a'} Q_{\phi_5}(s';a')$	$Q_{\phi_5}(s;a)$	$\max_{a'} Q_{\phi_5}(s';a')$
$Q_{\phi_3}(s;a)$	$\max_{a'} Q_{\phi_3}(s';a')$	$Q_{\phi_6}(s;a)$	$\max_{a'} Q_{\phi_6}(s';a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

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: used_factory

Q-updates

$Q'_1(s,a)$	$\max_{a'} Q'_1(s',a')$	$Q'_4(s,a)$	$\max_{a'} Q'_6(s',a')$
$Q'_2(s,a)$	$\max_{a'} Q'_2(s',a')$	$Q'_5(s,a)$	1
$Q'_3(s,a)$	$\max_{a'} Q'_3(s',a')$	$Q'_6(s,a)$	$\max_{a'} Q'_6(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: used_factory

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	1
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: used_factory

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	1
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_1}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_3}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: got_iron

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks	
ϕ_1 =	(got_iron ^ used_factory) ^ got_gold
ϕ_2 =	([got_grass _ got_wood] ^ used_factory)
ϕ_3 =	(got_iron ^ used_factory)
ϕ_4 =	used_factory ^ got_gold
ϕ_5 =	used_factory
ϕ_6 =	got_gold

Detected events: got_iron

Q-updates			
$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

Example

Subtasks

ϕ_1 = (got_iron ^ used_factory) ^ got_gold
ϕ_2 = ([got_grass _ got_wood] ^ used_factory)
ϕ_3 = (got_iron ^ used_factory)
ϕ_4 = used_factory ^ got_gold
ϕ_5 = used_factory
ϕ_6 = got_gold

Detected events: got_iron

Q-updates

$Q_{\phi_1}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$	$Q_{\phi_4}(s,a)$	$\max_{a'} Q_{\phi_4}(s',a')$
$Q_{\phi_2}(s,a)$	$\max_{a'} Q_{\phi_2}(s',a')$	$Q_{\phi_5}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$
$Q_{\phi_3}(s,a)$	$\max_{a'} Q_{\phi_5}(s',a')$	$Q_{\phi_6}(s,a)$	$\max_{a'} Q_{\phi_6}(s',a')$

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: got_iron

Q-updates

$Q'_1(s,a)$	$\max_{a'} Q'_4(s',a')$	$Q'_4(s,a)$	$\max_{a'} Q'_4(s',a')$
$Q'_2(s,a)$	$\max_{a'} Q'_2(s',a')$	$Q'_5(s,a)$	$\max_{a'} Q'_5(s',a')$
$Q'_3(s,a)$	$\max_{a'} Q'_5(s',a')$	$Q'_6(s,a)$	$\max_{a'} Q'_6(s',a')$

Example

Subtasks		
ϕ_1	=	(got_iron ^ used_factory) ^ got_gold
ϕ_2	=	([got_grass _ got_wood] ^ used_factory)
ϕ_3	=	(got_iron ^ used_factory)
ϕ_4	=	used_factory ^ got_gold
ϕ_5	=	used_factory
ϕ_6	=	got_gold

Detected events: none

Q-updates

$Q_{\phi_1}(s;a)$	$\max_{a'} Q_{\phi_1}(s';a')$	$Q_{\phi_4}(s;a)$	$\max_{a'} Q_{\phi_4}(s';a')$
$Q_{\phi_2}(s;a)$	$\max_{a'} Q_{\phi_2}(s';a')$	$Q_{\phi_5}(s;a)$	$\max_{a'} Q_{\phi_5}(s';a')$
$Q_{\phi_3}(s;a)$	$\max_{a'} Q_{\phi_3}(s';a')$	$Q_{\phi_6}(s;a)$	$\max_{a'} Q_{\phi_6}(s';a')$

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

Experiments

Goal

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

Experiments

Goal

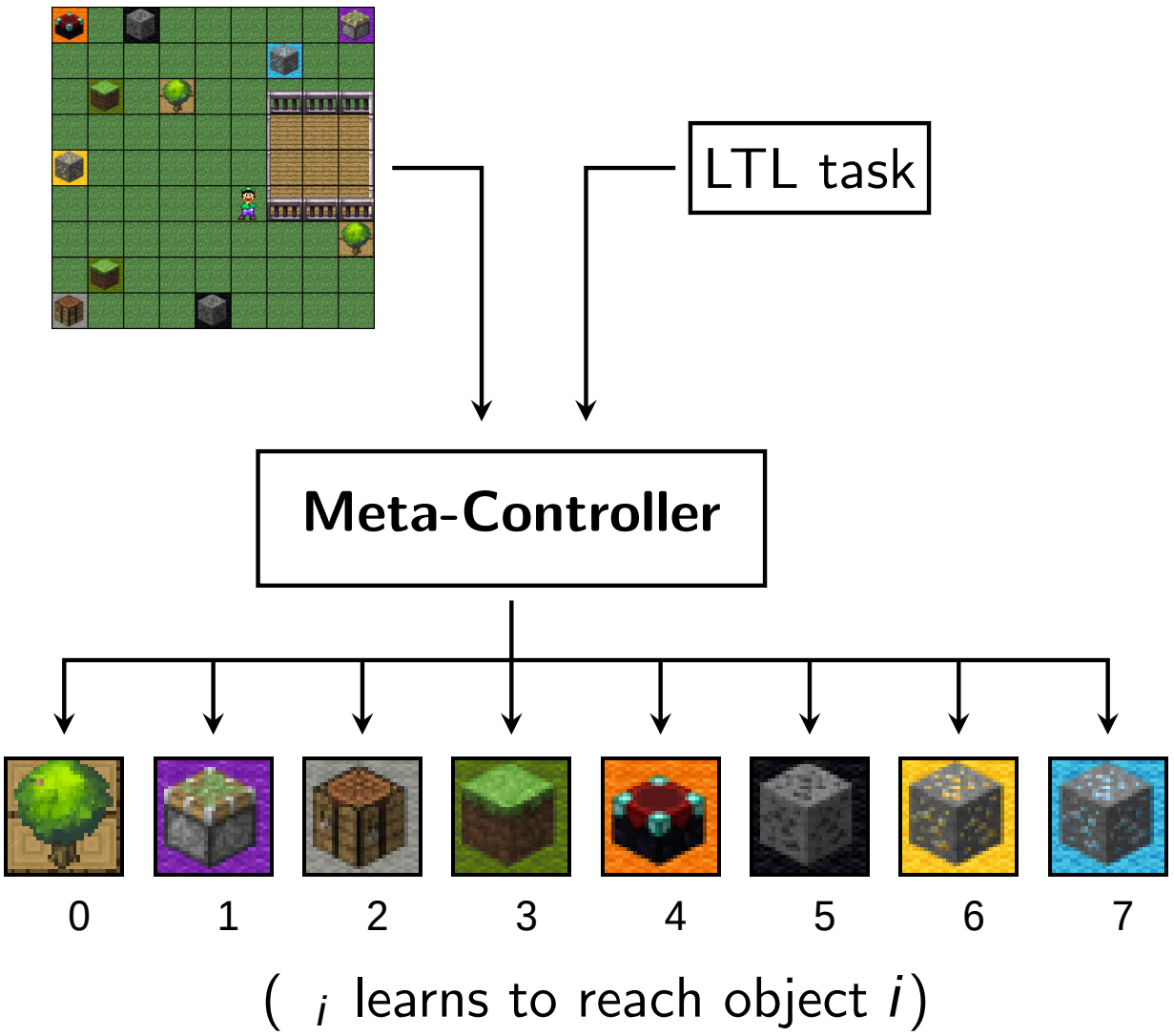
- Study LPOPL + DQN
- Compare with standard RL
- Compare with alternative decomposition methods for RL

Baselines

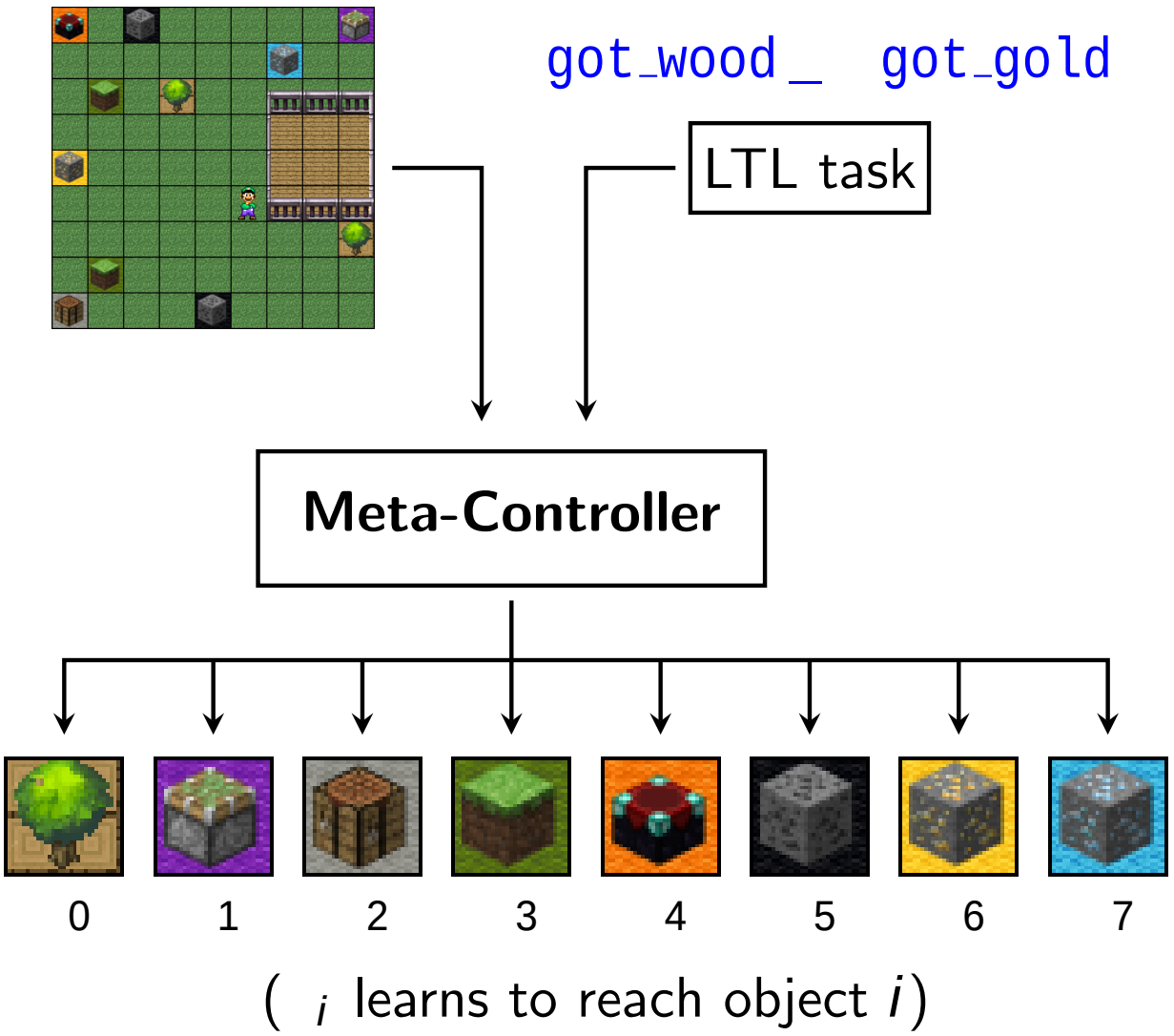
- DQN-L: Standard DQN[1] (no decomposition).
- HRL-E: Hierarchical Deep RL proposed by Kulkarni et al.[2]
- HRL-L: HRL-E but exploiting LTL to prune useless options.

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

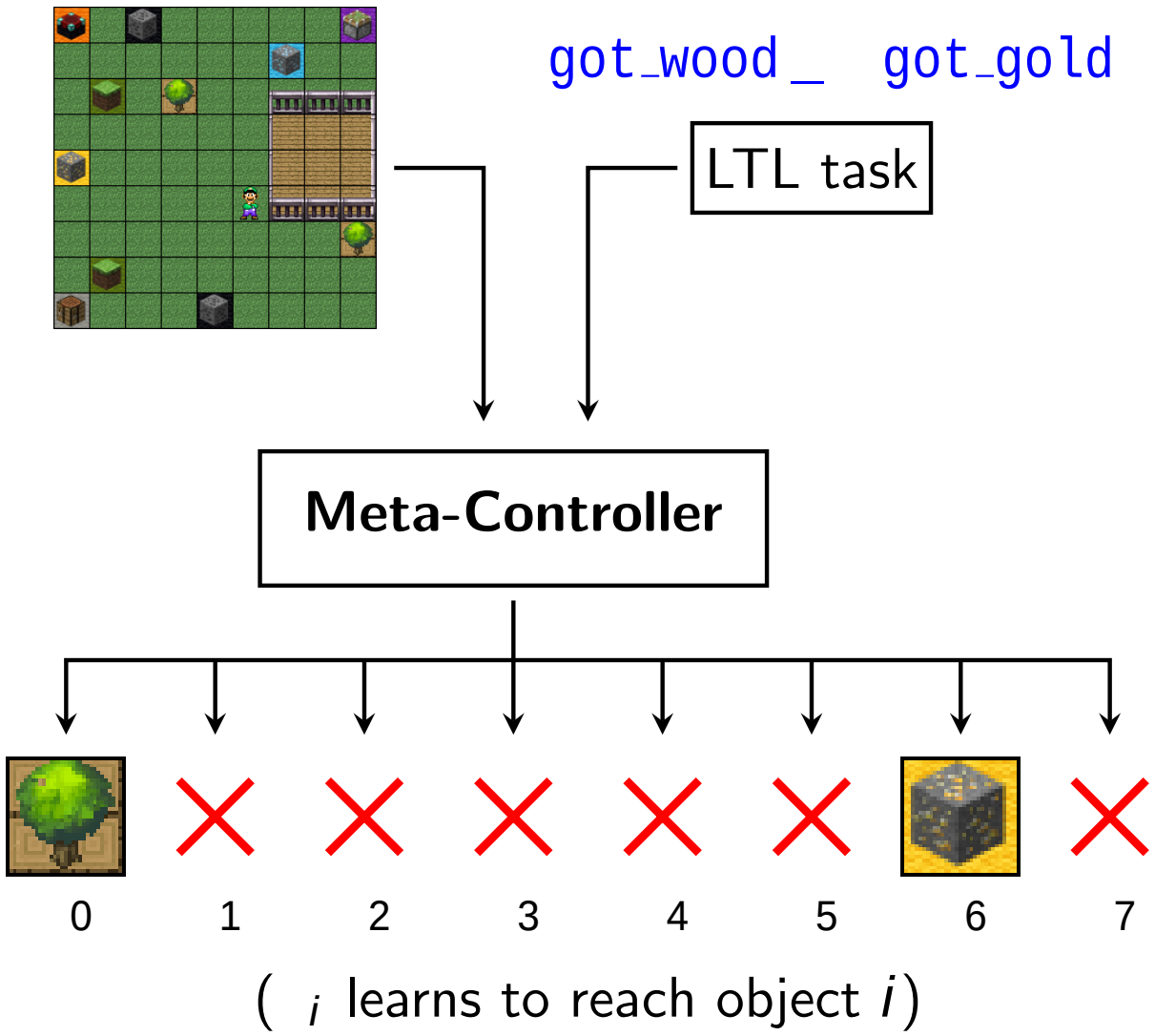
Baselines: HRL-E



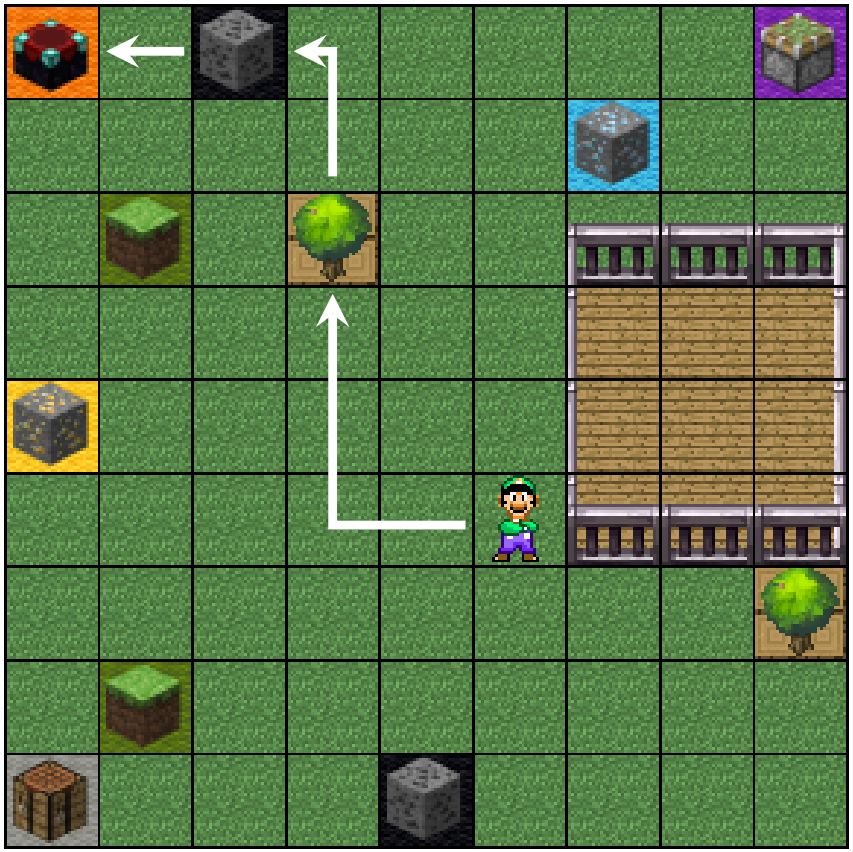
Baselines: HRL-L



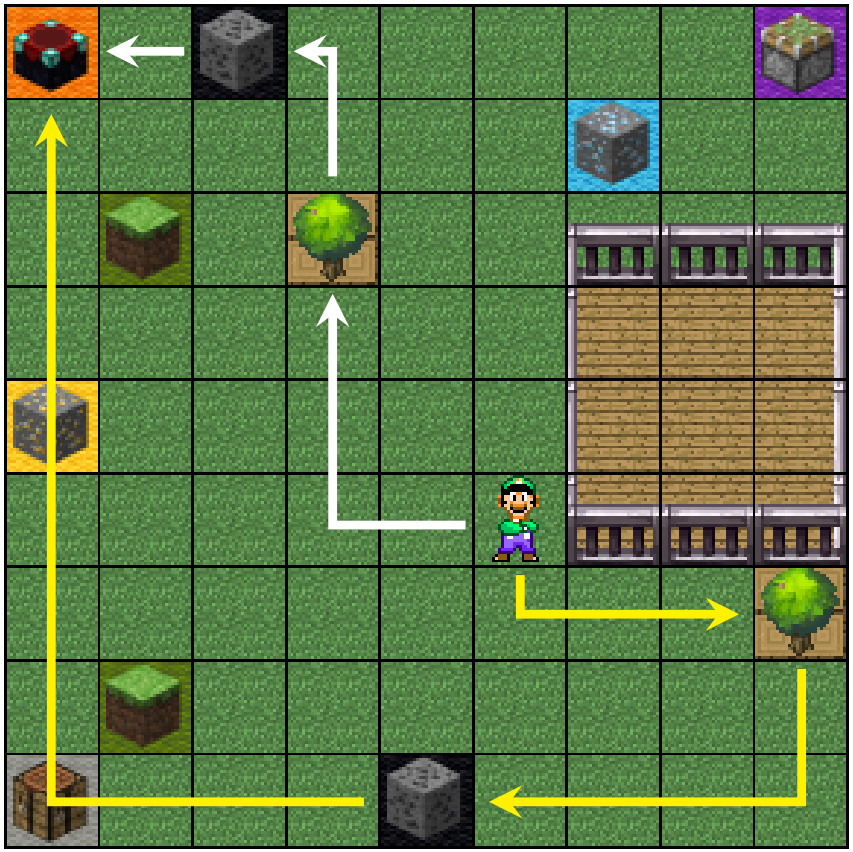
Baselines: HRL-L



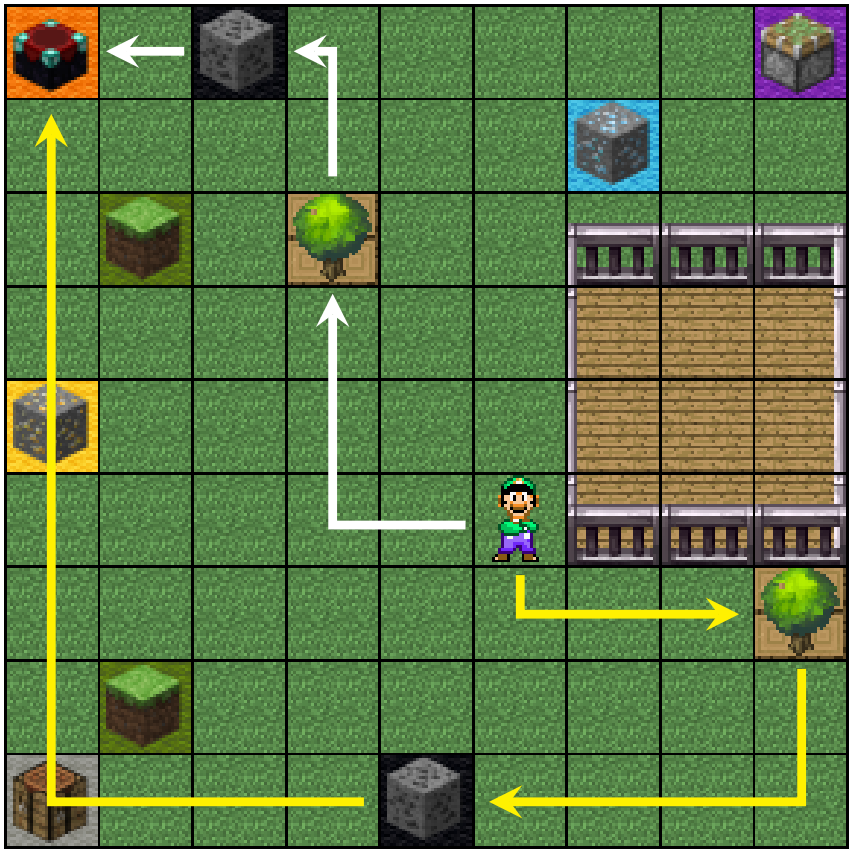
Hierarchical RL might converge to suboptimal policies



Hierarchical RL might converge to suboptimal policies



Hierarchical RL might converge to suboptimal policies



We tested over 5 random grids and 5 adversarial grids.

Experiment 1: subgoal sequences

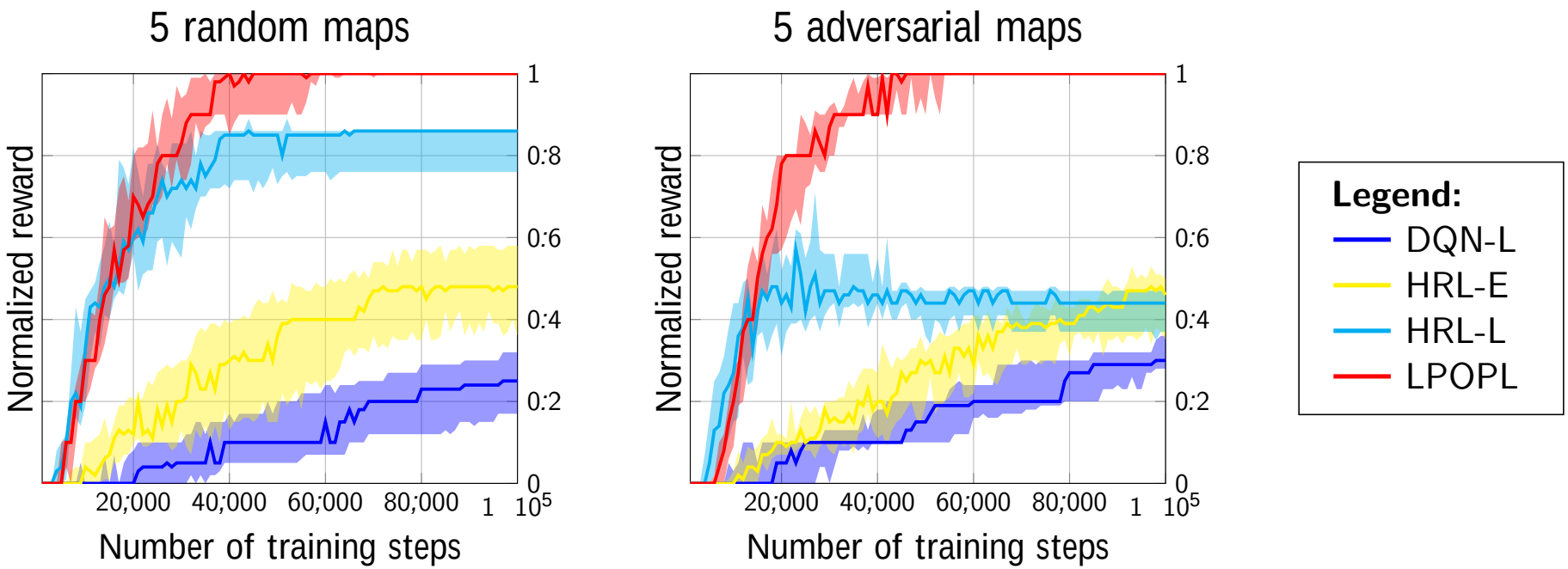
Description

10 tasks defined as sequence of subgoals (Andreas et al., 2017)
e.g. get iron, then get wood, then use factory.

Experiment 1: subgoal sequences

Description

10 tasks defined as sequence of subgoals (Andreas et al., 2017)
e.g. get iron, then get wood, then use factory.



Experiment 2: Interleaving subtasks

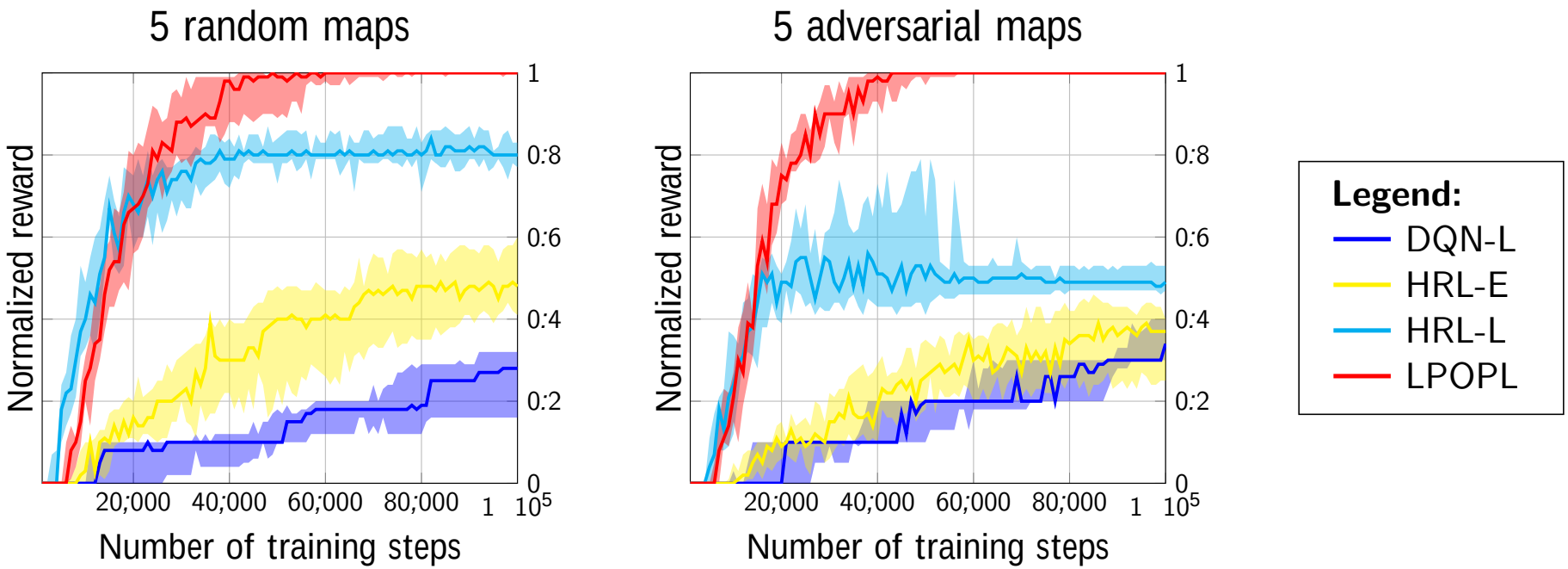
Description

Same set of 10 tasks but removing unnecessary order constraints.
e.g. (get iron and get wood), then use factory.

Experiment 2: Interleaving subtasks

Description

Same set of 10 tasks but removing unnecessary order constraints.
e.g. (get iron and get wood), then use factory.



Experiment 3: Safety constraints

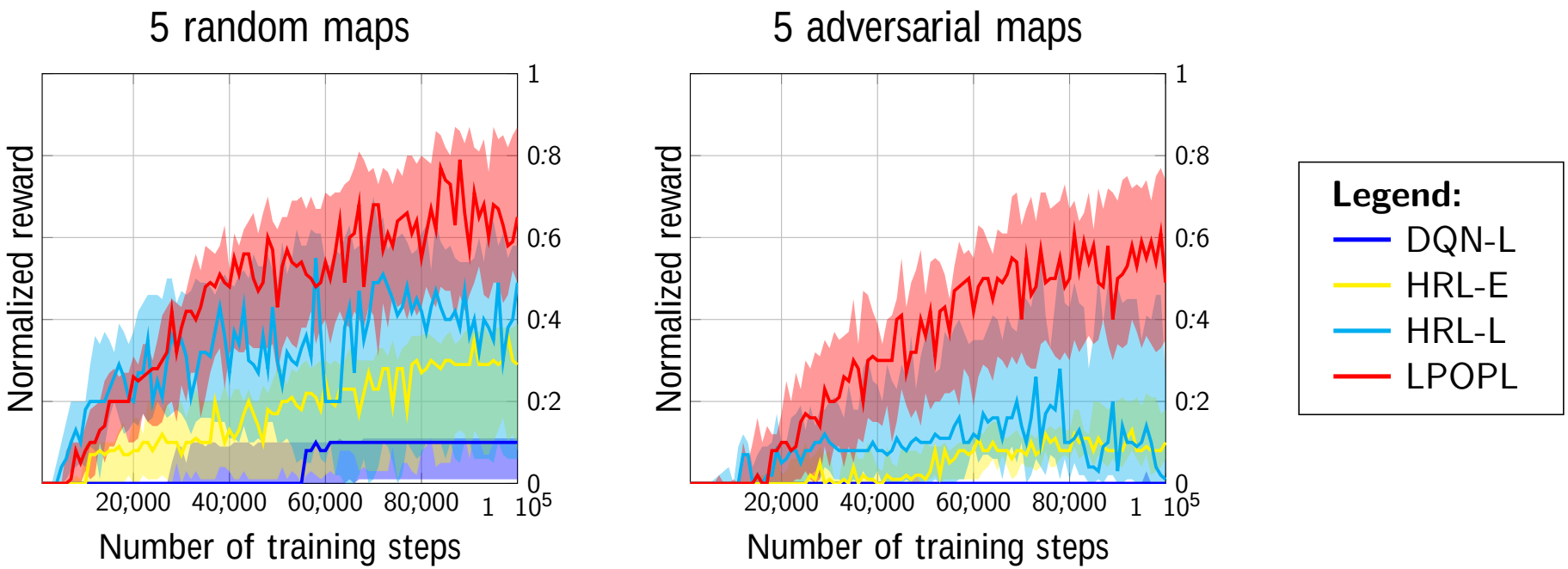
Description

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

Experiment 3: Safety constraints

Description

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



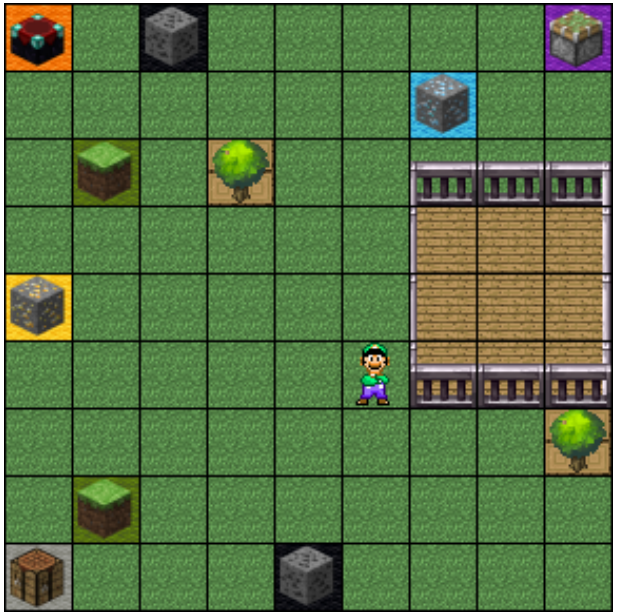
Outline

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



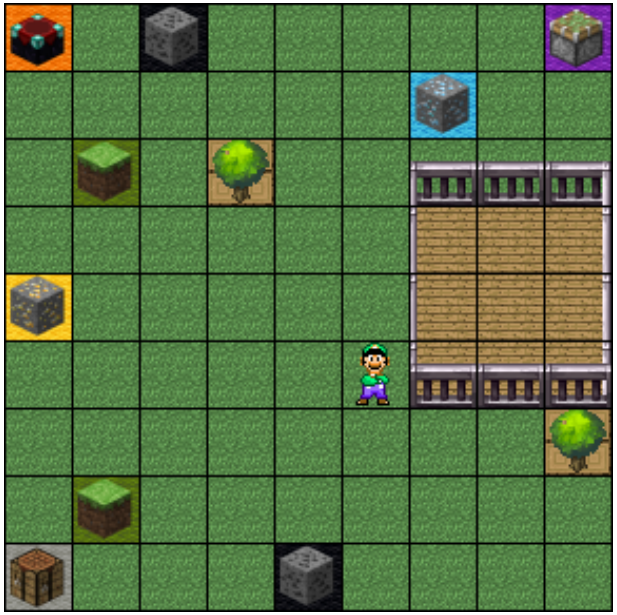
Problem

How to tell an RL agent what to do.

Our approach

Define tasks using LTL.
Decompose tasks using LTL progression.
Learn subtasks using off-policy RL.

Concluding remarks



Problem

How to tell an RL agent what to do.

Our approach

Define tasks using LTL.
Decompose tasks using LTL progression.
Learn subtasks using off-policy RL.

Experiments:

- LPOPL outperformed HRL and DQN over a variety of tasks.
- `https://bitbucket.org/RToroIcarte/lpopl`

Thanks!