Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Symbolic Reinforcement Learning using Inductive Logic Programming

Author:
Kiyohito Kunii

Supervisor:
Prof. Alessandra Russo
Mark Law
Ruben Vereecken

Submitted in partial fulfillment of the requirements for the MSc degree in MSc in Computing Science of Imperial College London

September 2018

Abstract

Reinforcement Learning (RL) is has been applied and proven to be successful in many domains. However, most of current RL models face limitations, namely, low learnig efficiency, uncable of using abstract reasoning, and inability of tranfer learning to a similar environment. In order to tackle these shortcoming, this paper introduces a new approach for RL called ILP(RL), which applies Answer Set Programming (ASP), a declarative logic programming suitable for complex search problems, and Inductive Learning of Answer Set Programs (ILASP), a learning framework of non-monotonic Inductive Logic Programming (ILP). ILP is another field of machine learning that is based on logic programming, and recent advance on ILP research have shown potential in many more applications. ILP(RL) learns a general concept of a valid move in an environment, called a hypothesis, using ILASP, and generate a sequence of actions to the destination using ASP. The learnt hypotheses is highly expressive and transferrable to a similar environment. While there are a number of past papers that attempt to incorporate symbolic representation to RL problems in order to achieve efficient learning, there has not been any attempt of applying symbolic learning into a RL problem. ILP(RL) was examined in a various simple maze games, and show that an agent learns faster than existing RL techniques. We also show that transfer learning successfully improve learning on a new but similar environment in a limited senarios. This proof of concept showpotentials for this new way of learning using ILP. Although the experiments were conducted in a simple environment, the results of the experiments show promissing, and there is an avenue for potential improvement.

Acknowledgments

I would like to thank Prof. Alessandra Russo for accepting to supervise my project, her enthusiasm for my work and invaluable guidance throughout.
I would also like to thank Mark Law for his expertise on inductive logic programming and answer set programming, and fruitful discussions, and Ruben Verrecken for his expertise on reinforcement learning and for providng me with advice and assistance for technical implementation.

Contents

List of Figures and Tables

Chapter 1

Introduction

1.1 Motivation

Reinforcement learning (RL) is a subfield of machine learning concerning where an agent learns how to behave in an environment by interacticing with the environment in order to maximise the total rewards it receives. The strength of RL is that it can be applied to many different domains where it is unknown to an agent how to perform a task. RL has been proven to work well in a number of complex environment, such as dynamic, continuous environments and even a real physical environment, given sufficient training and learning. Especially together with deep learning (DL), there have been many successful applications of RL in a number of domains, such as video games [**?**], the game of Go [**?**] and robotics [**?**].

Despite of these successful applications, however, there are still a number of issues for RL to overcome. First, most of RL algorithms requires large number of interactions and trials and errors, which is also computationally expense. Second, most of RL algorithms have no thought process to the decision making, and do not make use of high-level abstract reasoning, such as understanding symbolic representations or causal reasoning. Third, the transfer learning (TL), where the experience that an agent gained to perform one task can be useful in a different task and environment, is limited and the agent performs poorly even on a new but very similar environment or task.

In order to overcome these limitations, we introduce a new approach by applying Inductive Logic Programming (ILP) into RL senarios. ILP is another subfield of machine learning based on logic programming and derives a hypothesis in the form of logic program that, together with background knowledge, entails all of positive examples and none of the negative exmaples. ILP has several advantages compared to RL. First, unlike most of statistical machine learning methods, ILP requires a small number of training data due to the presense of language bias which defines what the program can learn. Second, the learnt program by ILP is expressed with a symbolic representation and therefore is easy to be interpreted by human users. Third, since the learnt hypothesis is a abstract and general notion, it can be easily applied to a different learning task, thus transfer learning is possible. ILP therefore enables the use of what is previously learnt to similar situations. The disadvantages of ILP sys-

tem are that the examples, or training data, have to be clearly defined and, unlike statistical machine learning, cannot be used with ambiguius noisy dataset. Another disadtantage is that ILP suffers from learning scalability. The search space for a hypothesis defined by the language bias increases with respect to the complexity for learning tasks and slows down learning process. Despite these shortocomings, there has been a number of progress in ILP, especially in ILP frameworks based on Answer Set Programming (ASP), a declarative logic programming which defines semantics based on Herbrand models (Gelfond and Lifschitz 1988). Due to the progress on both RL and ILP, we developed an new RL approach by incorporating a new ASP-based ILP framework called Learning from Answer Set (ILASP), which learns a valid move of an environment, and uses the learnt hypothesis and background knowledge to generate a sequence of actions in the environment.
In recent RL research, there is a number of research of introducing symbolic representation into RL in order to achieve more data-efficient learning as well as transparent learning. One of the methods is to incorporate symbolic representations into the system [**?**]. This approach is promising and shows a potential. However, none of the papers attempt to apply inductive learnig in RL senarios. but the combining of ILP and RL has not been explored Most of the ILP frameworks are also tested in cases where all the environment is known to the agent in advance.
Since the recent ILP framworks enables to learn a complex hypothesis in a more realistic environments, Finally, the recent advance of Inductive Logic Programming (ILP) research has enabled us to apply ILP in more complex situations and there are a number of new algorithms based on Answer Set Programmings (ASPs) that work well in non-monotonic scenarios. Because of the recent advancement of ILP and RL, it is natual to consider that a combination of both approaches would be the next field to explore. Therefore my motivation is to combine these two different subfields of machine learnig and devise a new way of RL algorithm in order to overcome some of the RL problems.
Particularly since [**?**], there have been several researches that further explored the incorporation of symbolic reasoning into RL, but the combining of ILP and RL has not been explored.

1.2 Objectives

The main objective of this project is to provide a proof-of-concept new RL framework using Inductive Logic Programming and to investigate the potentials of improving the learning efficiency and transfer learning capability that current RL algorithms suffer. This main goal is devided into the following high-level objectives:
We did various experiments in grid mazes to highlight property of the learning process and the learning performance is compared with existing reinforcement learning algorithms. We show that the learning convergence of ILP(RL) is faster than existing RL.
The objectives of this paper and important aspects of are summarized as follows:

1. **Translation of state transitions into ASP syntax.**

In RL, an agent interacts with an environment by taking an action and observes a state and rewards (MDP). Since ASP-based ILP algorithms require their inputs to be specified in ASP syntax, the conversion between MDP and ASP is required.

2. **Development of learning tasks.**
   ILP frameworks is based on a search space specified by language bias for a learning task, and is needed to be specified by the user. Various hyper-parameters for the learning task are also considered.

3. **Using the learnt concept to execute actions.**
   Having learnt a hypothesis using a ILP algorithm, the agent needs to choose an action based on the learnt hypothesis. We investigate how the agent can effective plan a sequence of actions using the hypothesis.

4. **Evaluation of the new framework in various environments.**
   In order to investigate the applicabilities of ILP-based approach, evalauation of the new framework on various senarios is conducted in order to gain insights of the potentials, especially how it improves the learning process and capability of transfer learning.

This paper is a proof of concept for the new way of reinforcement learning using ILP and therefore there is a limit to extend the current framework can be applied. More advanced environment such as continuous states or schocastic environment are not considered in this paper. Possibilities of applying these more complex environment are discussed in Section 6.2.

1.3 Contributions

The main contribution of this paper is the development of a novel ILP based approach to reinforcement learning and contributes to an incorporation of ASP-based ILP learning frameworks and Reinforcement Learning by applying a latest ILP framework called Learning from Answer Sets. To my knowledge, this is the first attemp that a ILP learning framework is incorporated into a reinforcement learning senario to facilitate learning process.
In simple environments, we show that the agent learns rule of the game and reaches an optimal policy faster than existing RL algorithms, learnt concepts is easy to understand for human users. We also show that the learnt hypothesis is a general concept and can be applied to other environment to mitigate learnig process.
The full hypotheses were learnt in the very early phase of learning and exploration phase. Thus with sufficient exploration, the model of the environment is correct and therefore it is able to find the optimal policy/path.
We show that ILP(RL) is able to solve a reduced MDP where the rewards are assumed to be associated with a sequence of actions planned as answer sets. Although this is a limitated solution, there is a potential to expand it to solve full MDP as discussed in Further Research.
TODO more details on the strength of the algorithm. Validity

1.4 Report Outline

Chapter 2. The necessary background of Answer Set Programming, Inductive Logic Programming and Reinforcement Learning for this paper are described.

Chapter 3. The descriptions of the new framework, called ILP(RL), is explained in details, and we highlight each aspect of learning steps with examples.

Chapter 4. The performance of ILP(RL) is measured in a simple game environment and compared against two existing RL algorithms. We measure learning efficiency and the capability of the transfer learning. We evaluate the outcomes and discuss some of the issues we currently face with the current framework.

Chapter 5. We review previous research on the related fields. Since there is no research that attemps to apply ILP to RL, we review applications of ASP in RL and the symbolic representations in RL for relevant research.

Chapter 6. We summarise the framework and experiments of ILP(RL) and discuss the potentials of further research.

Chapter 2

Background

This chapter introduces necessary background of Answer Set Programming (ASP), Inductive Logic Programming (ILP) and Reinforcement Learning (RL), which provide the foundations of our research.

2.1 Answer Set Programming (ASP)

Answer Set Programming (ASP) is a form of declarative programming aimed at knowledge-intensive applications such as difficult search problems [**?**]. We first introduce stable model on which ASP is based on, and describe ASP syntax.

2.1.1 Stable Model Semantics

The semantics of the logic is based on the notion of interpretation, which is defined under a *domain*. A domain contains all the objects that exist. In logic, it is convention to use a special interpretations called *Herbrand interpretations* rather than general interpretations.

Definition 2.1. *Herbrand Domain* (a.k.a *Herbrand Universe*) of a normal logic program *P*, denoted HD(P), is the set of all ground terms that are constants and function symbols appeared in *P*.

Definition 2.2. *Herbrand Base* of a normal logic program *P*, denoted HB(P) is the set of all ground predicates that are formed by predicate symbols in *P* and terms in HD_P.

Definition 2.3. *Herbrand Interpretation* of a set of a program *P*, denoted HI(P) is a subset of the HB_P, and is a set of ground atoms that are true in terms of interpretation.

Definition 2.4. *Herbrand Model* of a normal logic program *P* is a HI(P) if and only if a set *P* of clauses is satisfiable. In other words, the set of clauses *P* is unsatisfiable if no Herbrand model was found.

Definition 2.5. *Least Herbrand Model* (denoted as *M(P)*) is an unique minimal Herbrand model for definite logic programs. The Herbrand Model is a minimum Herbrand model if and only if none of its subsets is an Herbrand model.

For normal logic programs, there may not be any least Herbrand Model.

Example 2.1.1. (Herbrand Interpretation, Herbrand Model and M(P))

$$\mathrm{P} = \begin{cases} p(X) \leftarrow q(X). \\ q\ (a). \end{cases} \qquad \mathrm{HD(P)} = \{\ \mathrm{a}\ \}\ ,\ \mathrm{HB(P)} = \{\ \mathrm{q(a)},\ \mathrm{p(a)}\ \}$$

Given above, there are four Herbrand Interpretations HI(P) = $\langle$ {q(a)}, {p(a)}, {q(a), p(a)}, {} $\rangle$, and one Herbrand Model (as well as M(P)) = {q(a), p(a)}

Definite Logic Program is a set of definite rules, and a *definite rule* is of the form $h \leftarrow a_1, ..., a_n$. h and $a_1, ..., a_n$ are all atoms. h is the *head* of the rule and $a_1, ..., a_n$ are the *body* of the rule. *Normal Logic Program* is a set of normal rules, and a normal rule is of the form $h \leftarrow \mathrm{a}_1, ..., a_n$, *not* b_1, ..., *not* b_n where h is the head of the rule, and a_1, ..., a_n, b_1, ..., b_n are the body of the rule (both the head and body are all atoms).
To solve a normal logic program *Th*, the program P needs to be grounded. The *grounding* of *Th* is the set of all clauses that are c $\in$ *Th* and variables are replaced by terms in the Herbrand Domain.

Definition 2.6. The algorithm of grounding starts with an empty program Q = { } and the relevant grounding is constructed by adding to each rule R to Q such that

- R is a ground instance of a rule in P.
- Their positive body literals already occurs in the in the of rules in Q.

The algorithm terminates when no more rules can be added to Q.

Example 2.1.2. (Grounding)

$$\mathrm{P} = \begin{cases} q(X) \leftarrow p(X). \\ p(a). \end{cases}$$

ground(P) in this example is {p(a), q(a)}.

Not only the entire program needs to be grounded in order for an ASP solver to work, but also each rule must be *safe*. A rule *R* is safe if every variable that occurs in the head of the rule occurs at least once in $\mathrm{body}^+(\mathrm{R})$. Since there is no unique least Herbrand Model for a normal logic program, Stable Model of a normal logic program was defined in [**?**]. In order to obtain the Stable Model of a program P, P needs to be converted using *Reduct* with respect to an interpretation X.

Definition 2.7. The *reduct* of P with respect to X can be constructed such that

- If the body of any rule in P contains an atom which is not in X, those rules need to be removed.

- All default negation atoms in the remaining rules in P need to be removed.

Example 2.1.3. (Reduct)

$$P = \begin{cases} p(X) \leftarrow not\ q(X). \\ q(X) \leftarrow not\ p(X). \end{cases} \quad X = \{p(a), q(b)\}$$

Where X is a set of atoms. ground(P) is
p(a) ← not q(a).
p(b) ← not q(b).
q(a) ← not p(a).
q(b) ← not p(b).

The first step removes p(b) ← not q(b). and q(a) ← not p(a).
p(a) ← not q(a).
q(b) ← not p(b).

The second step removes negation atoms from the body.
Thus reduct P^x is $(ground(P))^x$ = {p(a), q(b).}

Definition 2.8. A *Stable Model* of P, denoted SM(P)??, is an interpretaiton X if and only if X is the unique least Herbrand Model of $ground(P)^x$ in the logic program.

2.1.2 Answer Set Programming (ASP) Syntax

Definition 2.9. Answer set of normal logic program P is a Stable Model defined by a set of rules, where each rule consists of literals, which is made up with an atom *p* or its default negation *not p* (*negation as failure*). Answer Set Programming (ASP) is a normal logic program with extensions: constraints, choice rules and optimisation statements.

A *defnite rule* is:

$$\underbrace{h}_{\text{head}} \leftarrow \underbrace{a_1, a_2, ..., a_n}_{\text{body}} \tag{2.1}$$

A *normal rule* is:

$$\underbrace{h}_{\text{head}} \leftarrow \underbrace{a_1, a_2, ..., a_n, not\ b_{n+1}, ...not\ b_{n+k}}_{\text{body}} \tag{2.2}$$

where h, a_1, ... b_{n+k} are all atoms. A rule with empty body is a *fact*, and a rule with empty head is a *constraint*. The constraint filters any irrelevant answer sets. When computing $ground(P)_x$, the empty head becomes ⊥, which cannot be in the answer sets.
A *choice rule* can express possible outcomes given an action choice, which is of the form:

$$\underbrace{l\{h_1, ..., h_m\}u}_{\text{head}} \leftarrow \underbrace{a_1, ..., a_n, not\ b_1, ..., not\ b_n}_{\text{body}} \tag{2.3}$$

where l and u are integers and h_i for $1 \leq i \leq m$ are atoms. The head is called *aggregates*.
Optimisation statement is useful to sort the answer sets in terms of preference, which is of the form:

$$\begin{aligned}&\#minimize[a_1{=}w_1,...a_n{=}w_n]\ or\\&\#maximize[a_1{=}w_1,...a_n{=}w_n]\end{aligned} \tag{2.4}$$

where $w_1,..., w_n$ is integer weights and $a_1,...,a_n$ is ground atoms. ASP solvers compute the scores of the weighted sum of the sets of ground atoms based on the true answer sets, and find optimal answer sets which either maximise or minimise the score.
Clingo is one of the modern ASP solvers that executes the ASP program and returns answer sets of the program [**?**], and we will use Clingo 5 for the implementation of our new framework.

2.2 Inductive Logic Programming (ILP)

Inductive Logic Programming (ILP) is a subfield of machine learning research area aimed at supervised inductive concept learning, and is the intersection between machine learning and logic programming [**?**]. The purpose of ILP is to inductively derive a hypothesis H that is a solution of a learning task, which coveres all positive examples and none of negative examples, given a hypothesis language for search space and cover relation [**?**]. ILP is based on learning from entailment, as shown in Equation 2.2.

$$B \wedge H \models E \tag{2.5}$$

where E contains all of the positive examples (E^+) and none of the negative examples (E^-).
An advantage of ILP over statistical machine learning is that the hypothesis that an agent learnt can be easily understood by a human, as it is expressed in first-order logic, making the learning process more transparent. By contrast, a limitation of ILP is learning scalability. There are usually thousands or more examples in many real-world examples. Scaling ILP task to cope with large examples is a challenging task [**?**].

2.2.1 ILP under Answer Set Semantics

There are several ILP non-monotonic learning frameworks under the answer set semantics . We first introduce two of them: *Cautious Induction* and *Brave Induction* ([**?**]), which are foundations of *Learning from Answer Sets* discussed in Section 2.2.2, a state-of-art ILP framework that we will use for our new framework. (for other non-monotonic ILP frameworks, see [**?**], [**?**], [**?**] and [**?**]).

Cautious Induction

Definition 2.10. *Cautious Induction task* [1] is of the form $\langle$ B, E^+, E^- $\rangle$, where B is the background knowledge, E^+ is a set of positive examples and E^- is a set of negative examples.
$H \in ILP_{cautious} \langle B, E^+, E^- \rangle$ if and only if there is at least one answer set A of $B \cup H$ ($B \cup H$ is satisfiable) such that for every answer set A of $B \cup H$:

- $\forall e \in E^+ : e \in A$
- $\forall e \in E^- : e \notin A$

Example 2.2.1. (Cautious Induction)

$$B = \begin{cases} \textit{exercises} \leftarrow \textit{not} \;\; \textit{eat\_out}. \\ \textit{eat\_out} \leftarrow \textit{exercises}. \end{cases} \quad E^+ = \{\text{tennis}\},\, E^- = \{\text{eat\_out}\}$$

One possible $H \in ILP_{cautious}$ is {tennis← exercises, ← not tennis }.

The limitation of Cautious Induction is that positive examples must be true for all answer sets and negative examples must not be included in any of the answer sets. These conditions may be too strict in some cases, and Cautious Induction is not able to accept a case where positive examples are true in some of the answer sets but not all answer sets of the program.

Example 2.2.2. (Limitation of Cautious Induction)

$$B = \begin{cases} \textit{1\{situation(P, awake), situation(P, sleep)\}1} \leftarrow \textit{person(P)}. \\ \textit{person(john)}. \end{cases}$$

Neither of *situation(john, awake)* nor *situation(john, sleep)* is false in all answer sets. In this example, it only returns person(john). Thus no examples could be given to learn the choice rule.

Brave Induction

Definition 2.11. *Brave Induction task* is of the form $\langle$ B, E^+, E^- $\rangle$ where, B is the background knowledge, E^+ is a set of positive examples and E^- is a set of negative examples. $H \in ILP_{brave} \langle B, E^+, E^- \rangle$ if and only if there is at least one answer set A of $B \cup H$ such that:

- $\forall e \in E^+ : e \in A$

[1] This is more general definition of Cautious Induction than the one defined in [**?**], as the concept of negative examples was not included in the original definition.

- $\forall$ e $\in$ E$^-$: e $\notin$ A

Example 2.2.3. (Brave Induction)

$$B = \begin{cases} \textit{exercises} \leftarrow \textit{not eat\_out.} \\ \textit{tennis} \leftarrow \textit{holiday} \end{cases} \qquad E^+ = \{\text{tennis}\}, E^- = \{\text{eat\_out}\}$$

One possible H $\in$ ILP_{brave} is {tennis}, which returns {tennis, holidy, exercises} as answer sets.

The limitation of Brave Induction that it cannot learn constraints, since the above conditions for the examples only apply to at least one answer set A, whereas constrains rules out all answer sets that meet the conditions of the Brave Induction.

Example 2.2.4. (Limitation of Brave Induction)

$$B = \begin{cases} \textit{1\{situation(P, awake), situation(P, sleep)\}1} \leftarrow \textit{person(P).} \\ \textit{person(C)} \leftarrow \textit{super\_person(C).} \\ \textit{super\_person(john).} \end{cases}$$

In order to learn the constraint hypothesis H = { $\leftarrow$ not situation(P, awake), super_person(P)}, it is not possible to find an optimal solution.

2.2.2 Inductive Learning of Answer Set Programs (ILASP)

Learning from Answer Sets (LAS)

Learning from Answer Sets (LAS) was developed in [**?**] to faciliate more complex learning tasks that neither Cautious Induction nor Brave Induction could learn. Examples used in LAS area tuple of pair of atoms called *Partial Interpretations*, which are of the form:

$$E = \langle e^{\text{inc}}, e^{\text{exc}} \rangle. \tag{2.6}$$

(called *inclusions* and *exclusions* of *E* respectively). A Herbrand Interpretation extends a partial interpretation if it includes all of E^{inc} and none of E^{exc}.
A *Learning from Answer Sets* task is of the form:

$$T = \langle B, S_{\text{M}}, E^+, E^- \rangle \tag{2.7}$$

where B is background knowledge, S_M is hypothesis space, and E^+ and E^- are examples of positive and negative partial interpretations. S_M consists of a set of normal rules, choice rules and constraints. S_M is specified by *language bias* of the learning task using *mode declaration*. Mode declaration specifies what can occur in a hypothesis by specifying the predicates, and consists of two parts: *modeh* and *modeb*. *modeh* and *modeb* are the predicates that can occur in the head of the rule and body of the rule respectively. Language bias is the specification of the language in the hypothesis in order to reduce the search space for the hypothesis.

Definition 2.12. Learning from Answer Sets (LAS)
Given a learning task T, the set of all possible inductive solutions of T, denoted $ILP_{LAS}(T)$, and a hypothesis H is an inductive solution of $ILP_{LAS}(T)$ $\langle$ B, S_M, E^+, E^- $\rangle$ such that:

1. $H \subseteq S_M$
2. $\forall\ e^+ \in E^+ : \exists\ A \in$ Answer Sets($B \cup H$) such that A extends e^+
3. $\forall\ e^- \in E^- : \nexists\ A \in$ Answer Sets($B \cup H$) such that A extends e^-

Inductive Learning of Answer Set Programs (ILASP)

Inductive Learning of Answer Set Programs (ILASP) is an algorithm that is capable of solving LAS tasks, and is based on two fundamental concepts: *positive solutions* and *violating solutions*.
A hypothesis H is a positive solution if and only if

1. $H \subseteq S_M$
2. $\forall\ e^+ \in \exists\ A \in$ Answer Sets($B \cup H$) such that A extends e^+

A hypothesis H is a violating solution if and only if

1. $H \subseteq S_M$
2. $\forall\ e^+ \in E^+\ \exists\ A \in$ Answer Sets($B \cup H$) such that A extends e^+
3. $\exists\ e^- \in E^-\ \exists\ A \in$ Answer Sets($B \cup H$) such that A extends e^-

Given both definitions of positive and violating solutions, ILP_{LAS} $\langle$ B, S_M, E^+, E^- $\rangle$is positive solutions that are not violating solutions.

Example 2.2.5. (ILASP).
XXX

A Context-dependent Learning from Answer Sets

Context-dependent learning from answer sets ($ILP_{LAS}^{context}$) is a further generalisation of ILP_{LAS} with *context-dependent examples*[**?**][2]. *Context-dependent examples* are examples that each unique background knowledge, or *context*, only applies to specific examples. This way the background knowledge is more structured rather than one fixed background knowledge that are applied to all examples.
Formally, partial interpretation is called *context-dependent partial interpretation (CDPI)*, which is of the form:

$$\langle e, C \rangle \tag{2.8}$$

[2]The original paper is developed on top of *learning from ordered answer sets ($ILP_{LOAS}^{context}$)*. In this paper, we do not use ordered answer sets and therefore simplified to $ILP_{LAS}^{context}$.

where *e* is a partial interpretation and C is a context, or an ASP program without weak constraints.

Definition 2.13. $ILP_{LAS}^{context}$ task is of the form T = ⟨ B, S_M, E^+, E^-⟩ B. A hypothesis H is an inductive solution of T if and only if

1. H ⊆ S_M in $ILP_{LAS}^{context}$
2. ∀⟨ e, C⟩ ∈ E^+, ∃A ∈ Answer Sets (B ∪ C ∪ H) such that A extends e
3. ∀⟨ e, C⟩ ∈ E^-, ∄A ∈ Answer Sets (B ∪ C ∪ H) such that A extends e

One of the main advantages of adding contex dependent examples is that when a hypothesis is computed, only the relevant set of examples are used for search rather than using all examples at once. Relevant examples are counterexamples for the previously computed hypothesis in the previous iterations. This iterative approach significantly increases the efficiency of solving learning tasks, and enables more expressive structure of the background knowlege to particular examples.
$ILP_{LAS}^{context}$ is used in ILASP2i, which we use for our new framework.

Example 2.2.6. ($ILP_{LAS}^{context}$).
XXX

2.3 Reinforcement Learning (RL)

Reinforcement learning (RL) is a subfield of machine learning regarding how an agent behaves in an environment in order to maximise its total reward. As shown in Figure 2.1, the agent interacts with an environment, and at each time step the agent takes an action and receives observation, which affects the environment state and the reward (or penalty) it receives as the action outcome. In this section, we briefly introduce the background in RL necessary for our new framework as well as benchmark of experiments discussed in Chapter 4.

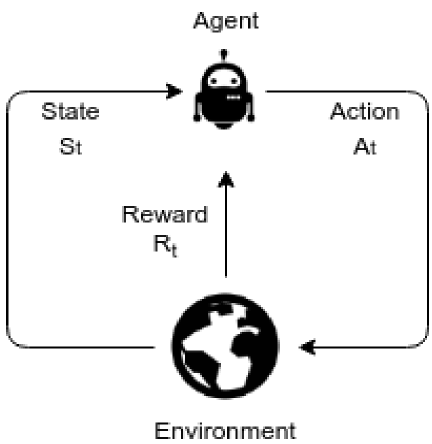

Figure 2.1: The interaction between an agent and an environment

2.3.1 Markov Decision Process (MDP)

An agent interacts with an environment at a sequence of discrete time step, which is part of the sequential history of observations, actions and rewards. The sequential history is formalised as

$$H_{\mathrm{t}} = S_1, R_1, A_1, ..., A_{\mathrm{t\text{-}1}}, S_{\mathrm{t}}, R_{\mathrm{t}}. \tag{2.9}$$

where S is *states*, R is *rewards* and A is *actions*. A state S_t determines the next environment and has a *Markov property* if and only if

$$P[S_{\mathrm{t+1}}|S_{\mathrm{t}}] = P[S_{\mathrm{t+1}}|S_1, ..., S_{\mathrm{t}}]. \tag{2.10}$$

In other words, the probability of reaching S_{t+1} depends only on S_t, which captures all the relevant information from the earilier history ([**?**]). When an agent must make a sequence of decision, the sequential decision problem can be formalised using *Markov decision process (MDP)*. MDP formally represents a fully observable environment of an agent for RL.

Definition 2.14. Markov Decision Process (MDP)
Markov decision process (MDP) is defined in the form of a tuple $\langle$S, A, T, R$\rangle$ where:

- S is the set of finite states that is observable in the environment.
- A is the set of finite actions taken by the agent.
- T is *a state transition function*: S $\times$ A $\times$ S $\rightarrow$ [0,1], which is a probability of reaching a future state $s' \in$ S by taking an action a $\in$ A in the current state s $\in$ S.
- R is a reward function $R_a(s, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$, the expected immediate reward that action *a* in state *s* at time *t* will return.

The objective of an agent is to solve an MDP by taking a sequence of actions to maximise the total cumulative reward it receives. An method to find the optimal solution of an MDP is Reinforcement Learning (RL). Formally, the objective of a RL agent is to maximise *expected discounted return*,which is defined as:

$$G_{\mathrm{t}} = R_{\mathrm{t+1}} + \gamma R_{\mathrm{t+2}} + ... = \sum_{k=0}^{\infty} \gamma^{\mathrm{k}} R_{\mathrm{t+k+1}} \tag{2.11}$$

where γ is a discount rate $\gamma \in$ [0,1], a parameter, which represents the preference of the agent for the present reward over future rewards. If γ is low, the agent is more interested in maximising immediate rewards.
TODO explain what k is
For RL programs, it is not necessary for the agent to know T and R in advance, but they are present in the environment and can be realised each time the agent takes an action.

2.3.2 Policies and Value Functions

Most of RL algorithms are concerned with estimating *Value functions*. Value functions estimate the expected return, or expected future reward, for a given action in a given state. The expected reward for an agent is dependent on the agent's action. Value functions are defined by *policies*, which maps from staets to probabilities of choosing each action. The state value function $v_\pi(s)$ of an MDP under a policy π is the expected return starting from state *s*, which is of the form:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] \text{ for all } s \in \mathcal{S} \tag{2.12}$$

The optimal state-value function $v^*(s)$ maximises the value function over all policies in the MDP, which is of the form:

$$v^*(s) = \max_\pi v_\pi(s) \tag{2.13}$$

Example 2.3.1. (Value Functions).
XXX

Similar to the state value function v_π, we define a *action-value function* under a policy π, which represents the value of taking action a in state s following a policy π.

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] \text{ for all } s \in \mathcal{S} \tag{2.14}$$

The optimal state-action-value function $q^*(s,a)$ maximises the action-value function over all policies in the MDP, which is of the form:

$$q^*(s,a) = \max_\pi q_\pi(s,a) \tag{2.15}$$

A solution to the sequential decision problem is called a *policy* π, a sequence of actions that leads to a solution. An optimal policy achieves the optimal value function (or action-value function), and it can be computed by maximising over the optimal value function (or action-value function).

Example 2.3.2. (Action-value Functions).
XXX

2.3.3 Model-based and Model-free Reinforcement Learning

A *model* is a representation of an environment that an agent can use it to understand how the environment should look like. There are two different types of RL methods: *model-based* and *model-free* RL method. Model-based learning is that, given the model of the environment is known, the agent use the model to plan for a series of actions to achieve the agent's goal. The model itself can be learnt by interacting with the environment by taking actions, which returns states and rewards, and the parameters of the action models can be estimated by maximum likelihood methods [**?**]. Using a model, an agent can do planning to generate or improve a policy for the modeled environment. Most of the RL methods are model-free learning, where

the model is unknown and the agent learns to achieve the goal by solely iteracting with the environment. Thus the agent knows only possible states and actions, and the transition state and reward probability functions are unknown. When the model of the environmnet is correct, unlike model-free learning, the agent does not need trial and erros interactions with the environment and therefore more efficient to reach an optimal policy. However, when the model is not a representation of the environment, or the true MDP, the planning algorithms will not lead to the optimal policy, but a suboptimal policy.
One algorithm which combine both aspects of model-based and model-free learning to solve the issue of sub-optimality is called Dyna [**?**]. Dyna learns a model from real experience and use the model to generate simulated experience to update the evaluation functions. This approach is more efficient because the simulated experience is relatively easy to generate compared gaining real experience, thus less iterations are required.
This idea of learning a model of the environment and use it to execute planning is a core idea of our new framework.

2.3.4 Temporal-Difference (TD) Learning

One of the RL approaches for solving a MDP is called *Temporal-Difference (TD) Learning*. TD learning is an online model-free learning and learns directly from episodes of imcomplete experiences without a model of the environment. TD updates the estimates of value function as follow:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.16}$$

where $R_{t+1} + \gamma V(S_{t+1})$ is the target for TD update, which is biased estimated of $v_\pi (S_t)$, and $\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called TD error, which is the error in $V(S_t)$ available at time t+1. Since TD methods only needs to know the estimate of one step ahead and does not need the final outcome of the episodes (also called TD(0)), it can learn online after every time step. TD also works without the terminal state, which is the goal for an agent. TD(0) is proved to converge to v_π in the table-based case (non-function approximation).
Q-learning is off-policy TD learning defined in [**?**], where the agent only knows about the possible states and actionns. The transition states and reward probability functions are unknown to the agent. It is of the form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma max(a+t)Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.17}$$

where α is a constant step-size parameter, or learning rate, α between 0 and 1, and γ is a discount rate. The equation is used to update the state-action value function called *Q function*. The function Q(S,A) predicts the best action A in state S to maximise the total cumulative rewards.

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|s_t, a_t] \tag{2.18}$$

Q-learning is guaranteed to converge to a optimal policy in a fininte tabular representation. Since Q-learning is one of the most widely used RL methods and our experiments are conducted in a tabular representation, we use it as one of the benchmarks for our new framework.

2.3.5 Function Approximation

Q-learning with tarbular method works when every state-action value can be represented. In case of very large MDPs, however, it may not be possible to represent all state-action values with a tarbular representation. For example, robot arms has a continuous states in 3D dimentional space. This problem motivate the use of function approximation, which estimates value function with function approximation. Not only is it represented in tabular form, but also in the form of a parameterized function with *weight vector* w. Unlike Q-table, changing one weight updates the estimated value of not only one state, but many states, and this generalisation makes it more flexible to apply different scenarios that tabular approach could not be applied. The reason we are introduing the function approximation is that this will be used as a benchmark of our new framework.

The Prediction Objective ($\overline{VE}$)

With function approximation, an update at one state changes many other states, and therefore the values of all states will not be exactly accurate, and there is a tradeoff among states as to which state we make it more accurate, while other might be less accurate.
The error in a state s is the squeare of the difference between the approximate value $\hat{v}$(s,w) and the true value v_π(s). The objective function can be defined by weighting it over the state space by μ, the *Mean Squared Value Error*, denoted $\overline{VE}$.

$$\overline{VE}(w) \doteq \sum_{s \in S} \mu(s) \big[v_\pi(s) - \hat{v}(s,w)\big]^2. \tag{2.19}$$

Stochastic gradient descent (SGD)

Stochastic gradient descent (SGD) methods are commonly used to learn function approximation in value prediction, which works well for online reinforcement learning. The weight vector w in SGD is defined as:

$$w \doteq \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_\mathrm{n} \end{pmatrix} \tag{2.20}$$

and $\hat{v}$(s,w) is a differentiable function of w for all s $\in$ S. SGD adjusts the weights vector by a fraction of α, a step-size parameter, in the direction what will reduce the error on that example the most. Formally, it is defined as:

$$\begin{aligned} w_{\mathrm{t+1}} &\doteq w_{\mathrm{t}} - \frac{1}{2}\alpha \bigtriangledown \left[v_{\pi}(S_{\mathrm{t}}) - \hat{v}(S_{\mathrm{t}}, w_{\mathrm{t}})\right]^2. \\ &= w_{\mathrm{t}} + \alpha\left[v_{\pi}(S_{\mathrm{t}}) - \hat{v}(S_{\mathrm{t}}, w_{\mathrm{t}})\right] \bigtriangledown \hat{v}(S_{\mathrm{t}}, w_{\mathrm{t}}). \end{aligned} \tag{2.21}$$

The dradient of J(w), which is the vector of partial derivatives with respect to each weight vector, is defined as:

$$\bigtriangledown_{\mathrm{w}} J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_{\mathrm{n}}} \end{pmatrix} \tag{2.22}$$

Linear Value Function Approximation

A *linear function* of a weight vector is a type of simple function approximation to approximate action-value function.

$$\hat{q}(s, a) \approx q_{\pi}(s, a) \tag{2.23}$$

The vector x(s,a) represents a *feature vector*, which is of the form:

$$x(S, A) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_{\mathrm{n}}(S, A) \end{pmatrix} \tag{2.24}$$

where each x(s,a) is a feature of corresponding action-state pair.
Using SGD update with linear function approximation, the gradient of the approximate value function with respect to w is:

$$\bigtriangledown \hat{q}(s, a, w) = x(s, a) \tag{2.25}$$

Thus the general SGD update defined in (2.21) can be simplified to the following:

$$w_{\mathrm{t+1}} = w_{\mathrm{t}} + \alpha\left[q_{\pi}(S_{\mathrm{t}}, A_{\mathrm{t}}) - \hat{q}(S_{\mathrm{t}}, A_{\mathrm{t}}, w_{\mathrm{t}})\right] x(S_{\mathrm{t}}, A_{\mathrm{t}}) \tag{2.26}$$

Unlike non-linear value function approximation, linear method is guaranteed to converge to a global optimum. One disadvantage of the linear method is that it cannot express any relationship between features. For example, it cannot represent that feature *i* is useful only if feature *j* is not present. Nevertheless, this approach is sufficient enough for our experiments, which will be described in Chapter 4.

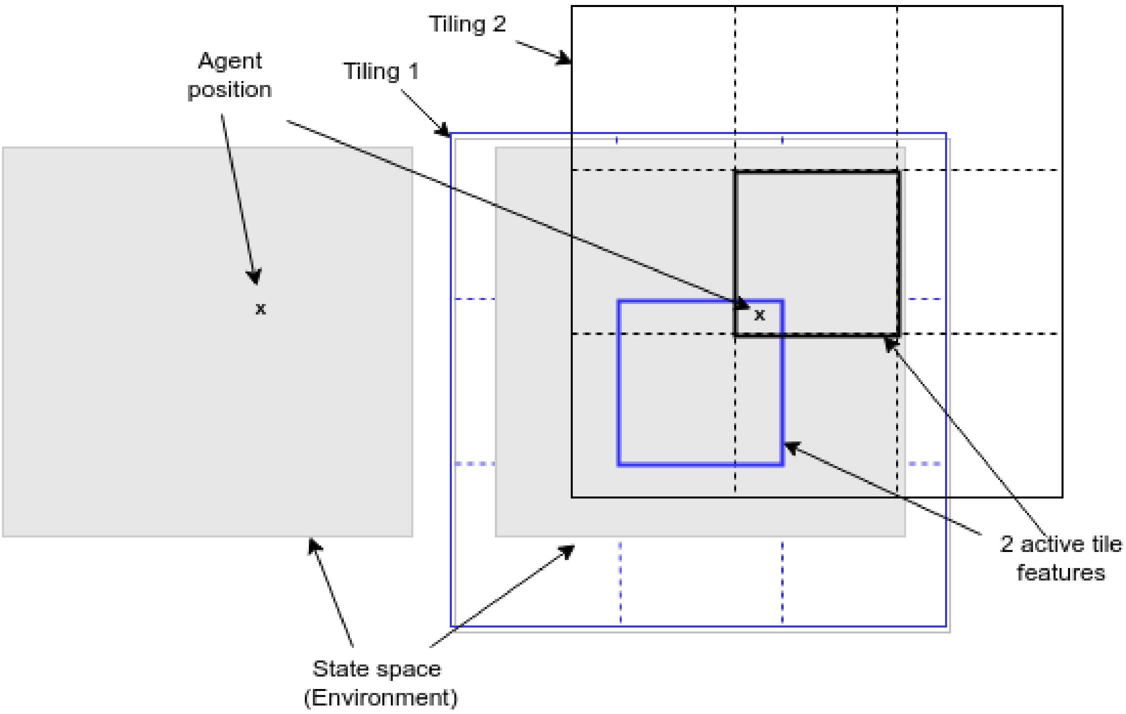

Figure 2.2: Tiling coding illustration

Tile Coding (TBD)

There are different linear function approximation methods to represents states as features. Feature construction depends on a problem you are solving. We introduce *Tile Coding* which will be used for our benchmark.
Illustration of tile coding is shown in Figure 2.2. In Tile Coding, state set is represented as a continuous two-dimentionala space. If a state is within the space, then the value of the corrensponding feature is set to be 1 to indicate that the feature is present, while 0 indicates that the feature is absent. This way of representing the feature is called *binary feature*. *Coarse coding* represents a state with which binary features are present within the space. One area is associated with one weight w, and training at a state will affect the weight of all the areas overlapping that state. the approximate value function will be updated within at all states within the union of the areas, and a point that has more overlap will be more affected. The size and shape of the areas will determine the degree of the generalisation. Large areas will have more generalisation. In addition, tiles can be overlap, and the change of the weight in that state will affect all other states within the intersection of the spaces. The degree of overlap within a space will determined the degree of the generalisation. The shape of the space also affect how it is generalised.
Tile coding is a type of coarse coding. *Tiling* is a partition of state space, and each element of the partition is called a *tile*.
The state stpace is partitioned into multiple tiles with multiple tilings. Each tile in each tiling is associated with
In order to do coarse coding with tile coding, multiple tilings are required, each tiling is offset from one another by a fraction of a tile width.

As illustrated in Figure XXX, when a state occurs, several features with corresponding tiles become active,
Tile coding has computational advantage, since each component of tiling is binary value, XXXX.
a trained state will be generalised to other states if they are within any of the same tiles.
Similar to coarse coding, the size and shape of tiles will determine the degree of approximation.

Example 2.3.3. (Tile Coding).
XXX

2.3.6 Transfer Learning (TBD)

Transfer learning is a method that knowledge learnt in one or more tasks can be used to learn a new task better than if without the knowlege in the first task.
Transfer learning is an active research areas in machine learning, but not many have been done in RL. Since training tend to be time consuming and computational expensive, transfer learning allow the trained model to be applied in a different setting.
Transfer learning in RL is particularly important since most of the RL research have been done in a simulation or game scenarios, and training RL models in a real physical environment is more expensive to conduct.
Even in a virtual environments like games, the transfer learning between different tasks will greatly will have a big impact on potential applications.
This will also speed up learning
Transfer learning in ILP domain have been proved to be successful in many fields,
Since this project is combining ILP into RL senarios, this has a potential for extending this particular research.
We conducted experiements on transfer learning capabilities, which we describe in XXX.
One of the purposes of transfer learning is so that the agent requires less time to learn a new task with the help of what was learn in previous tasks.
Another goal would be to measure how effectively the agent reuses its knoledge in a new task. In this case the performan of learningon the first task is usally not measured.
There are many different matrices used to measure the performance of the transfer learning. Five common matrics are defined in XX as follow.
TODO source task selection
Since each matric measures different aspect of transfer learning, using multiple metrics would provide more comprehensive views of the performance of an RL algorithm.

Chapter 3

Framework

3.1 Overview

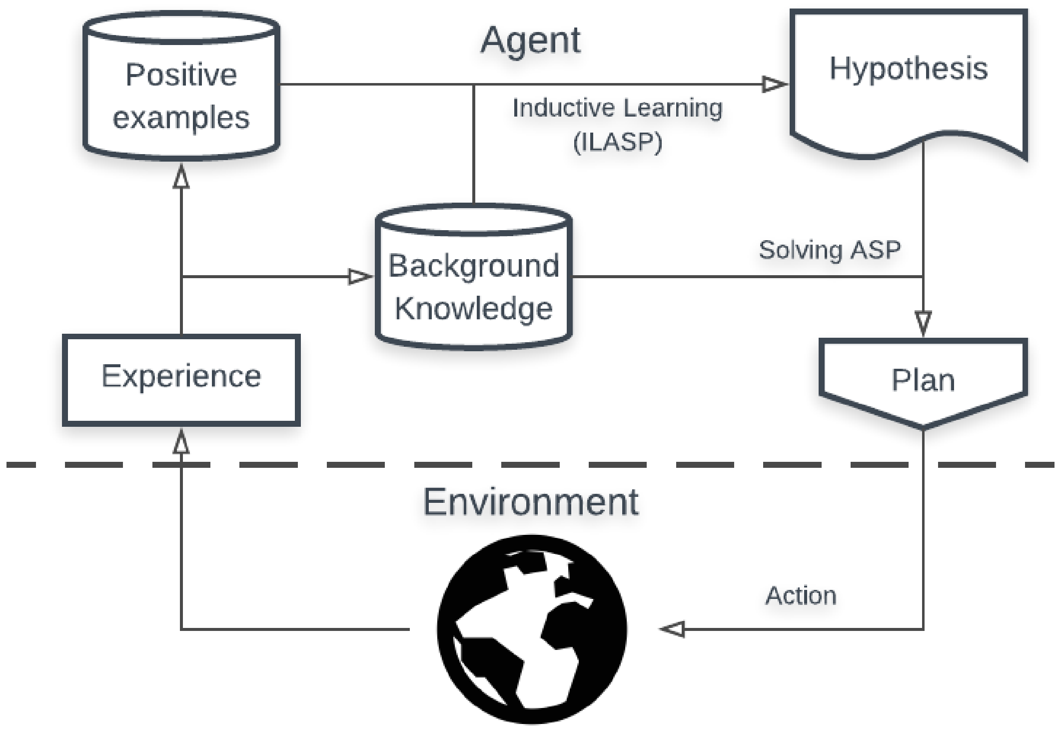

Figure 3.1: ILP(RL) overview

The overall architecture of the learning framework, called *ILP(RL)*, is shown in Figure 3.1. By interacting with the environment, an agent receives state transition experiences as positive examples, which is used by ILASP, together with pre-defined background knowledge and language bias, to learn and improve hypotheses. The agent also remembers surrounding information it has seen as background knowledge, which is used to make an action plan together using the learnt hypotheses by solving an answer set program. Mechanisms of each step is explained in details in the following sections.

3.2 Inductive Learning

The first step is to translate experiences generated by the interaction with an environment into ASP syntax. Similar to an existing RL, an agent explores an environment following an exploration strategy. Every time the agent takes an action, these experiences are recorded in two different forms: a positive example and background knowledge. Positive examples and background knowledge are used by ILASP for inductive learning, and background knowledge is used by both ILASP and ASP for solving for answer sets. Thus it is necessary to convert them into ASP syntax.

3.2.1 Positive examples

Positive examples contains information about how the state transitions between the current state and the next state. Positive examples are equivalent to context-dependent partial interpretation (CDPI) in $ILP_{LAS}^{context}$. As defined in the equation 2.8, CDPI is of the form $\langle$ e, C $\rangle$ where e = $\langle$ e^{inc}, e^{exc} $\rangle$. Each of the components in CDPI is defined as follows:

Definition 3.1. e^{inc} of CDPI for ILP(RL) is the next state of an agent $\forall\ s_{t+1} \in S$ such that answer sets of $B \cup H$ does not cover.

Definition 3.2. e^{exc} of CDPI for ILP(RL) is the next state of an agent $\forall\ s_{t+1} \notin S$ such that answer sets of $B \cup H$ covers, as well as $\forall\ s'_{t+1} \in S_{neighbor}$ such that $s_{t+1} \neq s'_{t+1}$.

where B is the current background knowledge, H is the current hypotheses learnt by previous inductive learning using ILASP, S is all the states in the environment, and $S_{neighbor}$ is all adjacent states of s_t as well as s_t itself.

Definition 3.3. Contex C of CDPI for ILP(RL) contains an action a_t, a state s_t, and adjacent walls of s_t.

In this paper, we assume that part of context contains only whether a wall exists or not, with the presence of a wall, the agent cannot move to the state where a wall exist.

Positive examples in ASP syntax

A positive example is expressed as a following ASP form:

$$\#\mathsf{pos}(\{e^{\text{inc}}\}, \{e^{\text{exc}}\}, \{\mathsf{C}\}) \tag{3.1}$$

For e^{inc} and e^{exc}, s_{t+1} is expressed as *state_after((x,y))*, where x and y represent coodinates of X and Y axis in an environment respectively. Thus both e^{inc} and e^{exc} contain only state_after((x,y)) or empty. For context C, a_t is translated int *action((a))*, s_t is translated into *state_before((x,y))*, and adjacent walls of s_t are translated into *wall((x′,y′))*.

Example 3.2.1. (Positive examples).

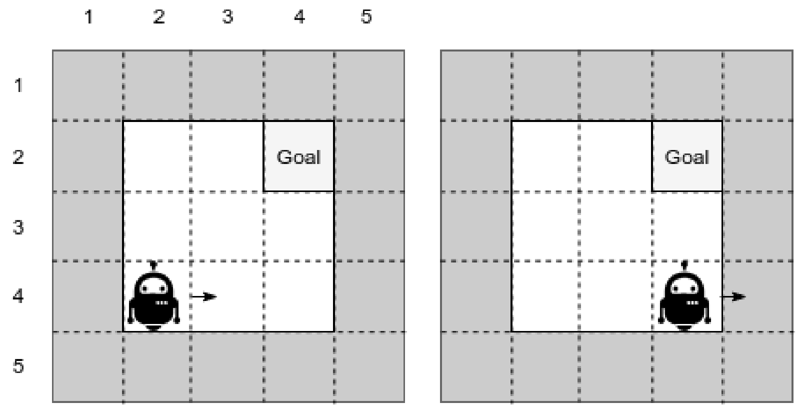

Figure 3.2: 5×5 grid world example

We use a simple 5x5 gridworld environment to hilight each steps of the framework. To illustrate how an agent gains a positive example, suppose the agent takes an action "right" to move from (2,4) to (3,4) cell, as shown on Figure 3.2 on the right. All other alternative states that the agent could have ended up by taking different actions (down, up, and left) are in the exclusions. Context examples are the state that the agent is before taking an action and surrounding walls information. The following positive example is generated.

```
#pos({state_after((3,4))},
     {state_after((2,4)),state_after((1,5)),state_after((0,4)),state_after((1,4))},   (3.2)
     {state_before((2,4)). action(right). wall((1, 4)). wall((4, 2)).})
```

This example will be used to learn how to move up as one of the agent's hypotheses. Similarly, the agent is at (4,4) and tries to move right, as shown on the left on Figure 3.2. In this case, however, there is a wall at (5,4) and therefore the agent ends up in the same state. From this example, the following positive example is generated:

```
#pos({state_after((4,4))},
     {state_after((4,3)),state_after((3,4)),state_after((5,4)),state_after((4,5))}   (3.3)
     {state_before((4,4)). action(right). wall((5,4)). wall((4,5)).}).
```

3.2.2 Background Knowledge

Having defined the positive examples, we now define necessary background knowledge for inductive learning. In order to learn valid move for each direction in the form of state transition, the definition of adjacent is given as follows:

$$\begin{aligned}&\mathsf{adjacent(right,\ (X{+}1,Y),(X,Y))\ {:}{-}\ cell((X,Y)),\ cell((X{+}1,Y)).}\\&\mathsf{adjacent(left,(X,Y),\ (X{+}1,Y))\ {:}{-}\ cell((X,Y)),\ cell((X{+}1,Y)).}\\&\mathsf{adjacent(down,\ (X,Y{+}1),(X,Y))\ {:}{-}\ cell((X,Y)),\ cell((X,Y{+}1)).}\\&\mathsf{adjacent(up,\ (X,Y),\ (X,Y{+}1))\ {:}{-}\ cell((X,Y)),\ cell((X,Y{+}1)).}\end{aligned}\tag{3.4}$$

In existing RL methods, the agent is only able to see the state where the agent is at, and this is an additional assumption that the agent is able to see surrounding states. These adjacent concepts themselves could be learnt using ILASP separately, but in this preliminary research we focus on learning valid move, and this extension will be discussed in Further Research in XX. In addition, *cell((X,Y))* is defined as follows:

$$\mathsf{cell((0..X,\ 0..Y)).}\tag{3.5}$$

where, 0..X defines the range of X coodinates and X and Y are the maximum state integers in X and Y coodinates respectively. For 5x5 gridworld environment, for example, cell is defined as cell((0..5, 0..5)).

Example 3.2.2. (Background Knowledge).
XXX

3.2.3 Language Bias

Hypotheses have to be within a subset of a search space Sm using a language bias. In our case, the seach space should contain any conditions that defines a state after the transition. In order to execute ILASP, a *search space* of possible hypotheses is required, which is defined using a *language bias*.

$$\begin{aligned}&\mathsf{\#modeh(state\_after(var(cell))).}\\&\mathsf{\#modeb(1,\ adjacent(const(action),\ var(cell),\ var(cell)),\ (positive)).}\\&\mathsf{\#modeb(1,\ state\_before(var(cell)),\ (positive)).}\\&\mathsf{\#modeb(1,\ action(const(action)),(positive)).}\\&\mathsf{\#modeb(1,\ wall(var(cell))).}\end{aligned}\tag{3.6}$$

Without these in the form of mode bias, the search space for ILASP will be empty.
(positive) states that the predicates only appears as positive predicates and not negation as failure, therefore reducing the seach space. In this case, wall(var(cell)) could appears as ”not wall”. var is variables of type cell. action is constant, means it has

to be grounded as a particular action, because we want to learn different hypothesis for different action.

where *var(t)* and *const(t)* are a placeholder for variable and constant terms of type t respectively.

const(t) must be specified as #constant(t,c), where t is a type and c is a constant term. In our environment, action is specified as constant since ILASP should learn different hypothesis for each action.

```
#constant(action, right).
#constant(action, left).
#constant(action, down).        (3.7)
#constant(action, up).
cell((0..7, 0..6)).
```

As we describe in XXX, the search space increases in propotion to the complexity of learning tasks, which slows down the learning process. For example, the search space in this particular setting is in XX.

3.2.4 Learning Tasks

Throughout the learning process, the agent accumulates positive examples and learn hypothesis H.

Our learning task is to find a hypothesis H ¡ SM such that B U H has at least one answer set that extends at least one postiive example and none of the negative example.

In order to execute inductive learning using ILASP, the following definitions are supplied as well as the positive examples,

In addition to the above search space,

#max_penalty defines the maximum size of the hypothesis, by default it is 15. Increasing #max_penalty allows ILASP to learn longer hypothesis in expense of longer computation. #max_penalty(50).

Together with the above defition as well as accumulated positive examples, ILASP is able to learn an hypothesis. The quality of H depends on the experiences for the agent. For example, In the early phase of learning, the agent does not have many examples, and learns an hypothesis that may not be insightfull. For example, if the agent has only one positive example,

Next, the scope of *cell* are defined, as cell((0..X, 0..Y)), where X and Y are size of width and height respectively.

Finally, since our learning task is the rule of the game, which involve state transition, it needs to know how it means to be ”being next to XX”, Therefore the following assumptions are provided as background knowledge.

```
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
```
(3.8)

Learnt hypotheses are improved when a new positive example is added and the current hypothesis does not cover the new positive example. It it does cover it, there is no need to rerun ILASP again.

Example 3.2.3. (Inductive Learning). Using the same example as XXX

This definition itself could be learnt by setting another learning tasks, and it is a potential learning problem. However, we focus on learning task of the rule of the game in this paper.
The full details for ILASP learning tasks is described in Appendix XXX. Positive excludes the possibility of negation as a failure in order to reduce the search space.
Future research will relax these assumptions and attemp to learn more general hypothesis, e.g learning adjacent defintion.
These learnt H will be used to generate a plan in the abduction phase.
After executing the plan, the agent will have more positive examples, which will be used to improve the quality of H.
The learnt hypothesis is XXX
This hypothesis, for example, does not explain how to move ”down”. In order to learn how to move ”down”, it needs an positive example of moving up.
later on H improving as we collect more examples as well as background knowledge.

3.3 Planning with Answer Set Programming

3.3.1 Constructin of Answer Set Program

3.3.2 Plan Generation

The plan generation is executed only after the agent finds the goal. Untile the goal is found, the agent keeps exploraing randomly. In RL algorithm, the agent also needs the positive reward by reaching a termination state. Once the agent find the goal once, we can generate a plan using the current hypothesis by solving Answer Set Program.
If the hypotheses were not accurate, clingo might not generate all the actions leading to the goals.
The syntax of ASP is different from ILASP phase, because we need to include time sequence when solving ASP. In ILASP, it is only state_before and state_after, but in plan

generation, there will be more than one state transition. These syntax conversion needs to be done for learnt hypothesis

$$\begin{aligned}&1\{\mathsf{action(down,T);\ action(up,T);\ action(right,T);\ action(left,T);\ action(non,T)}\}1\\&\mathsf{:\text{-}\ time(T),\ not\ finished(T).}\end{aligned} \tag{3.9}$$

This choice rule states that action must be one of four actions (defined maximum and minimum numbers in 1), T is the time step at which the agent takes each action, unless *finished(T)* is grounded.
finished(T) is associated with goal definition, and it is defined as:

$$\begin{aligned}&\mathsf{finished(T)\text{:-}\ goal(T2),\ time(T),\ T \geq T2.}\\&\mathsf{goal(T)\text{:-}\ state\_at((5,\ 1),\ T),\ not\ finished(T\text{-}1).}\\&\mathsf{goalMet\text{:-}\ goal(T).}\\&\mathsf{:\text{-}\ not\ goalMet.}\end{aligned} \tag{3.10}$$

Example 3.3.1. (Plan Generation). As shown in example XX, the learnt example will be converted by adding time sequence for ASP plan generation.
Together with hypothesis, the background knowledge will used to solve for answer sets program.
However, since hypothesis is not complete, there is more than one answer set at each time step. Since one of the answer sets state_at is correct, the rest will be in the exclusions in the answer set, which further improve the hypothesis.
In this example, the following is the answer set program
The answer set using the hypothesis XXX is
XXX.
The answer set using the improved hypothesis is XXX,
Which correctly returns a sequence of actions and predicted states.

3.3.3 Plan Execution

The answer sets returned by clingo is a set of states and actions, which is the plan of the agent at each time step.
The set of states is of the form state_at((X,Y),T), where X and Y represent x-axi and y-axi in a maze respectively, T represents a time that the agent is at this particular X,Y cell.
action(A,T) tells which action the agent should take at each time. By following the actions, the agent should collect both predicted state that the agent will end up, and the observed state that the agent actually end up. If there is a difference between these two, either B or H do not correctly represent the model of the environment, so needs to be improved.
When the agent encounters a new environment (e.g a new wall), this new information will be added to its background, which will be used to improved the hypothesis.
Similarly, after executing an action by following the generated plan, the agent receives a new positive example. If the new positive example is not covered by the

current hypotheses, ILASP reruns using the new example to improve the hypotheses. next time ILASP gets executed.
For example,

$$\begin{array}{l} \mathsf{state\_at((1,1),1),\ action(right,1)} \\ \mathsf{state\_at((2,1),2),\ action(right,2)} \\ \mathsf{state\_at((3,1),3),\ action(right,3)} \\ \mathsf{state\_at((4,1),4),\ action(right,4)} \\ \mathsf{state\_at((5,1),5)}, \cdots \end{array}$$

At the start of the learning, H is usually not correct or too general, using this H will generate lots of answer sets that are not useful for the planning. These examples will be collected and included as exclusions of a new positive example.
To avoid the agent from being stuck in a sub-optimal plan, the agent deliberately discards the plan and takes an random action with a probability of epsilon (which is less than 1) TODO define this mathematically. When the agent deviates from the planning, it often discovers new information, which will be added to B. Exploration is necessary to make sure that the agent might discovers a shorter path than the current plan, which will be demonstrated in the experiment.
It builds the model of the environment by improving two internal concepts: hypothesis H and background knowledge B.
In the further research, we could experiment with a more sophisticated exploration strategy, such as XXX and YYY.
Everytime the agent executes an action by following the plan, it checks whether the observed state is that is expected.
If there is a difference between the two, either
B is incorrect
H is not sophisticated enough,
If that is the case, the agent runs ILASP again using more positive examples it collected during the plan execution.

3.4 Exploration

In RL, there is a tradeoff between exploration and exploitation. While the agent exploit what is already experienced and knows the best action selection so far, it also needs to explore by taking a new action to discover a new state, which might make the agent discover an even shorter path and therefore higher total rewards in the future or in the long term. This exploration-exploitation issues has been an active reseach area in RL. In our case, if the goal is found and the model is correct, it is likely that following the plan will maximize the total rewards,
ILP(RL) kicks in once the agent reaches a goal once. However it is likely that the agent has not seen all the environment and therefore is likely to be in a sub-optimal plan. Therefore, similar to RL algorithm, the agent also has to explore a new state. There are a number of exploration strategy in RL (such as Boltzman approach, Count-based and Optimistic Initial value TODO REFERENCE). One of the

most commonly used strategy is ϵ-greedy strategy. As described in Chapter XXX, the agent takes an random action
Another common way is to decay the ϵ by the number of episodes, since at the beginning it is more likely that the agent has not fully explored the environment and therefore higher epsilon.
While statistical RL algorithms, such as Q-learning or Tile-coding explained in XX, update Q-value by a facter of alpha, ILP(RL) tends to strictly follow the plan the generated. Also it is known that model-based RL tends to get stuck in a sub-optimal if the model is incorrect, ILP(RL) also needs a exploration And since the agent does not know when the perfect hypothesis in a particular environment is fully realised, it needs to keep exploring the environment even though
This strategy may not be appropriate in cases there safety is a priority (since it is random action.) It is simple to implement. In the case of ILP(RL), the agent discard the plan from the abduction with a probably of epsilon and takes a random action in order to avoid getting stuck in a sub-optimal path. When the agent takes an random action and move into a new state, the agents creates a new plan from the new state and continue to move forward.
This exploration point will be highlighted in Experiment XXX.
Epsilon needs to be larger than Q-learnig because
The reason for using random exploration is that it can be used for both benchmark and ILP(RL) and thus enables us to do a fair comparision between them.

Example 3.4.1. (Plan Generation). Suppose the plan of the agent is the following.
In our implementation, exploration strategy is epsilon, so with the probablity of XX, the agent discards the plan and randomly selects an actions.
Suppose the agent decides to take an random action and moves ”up”, which may help it discover a new state. From the new state, the agent again plan to the goal from the current state. In this case, the new plan is
XXX.
After taking an random action, the agent again has a probably of epsilon for taking an random action. Throughout following the new plan, there is a small probalbility of chance that the agent takes an random action.

The current framework simply uses a simple random exploration, therefore even if the agent takes an random action and goes to a different state other than the planed one, the agent does the replan from the new state and quickly correct to the original plan path. This means it is likely that the agent of ILP(RL) only explores the adjacent cells and if there is a shorter path or new state
far from the current state, the agent is unlikely be able to find the new state unless epsilon value is very high.

3.5 Implementation

3.5.1 Technology

The framework of ILP(RL) have been implemented using Python 3

Python was selected as the programming language to impelement the framework, which is based on the fact that it is one of the commonly used language among reinforcement learning reseach, and it works in the experimental platform.
The benchmark to be compared with my framework is also implemented using Python.
for the whole framework described in **??**, where ILASP 2i[1] is used for inductive learning and clingo 5 is used for solving answer sets for a plan.
ILASP version 2i, which is designed to scale with the numbers of examples.
ILASP cache caches relevant sets of examples, so everytime ILASP runs the same task except extra examples each time, ILASP runs from where it finished the learning last time and start from there rather than going through all the examples again. The code is available in https://github.com/921kiyo/ILPRL.
The bottleneck for the learning in terms of learning time is hypothesis improvement. In order to optimise it,
ILASP 2i
matplotlib is a plotting libray that provides charts of our experiments.

3.5.2 Experiment Platform

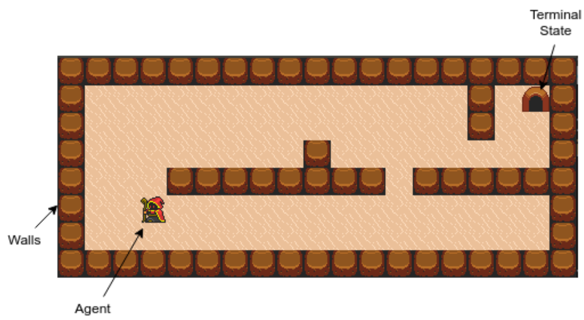

Figure 3.3: The VGDL (Video Game Definition Language) environment example

We use the Video Game Definition Language (VGDL), which is a high-level description language for 2D video games providing a platform for computational intelligence research ([**?**]). The VGDL allows users to easily craft their own environments, which makes us possible to do various experiments without relying on a default environment. The VGDL platform provides an interface with OpenAI Gym ([**?**]), which is a commonly used benchmark platform. The base game is a simple maze as shown in Figure 3.3. There are 3 different types of cells: a goal cell, walls and paths. The agent can take 4 different actions: up, down, right and left. The environment is not known to the agent in advance, and it attempts to find the goal by exploring the environment. In all experiments, the agent receives -1 in any states except the goal state, where it gains a reward of 10. Once the agent reaches the goal, or termination state, that episode is finished and the agent start the next episode from the starting point.

[1]https://sourceforge.net/projects/spikeimperial/files/ILASP/

Chapter 4

Evaluation

4.1 Setting

4.1.1 Evaluation Metrics

The two main measurements for the performance of our new architecture are learning efficiency and transfer learning capability. The learning efficiencies are measured in two different ways. First, the performance ILP(RL) is compared with exisitng RL algorithms in terms of convergence rate, which is measured in terms of number of episodes that the agent needs to get to an optimal policy. Second, the convergence of learning by ILASP is measured in terms of the number of hypothesis improvement devided by the total number of hypothesis improvement at episode 0. The reason we are measuring it at only episode 0 is that empirically the agent learns the target hypothesis at episode 0 and there is no hypothesis refinement after episode 0. This gives a normalised convergence rate of ILASP learning with the maximum 1.

4.1.2 Benchmarks

We use two existing RL methods as benchmarks: Q-learning and tile-coding. Q-learning is widely used RL technique, and given the environments used for the experiments are discrete and deterministic, this method is sufficient enought for our experiment.

Another benchmark is tile coding, which is a type of linear function approximation techniques described in Chapter XX. The reason for using an extra benchmark is that the comparision with q-learning might not be a fair comparision, since ILP(RL) has one extra assumption: the agent knows surrounding information (whether there are walls in adjacent cells), which is not a common assumption for Q-learning. Thus we incorporate the same surrounding information as features, and update the weights of each feature as a learning. We compare the performance of ILP(RL) with these two methods.

Parameter	ILP(RL)	Benchmarks
The number of episode	100	100
Time steps per episode	250	250
The number of experiments	30	30
Alpha	N/A	0.5
Epsilon	0.1	0.1

Table 4.1: List of parameters used in the experiments

4.1.3 Parameters

All the matrices used in the experiments are summarised in Table 4.1. Epsilon for ILP(RL) should be higher, since the agent follows the generated plan, whereas benchmark algorithms update value function with the degree of alpha. We conducted several experiments using different environments to highlight each aspect of the algorithm.
Since the performance of the agent is affected by the randomness of the exploration, and ILP(RL) is highly dependent on how quickly the agent finds the goal, each experiment is conducted 30 times and the perform is averaged across the experiments. At each episode, we also measure the performance without exploration to see the pure optimal policy.

4.2 Learning Evaluation

4.2.1 Experiment Result 1

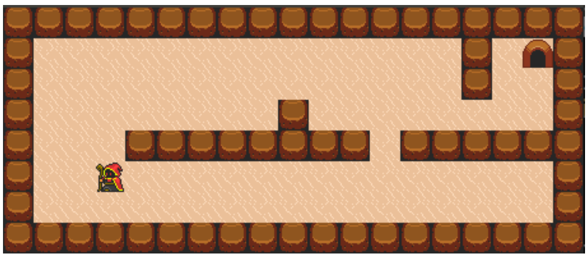

Figure 4.2: Game enviroment for experiment 1

The purpose of the first experiment is how the algorithm learns the model of the environment, or hypothesis in ILASP. The environment are defined as a simple maze where the goal is located the right uppper corner as shown in Figure 4.2.
The shortest path is taking the lower path instead of the upper one.
Figure 4.3 shows the traning performance between ILP(RL) and Q-learning. The convergence rate of ILP(RL) is faster than Q-learnig: ILP(RL) reaches the maximum reward between 40 and 50 episodes, whereas Q-learning reaches the same level at between 60 and 70 episodes. This is because unlike Q-learning where the value function is updated with the rate of alpha, whereas ILP(RL) gradually builds the model

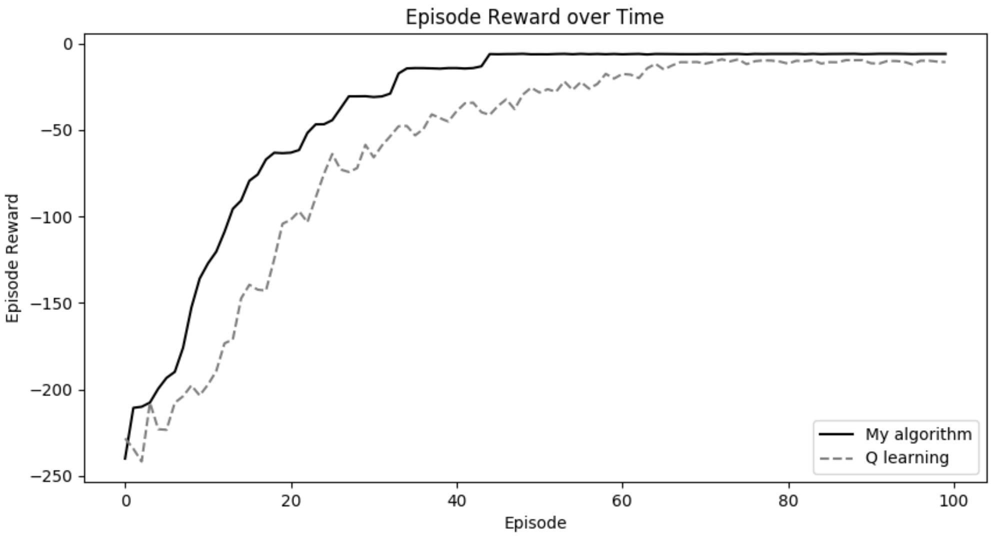

Figure 4.3: Result of experiment 1 (learnig curve)

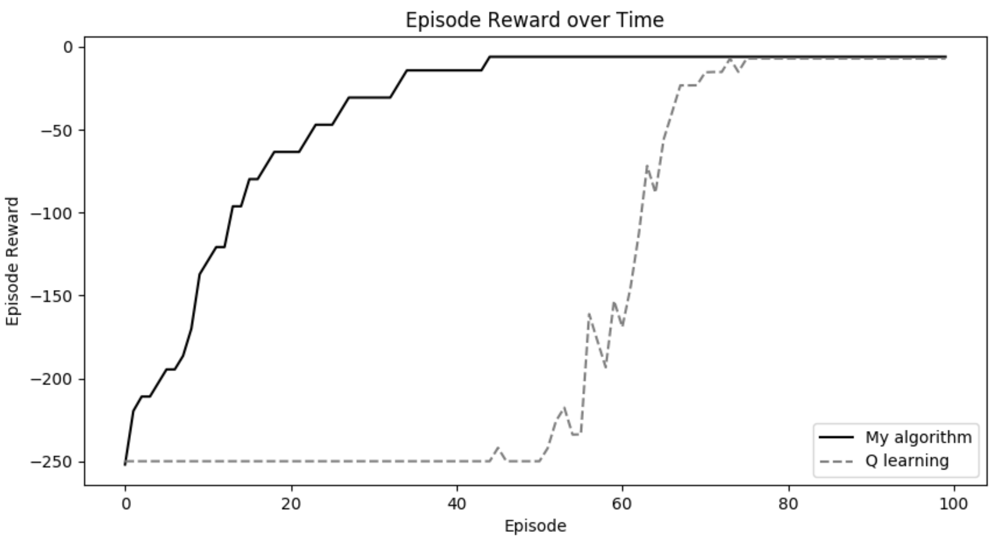

Figure 4.4: Result of experiment 1 (performance)

of the environment and use the background knowledge to accurately plan. This result is also consistent with the general notion that model-based learnig (ILP(RL)) is more data-efficient than model-free learning (Q-learning). The same trend is also shown in Figure 4.9, where we measure only the performance of the policy without random exploration.

Overall this results shows that ILP(RL) converges to the optimal policy faster than benchmarks in a simple scenarios, achieving more data-efficient learning.

In addition to the data-efficient learning, what the agent has learnt with ILP(RL) is expressive. Learnt hypotheses are shown in 4.2.1, which is the rule of the game and easy to understand for human users. Since the learnt hypothesis is a general concept, which can be used in a different environmet. This transfer learning capability is also described in Experiement 3 and 4.

```
state_after(V1) :- adjacent(right, V0, V1), state_before(V1), action(right), wall(V0).
state_after(V0) :- adjacent(right, V0, V1), state_before(V0), action(left), wall(V1).
state_after(V1) :- adjacent(down, V0, V1), state_before(V1), action(down), wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
```
(4.1)

In addition, we plot the learning convergence for ILASP at episode 0 in Figure 4.5, measured in terms of the number of hypothesis refinement to reach the final hypothesis as shown in 4.2.1. This shows that the agent quickly learns the hypothesis at the episode 0. The reason that the agent reaches the maximum reward at between 40 and 50 episodes, is mostly dependent on how quickly the agent finds the goal location, which enables it to plan. Since our exploration strategy is expilon random choice, there is a promissing that a better exploration strategy further accelerates the learning process.

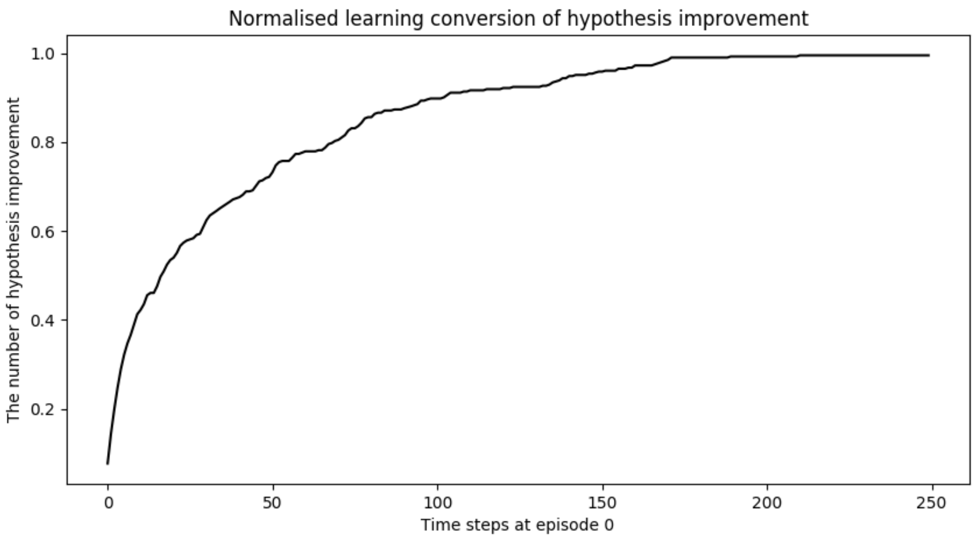

Figure 4.5: Normalised learning convergence by ILASP for experiment 1

4.2.2 Experiment Result 2

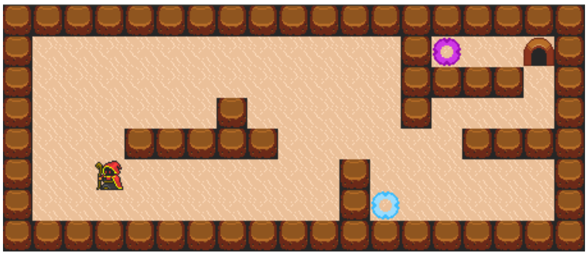

Figure 4.6: Game enviroment for experiment 1

Experiment 2 was conducted to see if the agent find a optimal path of using a teleport. In the environment shown in Figure 4.10, there are two ways to reach the goal: using a normal path to get the goal located on the top right corner, or using a telport. The environment is designed such that using a teleport is a shorter path and therefore gives higher total reward. Compared to Experiment 1, two extra search spaces and concepts are added as follow:

#modeb(1, link_start(var(cell)), (positive)).
#modeb(1, link_dest(var(cell)), (positive)).

Where teleport links are added to the environment. The teleport link is one-way: link_start takes the agent to link_dest, but link_dest does not take the agent back to link_start. The allows ILASP to learn additional hypothesis. The full learning task for this experiment is in Appendix XX.
Once the agent steps onto a state where link_start is located, it gets two positive experiences. In this game environment, the agent moves two cells in one time step instead of one cell per time step.
Also link_start and link_dest need to be stored in background knowledge rather than as contex examples, because ILASP needs to learn different hypothesis for link and non-link case. link locations need to be available for all positive examples so that ILASP correctly learn non-link, which is shown in Figure XX below.
The training performance shown in XX, which converges faster than XX.

state_after(V1) :- link_dest(V1).
state_after(V0) :- link_dest(V0), state_before(V0), action(right).
state_after(V1) :- adjacent(left, V0, V1), state_before(V0), action(right), not wall(V1).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V0), action(down), not wall(V1).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V1) :- adjacent(down, V0, V1), state_before(V1), action(down), wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
(4.2)

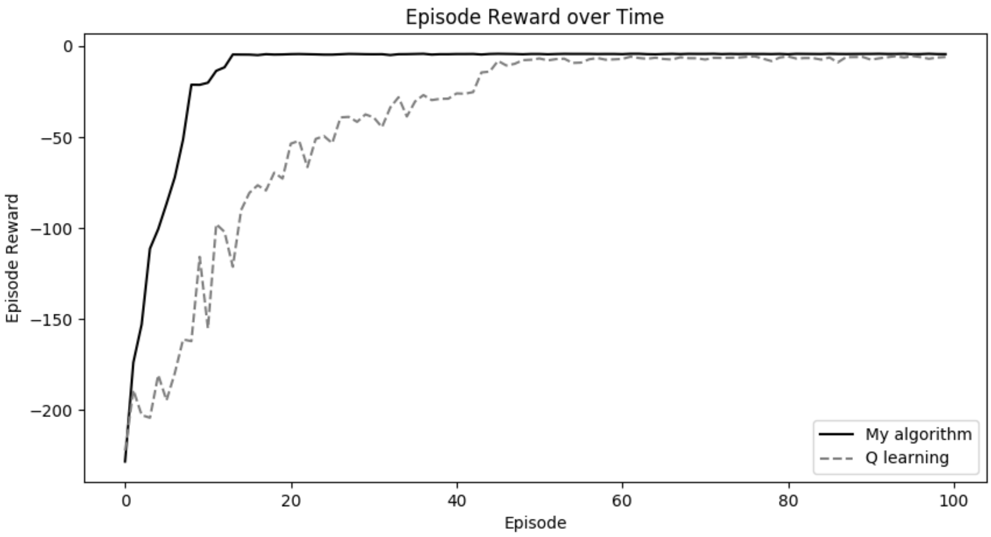

Figure 4.7: Result of experiment 2 (learnig curve)

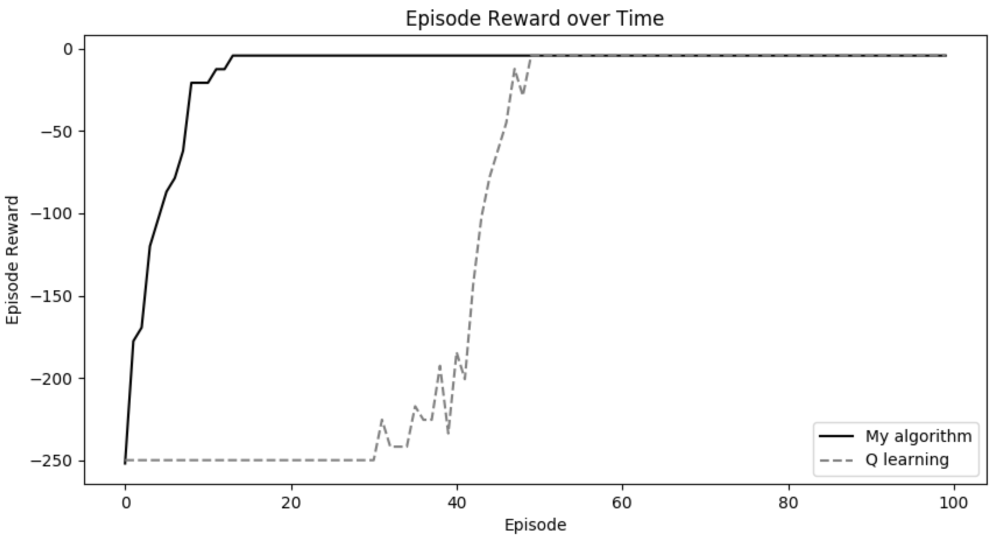

Figure 4.8: Result of experiment 2 (performance)

To highlight the learning the new concept of teleport link, Figure 4.2 is an intermediate incomplete hypothesis learnt by ILASP. These hypotheses are generated just after the agent steps onto the link. However, the first hypothesis says when link_dest is available state_after is true. Since link_dest is available in background knowledge rather than context, when solving for answer sets to generate a plan, it generates incorrect state_after at every time step. However, as shown in Algorithms XX, these generated state_after are all incorrect and therefore will be added to exclusions of the next positive examples. These exclusions will later refines hypotheses and results in Figure 4.3, the final complete hypotheses.

Learnt hypotheses are as follow:

$$
\begin{array}{l}
\text{state\_after(V1) :- link\_start(V0), link\_dest(V1), state\_before(V0).}\\
\text{state\_after(V0) :- link\_dest(V0), state\_before(V0), action(right).}\\
\text{state\_after(V1) :- adjacent(left, V0, V1), state\_before(V0), action(right), not wall(V1).}\\
\text{state\_after(V0) :- adjacent(left, V0, V1), state\_before(V1), action(left), not wall(V0).}\\
\text{state\_after(V1) :- adjacent(up, V0, V1), state\_before(V0), action(down), not wall(V1).}\\
\text{state\_after(V0) :- adjacent(up, V0, V1), state\_before(V1), action(up), not wall(V0).}\\
\text{state\_after(V1) :- adjacent(left, V0, V1), state\_before(V1), action(left), wall(V0).}\\
\text{state\_after(V1) :- adjacent(down, V0, V1), state\_before(V1), action(down), wall(V0).}\\
\text{state\_after(V1) :- adjacent(up, V0, V1), state\_before(V1), action(up), wall(V0).}
\end{array}
\tag{4.3}
$$

Compared the Experiment 1, there are two new hypotheses due to the presence of the teleport links. These learnt hypotheses are also applicables to an environment where there is no link, such as a game in experiment 1. In this case, the first two hypotheses in Figure XX are never be used since the body predicates relating to link_start(V0), link_dest(V1) are never be satisfied.

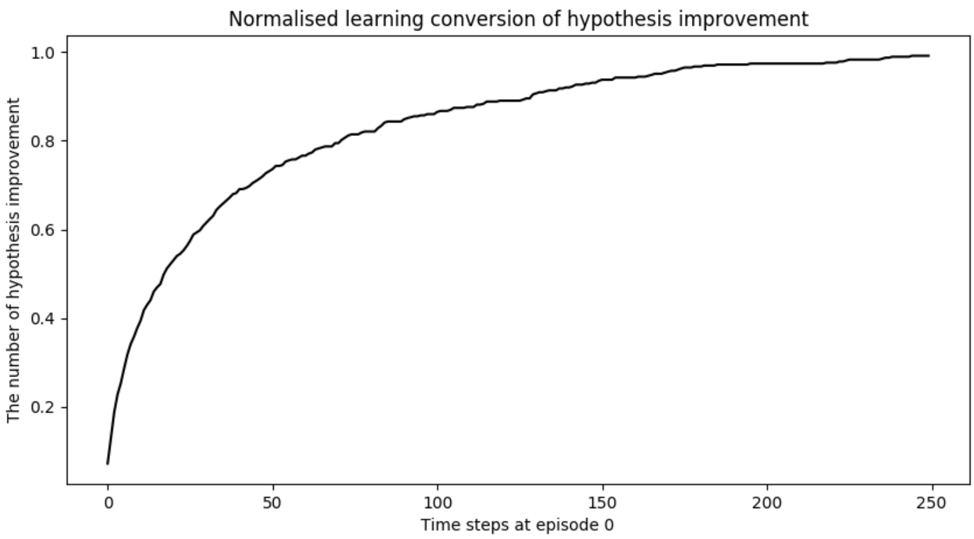

Figure 4.9: Normalised learning convergence by ILASP for experiment 2

4.3 Transfer Learning Evaluation

4.3.1 Experiment Result 3

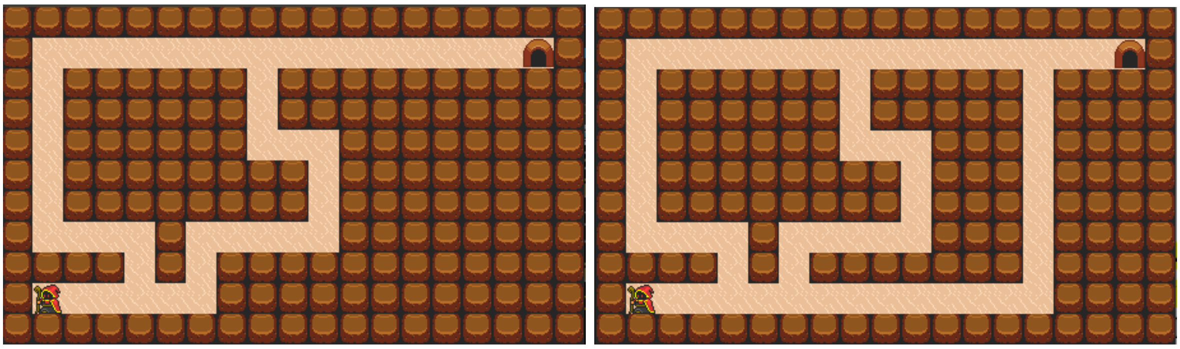

Figure 4.10: Game environment for experiment 3: before (left) and after (right) transfer learning

In Experiment 3, we investigated the potentials of transfer learning betweeen similar environments. We trained the agent using the environment on the left in Figure **??**, and transfer the learnt hypothesis as well as positive examples to a new environment. The learnt hypothesis is valid move of the game and a general concept that is applicable to any similar games. Positive examples are also transferred since if there is a new concept that the agent needs to learn in a new environment, the agent needs to refine the hypothesis by running ILASP, thus the all the positice examples are also transferred as well as hypotheses. Background knowledge are not transferred since these information are different in a new environment. The agent starts with an empty background knowledge in the new environment and gradually collects them as it explore the environment. The goal position is the same as in the first game and we assume that the transferred agent already knows the goal location, but the routes to the goal may be different. While this is a limited transfer learning since the goal position is known in advance, this is still a useful transfer in cases where the rest of the environment changes. In this experiment, we compare the two learning performance: one with transfer learning and one without it. The result is shown in Figure 4.11 and 4.12.

These are the hypotheses we are transferring to a new environment. Since the complete hypothesis is already known to the agent, it can do planning from the beginning.

state_after(V0) :- adjacent(right, V0, V1), state_before(V1), action(right), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(down, V0, V1), state_before(V0), action(up), not wall(V1).
state_after(V0) :- adjacent(down, V0, V1), state_before(V1), action(down), not wall(V0).
state_after(V1) :- adjacent(right, V0, V1), state_before(V1), action(right), wall(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V0), action(down), wall(V1).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).
(4.4)

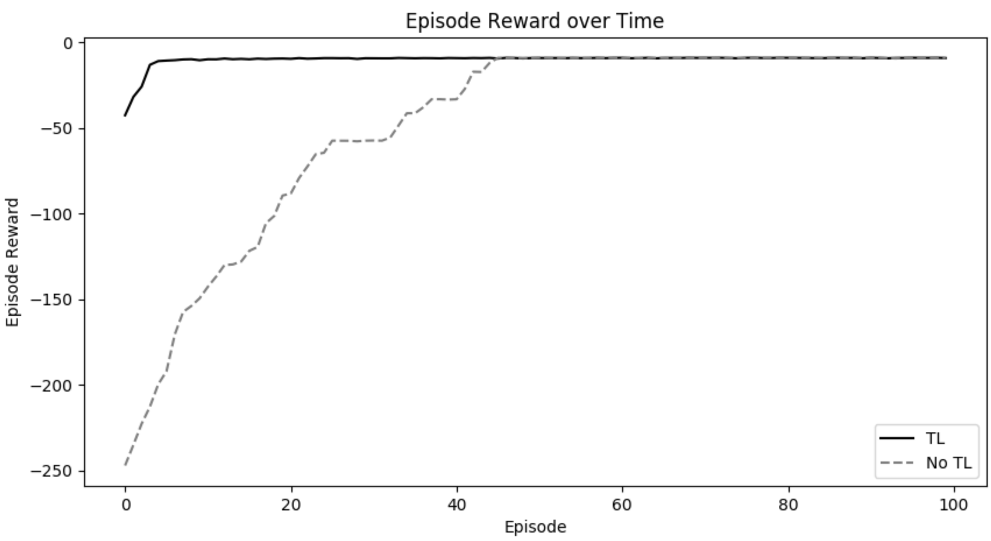

Figure 4.11: Result of experiment 3 (learnig curve)

4.3.2 Experiment Result 4

Finally the hypothesis is transferred to a new environment where there is a new concept that did not exist in the first environment and therefore the agent needs to learn it after the hypothesis is transferred.

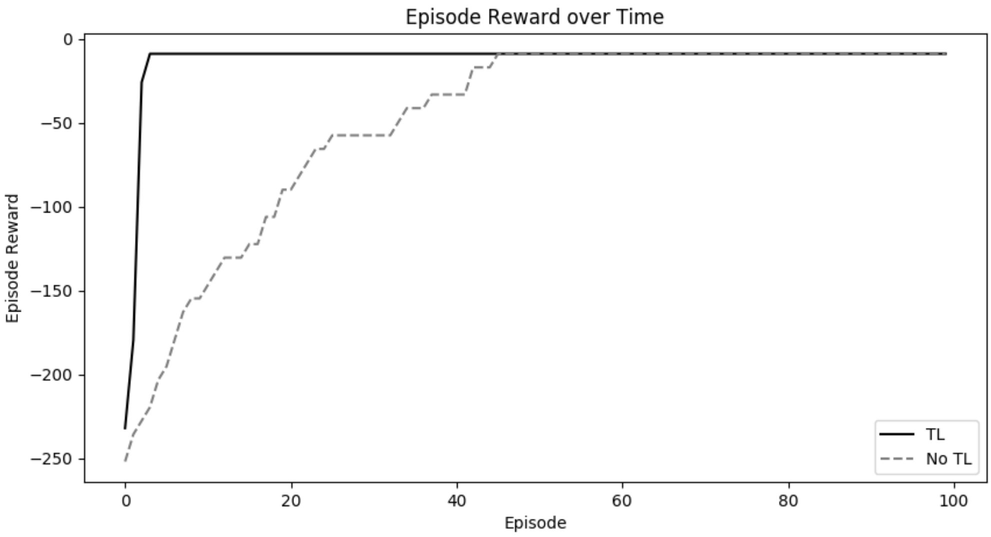

Figure 4.12: Result of experiment 3 (performance)

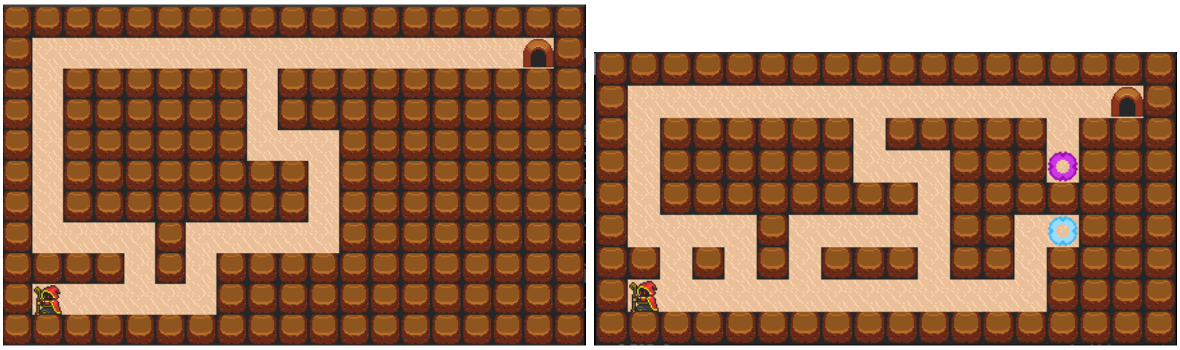

Figure 4.13: Game environment for experiment 4: before (left) and after (right) transfer learning

state_after(V1) :- link_start(V0), link_dest(V1), state_before(V0).
state_after(V1) :- adjacent(left, V0, V1), state_before(V0), action(right), not wall(V1).
state_after(V0) :- adjacent(left, V0, V1), state_before(V1), action(left), not wall(V0).
state_after(V1) :- adjacent(up, V0, V1), state_before(V0), action(down), not wall(V1).
state_after(V0) :- adjacent(up, V0, V1), state_before(V1), action(up), not wall(V0).
state_after(V0) :- adjacent(left, V0, V1), state_before(V0), action(right), wall(V1).
state_after(V1) :- adjacent(left, V0, V1), state_before(V1), action(left), wall(V0).
state_after(V0) :- adjacent(up, V0, V1), state_before(V0), action(down), wall(V1).
state_after(V1) :- adjacent(up, V0, V1), state_before(V1), action(up), wall(V0).

(4.5)

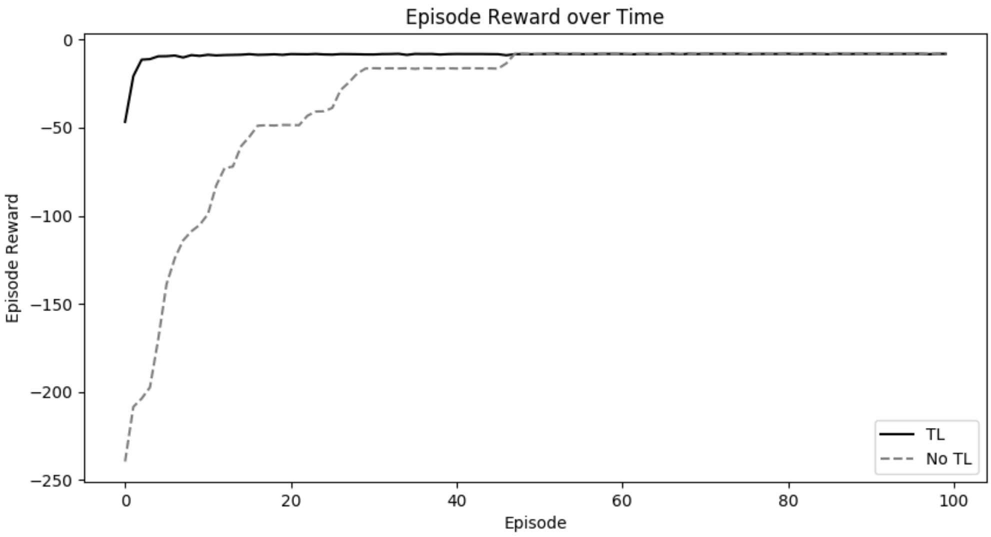

Figure 4.14: Result of experiment 4 (learnig curve)

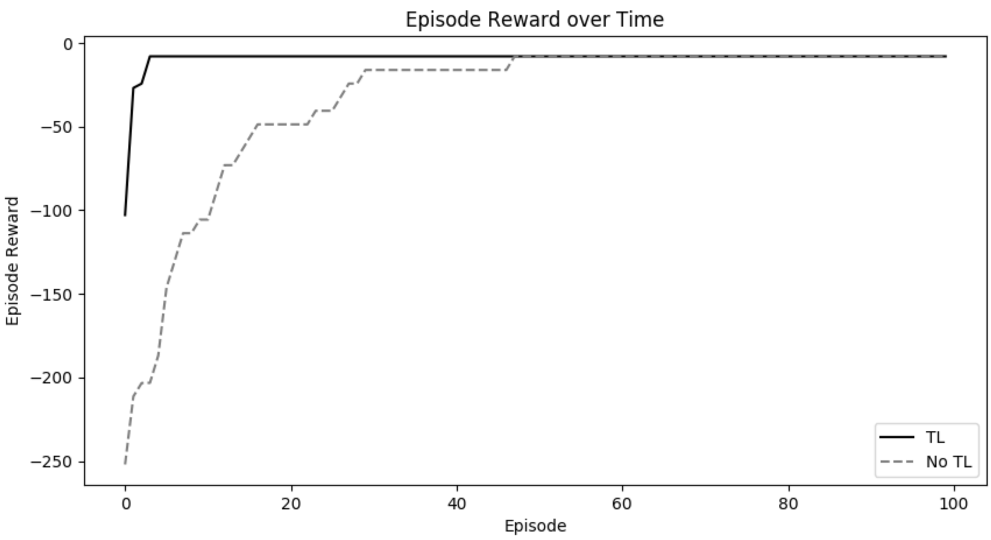

Figure 4.15: Result of experiment 4 (performance)

4.4 Strengths

To my knowledge, this is the first attemp that inductive logic programming is incorporated into a reinforcement learning senario to facilitate learning process. In simple environments, we show that the agent learns rule of the game faster than existing RL algorithms, learnt concepts is easy to understand for human users. We also show that the learnt hypothesis is a general concept and can be applied to other environment to mitigate learnig process.

The full hypotheses were learnt in the very early phase of learning and exploration phase. Thus with sufficient exploration, the model of the environment is correct and

therefore it is able to find the optimal policy/path.
We show that ILP(RL) is able to solve a reduced MDP where the rewards are assumed to be associated with a sequence of actions planned as answer sets. Although this is a limitated solution, there is a potential to expand it to solve full MDP as discussed in Further Research.
TODO more details on the strength of the algorithm. Validity

4.5 Limitations

Although this is the first time and inductive logic programming is applied into reinforcement leaning and there are new interesting property for ILP(RL), there are two major limitations with the current framework.

4.5.1 Scalability

The first limitation is scalability. As pointed in XXX or XXX, ILP framework is known to be less scalable. The current framework is tested in a relatively simple environments, and proven to be work better than RL algorithsm in terms of the number of episodes that is needed to converge to an optimal policy. However, learning in each episode is relatively slower than that of RL. This is shown in XXX, which shows average learnint time for ILASP.
This limitation is theoretically discussed in XXX, where the complexity of deciding satisfiability is $\sum_2^P$-complete. Since there is no negative examples used in our current framework, the complexity is NP-complete.
Whereas Q-learning update value function in the same way whether there is a new concept such as teleport links.
Figure XXX shows traning times for Experiment 1 and 2.
ILASP learning time for Experiment 1 and 2.
Unlike existing reinforcement learning, out algorithm refines hypothesis at every time steps within the same episode. Thus even though the efficiency in terms of the number of iteration is higher, training time within each iteration tends to be lower.

4.5.2 Flexibility

While most of existing reinforcement learning works in different kinds of environment without pre-configuration, our algorithm needs to define search space for learning hypothesis. As explained in the experiment 3, it was necessary to add two extra modeb before training. Thus the algorithm may not be feasible in cases where these learning concepts were unknown or difficult to define. In addition, not only it needs search space, surrounding information is assumed to be known to the agent. While this assumption may be reasonable in many cases, this is not common in traditional reinforcement learnig setting.
The current framework does not make use of rewards the agent collects and mainly uses the location of the goal for planning. In some senarios, there may not be a

termination state (goal) and instead there may be a different purpose to gain these rewards. Since the current implementation is dependent on finding the goal for planning rather than maximing total rewards, which is the common objective for most of RL algorithms, the application of the current framework may be limited to particular types of problems.

Another question remains to how to extend the framework to more realistic senarios. RL works in more complex environments such as 3D or real physical environment, whereas the experiences of the agent in the current framework need to be expressed as ASP syntax, thus expressing continuous states rather than discrete states is challenging.

Chapter 5

Related Work

In this section, I summarise recent studies related to symbolic (deep) reinforcement learning.
[?] introduced Deep Symbolic Reinforcement Learning (DSRL), a proof of concept for incorporating symbolic front end as a means of converting low-dimensional symbolic representation into spatio-temporal representations, which will be the state transitions input of reinforcement learning. DSRL extracts features using convolutional neural newtworks (CNNs) [?] and an autoencoder, which are transformed into symbolic representations for relevant object types and positions of the objects. These symbolic representations represent abstract state-space, which are the inputs for the Q-learning algorithm to learn a policy on this particular state-space. DSRL was shown to outperform DRL in stochastic variant environments. However, there are a number of drawbacks to this approach. First, the extraction of the individual objects was done by manually defined threhold of feature activation values, given that the games were geometrically simple. Thus this approach would not scale in geometrically complex games. Second, using deep neural network front-end might also cause a problem. As demonstrated in [?], a single irrelevant pixel could dramatically influence the state through the change in CNNs. In addition, while proposed method successfully used symbolic representations to achieve more data-efficient learning, there is still the potential to apply symbolic learning to those symbolic representations to further improve the learning efficiency, which is what we attemp to do in this paper. [?] further explored this symbolic abstraction approach by incorporating the relative position of each object with respect to every other object rather than absolute object position. They also assign priority to each Q-value function based on the relative distance of objects from an agent.
[?] added relational reinforcement learning, a classical subfield of research aiming to combining reinforcement learning with relational learning or Inductive Logic Programming, which added more abstract planning on top of DSRL approach. The new mode was then applied to much more complicate game environment than that used by [?]. This idea of adding planning capability align with our approach of using ILP to improve a RL agent. We explore how to effectively learn the model of the environment and effectively use it to facilitate data-efficient learning and transfer learning capability.
Another approach for using symbolic reinforcement learning is storing heuristics

expressed by knowledge bases [[**?**]). An agent lerans the concept of *Hierarchical Knowledge Bases (HKBs)* (which is defined in more details in [**?**] and [**?**]] at every iteration of training, which contain multiple rules (state-action pairs). The agent then is able to decide itself when it should exploit the heuristic rather than the state-action pairs of the RL using *Strategic Depth*. This approach effectively uses the heuristic knowledge bases, which acts as a sym-symbolic model of the game.

Another field related to our research is the combining of ASP and RL. The original concept of combining ASP and RL was in [**?**], where they developed an algorithm that efficiently finds the optimal solution of an MDP of non-stationary domains by using ASP to find the possible trajectories of an MDP. This approach focused more on efficient update of the Q function rather than inductive learning. In order to find stationary sets, an extension of ASP called BC^+, an action language, was used. BC^+ can directly translate the agent's actions into ASP form, and provide sequences of actions in answer sets.

Chapter 6

Conclusion

6.1 Summary of Work

In this paper, we developed a new RL algorithm by applying ILP to develop a new learning process. We used a latest ILP algorithm called ILASP, Learning from Answer Set Program to iteratively improve hypotheses.

6.2 Further Research

Having stated the limitations of the current framework, we discuss some of the possible impròments and further research in this section.
This is a proof of concept, a new type of model-based reinforcement learning using inductive logic programming.
More complicated environment
More general transfer learning.
Only empirically correct, no theoreticaly guarantee
Dynamic environment like moving enermy etc.
Non-stationality possible to be handled??
Our approach is similar to experience replay ??
More promising approach is to combine RL algorithm and using ILP approach to complement each other, rather than replacing the bellman equation altogether.

6.2.1 Value Iteration Approach

The proposed architecture is not finalised and will be reviewed regularly as we do more research. More research needs to be devoted to finalising the overall architecture, and the following issues in particular need to be considered.

6.2.2 Weak Constraint

- Further investigation of whether ILASP can learn the concept of adjacent, which is crucial concept to know in any environment.

- How to generalise the agent's model when the environment changes. The new environment could be very similar to the previous one, or could be a completely different environment thus the agent should create a new internal model rather than generalising the existing model.
- The current proposed architecture is based on Dyna with simulated experiences. However, this might not be the best overall architecture, and the feasibility of using simulated experience with the learnt model with ILASP needs to be further investigated.
- Possibility of using other representational concepts such as *Predictive Representations of State* or *Affordance* [**?**] for the agent's learning task. These concept have not been considered at the moment, but could help better transfer learning.
- Preparation for a backup plan in case ILASP approach does not work, so that the researchs feasible within 3 months of the researcheriod.

6.2.3 Generalisation of the Current Approach

Learning the concept of being adjacent

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015. pages

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. pages

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," apr 2015. pages

[4] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards Deep Symbolic Reinforcement Learning," sep 2016. pages

[5] V. Lifschitz, "What Is Answer Set Programming?.," *Aaai*, vol. 8, pp. 1594–1597, 2008. pages

[6] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," *5th International Conf. of Symp. on Logic Programming*, no. December 2014, pp. 1070–1080, 1988. pages

[7] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 107–124, 2011. pages

[8] S. Muggleton, "Inductive logic programming," *New Generation Computing*, vol. 8, no. 4, pp. 295–318, 1991. pages

[9] L. De Raedt, "Logical settings for concept-learning," *Artificial Intelligence*, vol. 95, no. 1, pp. 187–201, 1997. pages

[10] S. Muggleton, "Inductive Logic Programming: derivations, successes and shortcomings," *SIGART Bulletin*, vol. 5, pp. 1–5, 1993. pages

[11] C. Sakama and K. Inoue, "Brave induction: A logical framework for learning from incomplete information," *Machine Learning*, vol. 76, no. 1, pp. 3–35, 2009. pages

[12] R. P. Otero, "Induction of Stable Models," *Conference on Inductive Logic Programming (ILP)*, pp. 193–205, 2001. pages

[13] K. Inoue, T. Ribeiro, and C. Sakama, "Learning from interpretation transition," *Machine Learning*, vol. 94, no. 1, pp. 51–79, 2014. pages

[14] D. Corapi, A. Russo, and E. Lupu, "Inductive Logic Programming in Answer Set Programming," *Inductive Logic Programming*, pp. 91–97, 2012. pages

[15] M. Law, A. Russo, and K. Broda, "Inductive Learning of Answer Set Programs," *European Conference on Logics in Artificial Intelligence (JELIA)*, vol. 2, no. Ray 2009, pp. 311–325, 2014. pages

[16] M. Law, A. Russo, and K. Broda, "Iterative Learning of Answer Set Programs from Context Dependent Examples," aug 2016. pages

[17] S. Ray and P. Tadepalli, "Model-Based Reinforcement Learning," *Encyclopedia of Machine Learning*, pp. 690–693, 2010. pages

[18] R. S. Sutton, "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming," *Machine Learning Proceedings 1990*, vol. 02254, no. 1987, pp. 216–224, 1990. pages

[19] Watkins, "Learning from Delayed Reward." pages

[20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," jun 2016. pages

[21] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," oct 2017. pages

[22] A. d'Avila Garcez, A. R. R. Dutra, and E. Alonso, "Towards Symbolic Reinforcement Learning with Common Sense," apr 2018. pages

[23] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia, "Relational Deep Reinforcement Learning," jun 2018. pages

[24] D. Apeldoorn and K.-I. Gabriele, "An Agent-Based Learning Approach for Finding and Exploiting Heuristics in Unknown Environments," *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning*, pp. 1–8, 2017. pages

[25] D. Apeldoorn and G. Kern-isberner, "When Should Learning Agents Switch to Explicit Knowledge ?," vol. 41, pp. 174–186, 2016. pages

[26] D. Apeldoorn and G. Kern-isberner, "Towards an Understanding of What Is Learned : Extracting Multi-Abstraction-Level Knowledge from Learning Agents," pp. 764–767. pages

[27] L. A. Ferreira, R. A. C. Bianchi, P. E. Santos, and R. L. de Mantaras, "Answer Set Programming for Non-Stationary Markov Decision Processes," may 2017. pages

[28] M. Sridharan, B. Meadows, and R. Gomez, "What can I not do? Towards an Architecture for Reasoning about and Learning Affordances," *International Conference on Automated Planning and Scheduling*, no. Icaps, pp. 461–469, 2017. pages

Appendix A

Ethics

To our best knowledge, there is no particular ethical considerations for this particular research listed in Table XX. However, the field of RL ia an active research area and has been increasingly applied in industries these days, and therefore ethical frameworks for RL will be required for both academic research as well as industry applications.
Also the experiemnts of our algorithm were conducted using a game environment rather than real applications (e.g robots).
Rather compared to existing reinforcement learning methodologies.
Also there are a number of AI researchers discussing the ethics of AI in general. Since RL is considered to be part of AI research, these ethical considerations might be also applied.

	Yes	**No**
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓

Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓

Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Table A.1: Ethics Checklist

Appendix B

Learning tasks

This is the full learning task for ILASP in the experiment 1.

```
state_after(V1) :- link_dest(V1).
cell((0..7, 0..6)).
adjacent(right, (X+1,Y),(X,Y)):- cell((X,Y)), cell((X+1,Y)).
adjacent(left,(X,Y), (X+1,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(down, (X,Y+1),(X,Y)) :- cell((X,Y)), cell((X,Y+1)).
adjacent(up, (X,Y), (X,Y+1)) :- cell((X,Y)), cell((X,Y+1)).
#modeh(state_after(var(cell))).
#modeb(1, adjacent(const(action), var(cell), var(cell))).
#modeb(1, state_before(var(cell)), (positive)).
#modeb(1, action(const(action)),(positive)).
#modeb(1, wall(var(cell))).
#max_penalty(50).
#constant(action, right).
#constant(action, left).
#constant(action, down).
#constant(action, up).
```

Appendix C

Solving for ASP

This is the full learning task for ILASP in the experiment 1. The syntax and time are added for planning purpose.
TODO put comment on the code

```
1{action(down,T); action(up,T); action(right,T); action(left,T); action(non,T)}1 :-
time(T), not finished(T).
#show state_at/2.
#show action/2.
finished(T):- goal(T2), time(T), T ¿= T2.
goal(T):- state_at((5, 1), T), not finished(T-1).
goalMet:- goal(T).
:- not goalMet.
time(0..30).
cell((0..6, 0..5)).
#minimize1, X, T: action(X,T).
adjacent(right, (X+1,Y),(X,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(left,(X,Y), (X+1,Y)) :- cell((X,Y)), cell((X+1,Y)).
adjacent(down, (X,Y+1),(X,Y)) :- cell((X,Y)), cell((X,Y+1)).
adjacent(up, (X,Y), (X,Y+1)) :- cell((X,Y)), cell((X,Y+1)).
state_at((1, 4), 3).
```