Agent Learning in Relational Domains based on Logical MDPs with Negation

Song Zhiwei and Chen Xiaoping
Department of Computer Science, University of Science and Technology of China, Hefei, China
Email:{songzw, xpchen}@ustc.edu.cn

Cong Shuang
Department of Automation, University of Science and Technology of China, Hefei, China
Email: scong@ustc.edu.cn

Abstract— **In this paper, we propose a model named *Logical Markov Decision Processes with Negation* for Relational Reinforcement Learning for applying Reinforcement Learning algorithms on the relational domains with the states and actions in relational form. In the model, the logical negation is represented explicitly, so that the abstract state space can be constructed from the goal state(s) of a given task simply by applying a generating method and an expanding method, and each ground state can be represented by one and only one abstract state. Prototype action is also introduced into the model, so that the applicable abstract actions can be obtained automatically. Based on the model, a model-free $\Theta(\lambda)$-learning algorithm is implemented to evaluate the state-action-substitution value function. We also propose a state refinement method guided by two formal definitions of self-loop degree and common characteristic of abstract states to construct the abstract state space automatically by the agent itself rather than manually. The experiments show that the agent can catch the core of the given task, and the final state space is intuitive.**

Index Terms— **Relational Reinforcement Learning, Logical MDPs with Negation, $\Theta(\lambda)$-Learning, State Refinement**

I. Introduction

One of important challenges for designing an intelligent agent in *Artificial Intelligence* (AI) is the *decision-making* of the agent in the dynamic systems. *Markov Decision Processes* (MDPs) and its extensions have become the *de facto* standards of it. With relaxing some assumptions including determinism and complete knowledge of the domain, MDPs further leads to the theory of *Reinforcement Learning* (RL), with which an agent can learn the policy by interacting with environment without supervisors [1], [2]. It is widely agreed that the intelligent agent should have the the ability of learning so that it can adapt to the chang of the dynamic environment. And there are many successful applications of RL algorithms, such as information retrieval and medical diagnosis [3]–[5]. However, classical representations in RL require explicit state and action, and the space of states grows exponentially with the number of domain features, which limits their applications to large-scale problems. To address this problem, *Relational Reinforcement Learning* (RRL) was proposed [6], [7]. It concerns using of RL in relation domains with the states and actions in relational form naturally. It appears that people usually take advantage of rich relational structure in learning and generalization. RRL also facilitates formulating broad collections related tasks as a single domain, and reusing the learnt knowledge to these related tasks [8]. Up to now, several preliminary models and a number of algorithms of RRL have been proposed [9]–[19]. Among these work, Kersting and De Raedt [12] presented *Logical Markov Decision Processes* (LOMDPs), in which states are represented by logical formulas with variables, named *abstract state*, leading to a compact representation to the ground states so that the agent can learn the policy in the abstract level. However, the model also has some gaps on the representation, for example, one ground state can be represented by more than one abstract states.

In this paper, a new model *Logical Markov Decision Process with Negation* (nLMDP) is proposed inspired by LOMDPs, in which both definitions of state space and action space are different from LOMDPs. And the most distinct feature of the new model is that the abstract state space holds the property of complementarity by introducing the *negation* of logical languages. The complementary state space means each ground state can be represented by one and only one abstract state, constructed from the goal state(s) simply by applying a generating method and an expanding method. By contrast, in LOMDPs, a ground state can be represented by several candidate abstract states, and a manual pre-given total order of abstract states is used to select the desirable one. Another distinct feature is that *Prototype Action*, a super abstraction over general

This paper is based on "Unique State and Automatical Action Abstracting based on Logical MDPs with Negation" and "States Evolution in $\Theta(\lambda)$-Learning based on Logical MDPs with Negation," by Song Z.W., and Chen X.P., which appeared respectively in the Proceedings of the Second International Conference on Natural Computation (ICNC'06), Xi'an, China, September 2006, and the Proceedings of the 2007 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2007), Montréal, Quebec, Canada, October 2007, © 2007 IEEE.

abstract actions, is defined. Thus, given an abstract state, applicable abstract actions with valid substitutions can be obtained automatically. Furthermore, we define a value function of state-action-substitution Θ, and implement an algorithm of $\Theta(\lambda)$-*leaning* to evaluate the values of the substitutions.

Another important contribution of this paper is that we propose a state refinement method to construct the complementary abstract state space automatically by the agent itself, so that the agent can grasp the environment while it learns the policy. The core idea is applying the expanding method online, in which a state is selected and splitted into two new states according to a new *expanding conjunction*. We define the *self-loop degree* and the *common characteristic* for the abstract state formally, and take them as the criteria to select the state and find the expanding conjunction, respectively. In the learning process, the abstract state with maximal self-loop degree is selected and splitted into two new states by using the common characteristic as the expanding conjunction. So the abstract state space is expanded until it is suitable for the task.

This paper is organized as follows. Section II will briefly introduce some logical and MDP preliminaries. After proposing the nLMDP in section III, the $\Theta(\lambda)$-learning is presented in section IV. The state refinement algorithm is presented in section V. Section VI shows the experimental results and discussions. Some related work are discussed in section VII, and the paper ends with conclusions in section VIII.

II. Preliminaries

In the remainder of this paper, the terminology of logical languages and MDPs will be used. We now briefly introduce some main notions. For more details, see [20] and [2].

A. Logic

An *alphabet* Σ is a set of relation symbols, $\mathcal{P}$, and a set of *constants*, $\mathcal{C}$. If the arity $m(m \geq 0)$ of a relation symbols $\mathtt{p} \in \mathcal{P}$ is 0 then $\mathtt{p}$ is called a proposition. A *term* t is a variable $\mathtt{X}$, or a constant $\mathtt{c}$. An *atom* $\mathtt{p}(\mathtt{r_1}, \cdots, \mathtt{r_m})$ is a relation symbol $\mathtt{p}$ followed by a bracketed m-tuple of terms $\mathtt{t_i}$. A *conjunction* is a set of atoms. A *substitution* $\theta = \{\mathtt{X_1}/\mathtt{t_1}, \cdots, \mathtt{X_n}/\mathtt{t_n}\}$ is a set of assignments of terms $\mathtt{t_i}$ to variables $\mathtt{X_i}$. If all terms of θ are variables, $\theta = \{\mathtt{X_1}/\mathtt{Y_1}, \cdots, \mathtt{X_n}/\mathtt{Y_n}\}$, and $\mathtt{Y_i}(\mathtt{i} = \mathtt{1} \cdots \mathtt{n})$ are different variables, then θ^{-1} is a reverse substitution $\{\mathtt{Y_1}/\mathtt{X_1}, \cdots, \mathtt{Y_n}/\mathtt{X_n}\}$. A conjunction A is said to be θ-subsumed by a conjunction B, denoted as $A \preceq_\theta B$, if there exists a substitution θ such that $B\theta \subseteq A$. A *formula* is an atom or a conjunction. The variables of a term or a formula A is denoted as $vars(A)$. The terms of a formula A is denoted as $terms(A)$. A term, or a formula is called *ground* when it contains no variables. To highlight A is ground, a bar is added, i.e. $\bar{A}$, in this paper. The *Herbrand base* of Σ, denoted as $\mathtt{HB}^\Sigma$, is the set of all ground atoms which can be constructed with the predicate symbols and constants of Σ. An *interpretation* of a formula is a subset of $\mathtt{HB}^\Sigma$.

In logic programming, the logical *negation*, denoted as $\mathtt{not}$, is often used. There is a little difference between it and the classical negation $\neg$ that $\mathtt{not}$ implies that there is a *closed world assumption* [21]. $\mathtt{not}A$ is true, iff A can not be inferred. For a conjunction A, $\mathtt{not}A$ is called *not-conjunction* in this paper. A not-conjunction is also a formula. A set of several conjunctions and not-conjunctions is a formula, too.

B. Markov Decision Processes

A *Markov Decision Process* is a tuple $M = \langle S, A, T, R\rangle$, where S is a set of states, A is a set of actions, $T : S \times A \times S \to [0, 1]$ is a *transition model*, and $R : S \times A \times S \to \mathbb{R}$ is a reward model with real values. The set of actions applicable in a state $s \in S$ is denoted as $A(s)$. A transition from state s to s' caused by action $a \in A(s)$ occurs with a reward $R(s, a, s')$ and probability $T(s, a, s')$ which holds $\sum_{s'} T(s, a, s') = 1$. A *policy* $\pi : S \times A \to [0, 1]$ is a mapping from states, $s \in S$, and actions, $a \in A(s)$, to the probability $\pi(s, a)$ of taking action a when in state s.

Given an MDP $M = \langle S, A, T, R\rangle$, a policy π for M, and a *discount factor* $\gamma \in [0, 1]$, the *state value function* $V^\pi : S \to \mathbb{R}$ is the expected long-range rewards following the policy π. The *state-action value function* $Q^\pi : S \times A \to \mathbb{R}$ can be defined similarly. A policy π^* is optimal if $V^{\pi^*}(s) \geq V^{\pi'}(s)$ for all $s \in S$ and all π'. Optimal value functions are denoted as V^* and Q^*. From the *optimality equations*,

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\right]$$

basically all methods for solving MDPs can be derived.

A *Relational MDP* is now defined by the tuple $\langle S, A, T, R\rangle$ which is similar to MDP while the states and actions are in relational form [22]. The basic ingredients of it are a set of relational predicates $\mathcal{P}$, a set of action predicates $\mathcal{A}$, and a domain of constants $\mathcal{C}$. The Herbrand base $\mathtt{HB}^{\mathcal{P}\cup\mathcal{C}}$ is the set of all ground atoms which can be constructed from $\mathcal{P}$ and $\mathcal{C}$. The set of all states then consists of the power set of $\mathtt{HB}^{\mathcal{P}\cup\mathcal{C}}$. However, the illegal states must be excluded by an implicit logical background theory $\mathtt{BK}$. Thus the state space S of the RMDP is $\{s | s \in \mathscr{P}(\mathtt{HB}^{\mathcal{P}\cup\mathcal{C}}), \mathtt{BK} \models s\}$, denoted as $S^{\mathcal{P}\cup\mathcal{C}}$ in this paper. Similarly, an action space A can be constructed based on $\mathcal{A}$ and $\mathcal{C}$.

III. Logical Markov Decision Processes with Negation

Because the states of the RMDP are in ground level, the size of the state space grows exponentially. As an example of the blocks world, based on the relational predicates

$\{\texttt{on/2}, \texttt{clear/1}\}$, the action predicates $\{\texttt{move/2}\}$, and the domain $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{floor}\}$, the blocks world containing 5 blocks has 501 legal states. So applying RRL in relational domains must abstract over the ground level.

A. Abstract State

Definition 1 (Abstract State): An abstract state s is a formula $\left\{s^\top, \texttt{not}\, s_1^\perp, \cdots, \texttt{not}\ s_n^\perp\right\}$, which contains one conjunction $s^\top$ and several ($n \geq 0$) not-conjunctions $\texttt{not}\, s_i^\perp (i = 1, \cdots, n)$, and for which there holds that $a \notin s_i^\perp$ for any atom $a \in s^\top$. $s^\top$ is called a *fit-conjunction*, and $s_i^\perp$ are called a *non-fit-conjunctions*.

An abstract state represents a set of ground states. We say that the ground state $\bar{s}$ is represented by an abstract state $s = \left\{s^\top, \texttt{not}\, s_1^\top, \cdots, \texttt{not}\, s_n^\perp\right\}$, denoted as $\bar{s} \preceq_\theta s$, iff:

there exists a substitution θ such that $\bar{s} \preceq_\theta s^\top$ and doesn't exist a substitution φ such that $\bar{s} \preceq_\varphi [s^\top \cup s_i^\perp]$ for any i (i=1,$\cdots$,n).

It means that the state $\bar{s}$ must fit conjunction $s^\top$ and can't fit conjunctions $s_i^\perp$. $\bar{S}(s)$ denotes the states represented by the abstract state s.

From the above definition, $\{\texttt{p(X)}, \texttt{not}\{\texttt{p(X)}\}\}$ is an illegal abstract state and $\{\texttt{p(X)}, \texttt{not}\{\texttt{p(Y)}\}\}$ is a legal abstract state although $\texttt{p(X)}$ and $\texttt{p(Y)}$ can have the same semantic. To constrain the semantics of two atoms are different, we take $\neq$ on two variables as the only constraint except for the domain state predicates.

The goal abstract state of tasks can be got easily. For one, the goal abstract state of the *stack* task of the blocks world (putting all blocks on one stack) is

$$\langle s_* \rangle = \{\{\texttt{on(A,B)}, \texttt{clear(A)}\}, \texttt{not}\{\texttt{on(C,D)}, \texttt{clear(C)}, \texttt{C} \neq \texttt{A}\}\}.$$

Note that the number of blocks is not specified. The ground state $\bar{s} = \{\texttt{on(a,b)}, \texttt{on(b,floor)}, \texttt{clear(a)}, \texttt{clear(floor)}\}$ containing two blocks is an interpretation of $\langle s_* \rangle$.

To represent non-goal states, two abstract states can be generated from the goal abstract state $\langle s_* \rangle$:

$$\langle s_1 \rangle = \{\varnothing, \texttt{not}\{\texttt{on(A,B)}, \texttt{clear(A)}\}\},$$
$$\langle s_2' \rangle = \{\{\texttt{on(A,B)}, \texttt{clear(A)}, \texttt{on(C,D)}, \texttt{clear(C)}, \texttt{C} \neq \texttt{A}\}, \varnothing\}.$$

For abstract state $s = \left\{s^\top,\ \texttt{not}\, s_1^\perp, \cdots, \texttt{not}\, s_n^\perp\right\}$, the *generating method* is:

1) if $s^\top \neq \varnothing$, take fit-conjunction $\varnothing$ and non-fit-conjunction $s^\top$ as the first generated abstract state from s;
2) chose k different non-fit-conjunctions to the fit part while other non-fit-conjunctions reserved as the rest of $(2^n - 1)$ generated abstract states from s.

The set just containing an abstract state and all generated abstract states from it, is called the *generated abstract state space* from the state.

Theorem 1 (Global Complementarity): Under the closed world assumption, the generated abstract state space from an abstract state is complementary, that is the interpretations of all abstract states in it cover the whole ground state space $S^{\mathcal{P}\cup\mathcal{C}}$, and the intersection of each two interpretations is an empty set.

Proof: Under the closed world assumption, each ground state $\bar{s} \in S^{\mathcal{P}\cup\mathcal{C}}$ contains all the true ground atoms of that time. Given a state $\bar{s}$, for an abstract state $s_0 = \{s^\top, \texttt{not}\, s_1^\perp, \cdots, s_n^\perp\}$, each conjunction of $s^\top$ and $s_i^\perp$ must have a true or false value. If $s^\top (\neq \varnothing)$ is false then the first generated abstract state s_1 is true and all the others $(s_0, s_2, \cdots, s_{2^n})$ are false. Then assume $s^\top$ is true, if all $s_i^\perp$ are false, then s_0 is true, else there must exist only one generated abstract state s_j $(2 \leq j \leq 2^n)$ whose fit-conjunction just contains $s^\top$ and all the true $s_i^\perp$, and s_j will be true. We now see $\bigcup_{j=0}^{2^n} \bar{S}(s_j) = S^{\mathcal{P}\cup\mathcal{C}}$ and $\bar{S}(s_j) \cap \bar{S}(s_k) = \varnothing$ $(j \neq k)$, because the interpretations have been restricted to $S^{\mathcal{P}\cup\mathcal{C}}$. ■

Sometimes a task has more than one goal state. For example, a task is the block a must be on the top or bottom of a stack, and the goal states are:

$$\langle s_*' \rangle = \{\{\texttt{clear(a)}\}, \varnothing\}$$
$$\langle s_*'' \rangle = \{\{\texttt{on(a,A)}, \texttt{on(B,a)}\}, \texttt{not}\{\texttt{on(A,C)}\}\}$$

Generally, for m different goal states of a given task, $s_i = \left\{s_i^\top,\ \texttt{not}\, s_{i1}^\perp, \cdots, \texttt{not}\, s_{in_i}^\perp\right\} (i = 1, \cdots, m)$, there are m different complementary generated state spaces from each goal state, so the objective is merging the m state spaces into one complementary state space.

Theorem 2 (Merging Complementarity): For m goal states, if only one of $s_i^\top (i = 1, \cdots, m)$ can be true, then all generated states except the first generated states from each goal state, m goal states, and the state $\{\varnothing, \texttt{not}s_1^\top, \cdots, \texttt{not}s_m^\top\}$ compose a complementary abstract state space, and is called the generated state space from the m goal states.

Proof: For any ground state $\bar{s}$, if all $s_i^\top$ $(i = 1, \cdots, m)$ are false, then $\bar{s}$ is represented by $\{\varnothing, \texttt{not}s_1^\top, \cdots, \texttt{not}s_m^\top\}$, else assume $s_j^\top$ is true and $s_i^\top (i = 1, \cdots, j-1, j+1, \cdots, m)$ are false, according to the last theorem, $\bar{s}$ is represented by one and only one state of the generated space from s_j without the first generated state. ■

For last example, the merged complementary state space contains two goal states $\langle s_*' \rangle, \langle s_*'' \rangle$, and the following states:

$$\langle s_1' \rangle = \{\varnothing,\ \texttt{not}\{\texttt{clear(a)}\}, \texttt{not}\{\texttt{on(a,A)}, \texttt{on(B,a)}\}\}$$
$$\langle s_2'' \rangle = \{\{\texttt{on(a,A)}, \texttt{on(B,a)}, \texttt{on(A,C)}\}, \varnothing\}.$$

Thus, the generating method is extended with the above merging process if a task has two or more goal states, while the complementarity is preserved.

For the stack task of the blocks world, abstract states $\langle s_* \rangle, \langle s_1 \rangle, \langle s_2' \rangle$ represent all states complementally. However, three abstract states is not enough. There must be a method to expand the abstract space and keep the complementarity. The *expanding method* is to add a new conjunction s^+ to certain abstract state $s =$

$\{s^\top, \mathtt{not}\, s_1^\perp, \cdots, \mathtt{not}\, s_n^\perp\}$, and replace it with two new abstract states s' and s'':

$$s' = \{s^\top \cup s^+, \mathtt{not}\, s_1^\perp, \cdots, \mathtt{not}\, s_n^\perp\}$$
$$s'' = \{s^\top, \mathtt{not}\, s^+, \mathtt{not}\, s_1^\perp, \cdots, \mathtt{not}\, s_n^\perp\}$$

The former is taking s^+ as a subset of fit-conjunction, and the latter is taking s^+ as a non-fit-conjunction. Obviously, the new conjunction s^+, named as *expanding conjunction*, must also follow the hold that $a \notin s^\top$ and $a \notin s_i^\perp$ for each atom $a \in s^+$.

Theorem 3 (Local Complementarity): The two new expanded abstract states are complementary for the original abstract state. That is, the interpretations of the two expanded abstract states are the same as that of the original abstract state, and the intersection of them is an empty set.

Proof: If s is true, then there must exist just one abstract state from s' or s'' that is true followed by the value of s^+. So $\bar{S}(s') \cap \bar{S}(s'') = \varnothing$ and $\bar{S}(s') \cup \bar{S}(s'') \supseteq \bar{S}(s)$. No matter s' or s'' is true, s is true. Then $\bar{S}(s') \cup \bar{S}(s'') \subseteq \bar{S}(s)$. ■

Adding expanding conjunctions $\{\mathtt{on(B,E)}\}$, $\{\mathtt{on(D,F)}\}$, and $\{\mathtt{on(G,H)}, \mathtt{clear(G)}, \mathtt{G} \neq \mathtt{A}, \mathtt{G} \neq \mathtt{C}\}$ to abstract state $\langle s_2' \rangle$ in turn, the following abstract states are produced:

$$\begin{aligned}
\langle s_2 \rangle = \{\{&\mathtt{on(A,B)}, \mathtt{clear(A)}, \mathtt{on(C,D)}, \mathtt{clear(C)}, \mathtt{C} \neq \mathtt{A},\\
&\mathtt{on(B,E)}, \mathtt{on(D,F)}, \mathtt{on(G,H)}, \mathtt{clear(G)}, \mathtt{G} \neq \mathtt{A},\\
&\mathtt{G} \neq \mathtt{C}\}, \varnothing\}\\
\langle s_3 \rangle = \{\{&\mathtt{on(A,B)}, \mathtt{clear(A)}, \mathtt{on(C,D)}, \mathtt{clear(C)}, \mathtt{C} \neq \mathtt{A}\},\\
&\mathtt{not}\{\mathtt{on(B,E)}\}\}\\
\langle s_4 \rangle = \{\{&\mathtt{on(A,B)}, \mathtt{clear(A)}, \mathtt{on(C,D)}, \mathtt{clear(C)}, \mathtt{C} \neq \mathtt{A},\\
&\mathtt{on(B,E)}\}, \mathtt{not}\{\mathtt{on(D,F)}\}\}\\
\langle s_5 \rangle = \{\{&\mathtt{on(A,B)}, \mathtt{clear(A)}, \mathtt{on(C,D)}, \mathtt{clear(C)}, \mathtt{C} \neq \mathtt{A},\\
&\mathtt{on(B,E)}, \mathtt{on(D,F)}\}, \mathtt{not}\{\mathtt{on(G,H)}, \mathtt{clear(G)},\\
&\mathtt{G} \neq \mathtt{A}, \mathtt{G} \neq \mathtt{C}\}\}
\end{aligned}$$

Corollary 1 (Complementarity): Starting from a certain set of abstract states, and applying the above generating method once and the expanding method several times in turn, the new abstract space is complementary for $S^{\mathcal{P} \cup \mathcal{C}}$.

Normally, the certain set of abstract states is the goal abstract state(s) of a task, and the expanding method will not apply to it. These methods give designers a very useful tool to construct the abstract state space in an easy way. The only need for designers is the sequence of the expanding conjunctions.

B. Prototype Action

Given an abstract state, the applicable *abstract actions* of it can be obtained manually according to the preconditions of actions. For a simple case, consider abstract state $\langle s_2' \rangle$, the applicable abstract actions are $\mathtt{move(A,C)}$, $\mathtt{move(C,A)}$, $\mathtt{move(A,floor)}$, and $\mathtt{move(C,floor)}$. However, the manual method is very hard for more complex domains. To get abstract actions automatically, the *prototype action* will be introduced next.

Definition 2 (Prototype Action): A prototype action is an action atom $\mathring{a}$ with the precondition c and outcome O of it, denoted as $\mathring{a} : c \to O$. Where $\mathring{a}$ contains no constants, and c is in the form of abstract state.

Given an abstract state s and a ground state $\bar{s} \in \bar{S}(s)$, the semantics of the prototype action $\mathring{a} : c \to O$ is:

> *if there exist substitutions θ, φ, ψ such that $\bar{s} \preceq_\varphi s, \bar{s} \preceq_\psi c, \theta\varphi = \psi$ and the applicable ground action $\bar{a} = \mathring{a}\theta\varphi$ is executed then the outcome of the action is O.*

From the above semantics of the prototype action, we know that the applicable abstract actions are implicit and can be determined from the ground states represented by the abstract state. For abstract state s, if there exist a ground state $\bar{s} \in \bar{S}(s)$, a prototype action $\mathring{a}$, and three substitutions θ, φ, ψ such that $\bar{s} \preceq_\varphi s, \bar{s} \preceq_\psi c(\mathring{a}), \theta\varphi = \psi$ then $\mathring{a}\theta$ is an applicable abstract action for s, where $c(\mathring{a})$ is the precondition of $\mathring{a}$.

In model-free algorithms, the agent could know nothing about the outcome O and it could be omitted in the definition of prototype action. However, the agent often learns in a simulation environment, thus it could be useful to simulate the environment.

The outcome O can be specified by many methods or technics. A probabilistic STRIPS [23] implementation is defining O as a tuple $\langle P, D, E \rangle$, where $P = \{p_1, \cdots, p_m\}$ is the set of probabilities for m different cases by taking the action and holds $\sum_i p_i = 1$, $D = \{d_1, \cdots, d_m\}$ is the set of deleting conjunctions for m cases, and $E = \{e_1, \cdots, e_m\}$ is the set of adding conjunctions for m cases. d_i, e_i all are conjunctions and hold $vars(D, E) \subseteq vars(c) = vars(\mathring{a})$. The prototype action also can be denoted as $\mathring{a} : c \xrightarrow{p_i} e_i \setminus d_i$. For the abstract state s and the ground state $\bar{s} \in \bar{S}(s)$, the semantics of the STRIPS-like prototype action $\mathring{a} : c \xrightarrow{p_i} e_i \setminus d_i$ is:

> *If there exist substitutions θ, φ, ψ such that $\bar{s} \preceq_\varphi s, \bar{s} \preceq_\psi c, \theta\varphi = \psi$ and the applicable ground action $\bar{a} = \mathring{a}\theta\varphi$ is executed then the successor ground state should be $\bar{s}' = [\bar{s} \setminus d_i\theta\varphi] \cup e_i\theta\varphi$ with probability p_i.*

Intuitively, prototype actions should be the basic action ways of a domain. Indeed, a prototype action is a super abstract action, and there is only one prototype action $\mathtt{move(X,Y,Z)}$ in blocks world. It means moving the block $\mathtt{X}$ form block $\mathtt{Y}$ to $\mathtt{Z}$. Consider the probability 0.1 the agent cannot pickup the block $\mathtt{X}$ and 0.1 the block $\mathtt{X}$ fall on $\mathtt{floor}$ when it is in air, the only prototype action $\langle \mathring{a}_* \rangle$ is $\mathtt{move(X,Y,Z)}$:

$$\left\{\begin{array}{c}\mathtt{on(X,Y)},\\ \mathtt{clear(X)},\\ \mathtt{clear(Z)}\end{array}\right\}
\begin{array}{l}
\xrightarrow{0.8} \left\{\begin{array}{c}\mathtt{on(X,Z)},\\ \mathtt{clear(Y)}\end{array}\right\} \setminus \left\{\begin{array}{c}\mathtt{on(X,Y)},\\ \mathtt{clear(Z)}\end{array}\right\}\\
\xrightarrow{0.1} \{\} \setminus \{\}\\
\xrightarrow{0.1} \left\{\begin{array}{c}\mathtt{on(X,floor)},\\ \mathtt{clear(Y)}\end{array}\right\} \setminus \{\mathtt{on(X,Y)}\}
\end{array}$$

```
Input: ground state $\bar{s}$, value function $\Theta$
Output: abstract state $s$, substitution $\varphi$, prototype action
    $\mathring{a}$, substitution $\theta$
1: Find the unique abstract state $s$ and a substitution $\varphi$
   s.t. $\bar{s} \preceq_\varphi s$
2: Find a set of ground actions s.t. $\bar{A} = \{\mathring{a}\psi | \bar{s} \preceq_\psi c(\mathring{a})\}$
3: Select a set of ground actions from $\bar{A}$ s.t.
   $\bar{A}' = \{\mathring{a}\psi | (\mathring{a}, \psi) \in \bar{A}, terms(\mathring{a}\psi) \subseteq terms(s\varphi)\}$
4: Transform $\bar{A}'$ to applicable abstract actions $A$ s.t.
   $A = \{(\mathring{a}, \theta) | \mathring{a}\psi \in \bar{A}', \mathring{a}\psi = \mathring{a}\theta\varphi\}$
5: Chose a pair $(\mathring{a}, \theta)$ from $A$ based on $\Theta$ and a policy
   (e.g. softmax)
```

Figure 1. Algorithm of automatical action abstracting

C. nLMDPs

Definition 3 (Abstract Transition): An abstract transition is a function $T : S \times \mathring{A} \times \Theta \times S \to [0, 1]$. $T(s, \mathring{a}, \theta, s')$ means the probability of changing the abstract state from s to s' when the abstract action $\mathring{a}\theta$ is executed, and it holds $\sum_{s'} T(s, \mathring{a}, \theta, s') = 1$.

The probabilities in the STRIPS-like prototype action are hidden in the abstract transitions. For model-free algorithms, it is enough to only learn the transition probabilities.

The *abstract reward* and the *abstract policy* can be similarly defined.

Definition 4 (nLMDP): A Logical Markov Decision Process with Negation (nLMDP) is a tuple $M = \left\langle S, \mathring{A}, T, R \right\rangle$ where S is a given complementary abstract space, $\mathring{A}$ is a set of prototype actions, T is a finite set of abstract transitions based on S and $\mathring{A}$, and R is an abstract reward model.

An nLMDP is based on a given abstract state space and a given set of prototype actions while the valid substitutions can be determined. And the state space is complementary, so many RL algorithms based on an underlying MDP can be used in nLMDP directly.

IV. Θ(λ)-LEARNING

To learn an optimal policy based on the prototype action, there should have a value function for evaluating the triple of an abstract state, a prototype action, and a valid substitution. To distinguish the *state-action-substitution value function* $S \times \mathring{A} \times \Theta \to \mathbb{R}$ from state value function and state-action value function, it is also denoted as Θ. Indeed, Θ just divides the action of Q into a prototype action and a substitution. So the optimality equation for Θ is

$$\Theta^*(s, \mathring{a}, \theta) = \sum_{s'} T(s, \mathring{a}, \theta, s') \Big[R(s, \mathring{a}, \theta, s') + \gamma \max_{\mathring{a}'} \max_{\theta'} \Theta^*(s', \mathring{a}', \theta') \Big],$$

and the relation with V^* is

$$V^*(s) = \max_{\mathring{a}} \max_{\theta} \Theta^*(s, \mathring{a}, \theta).$$

```
1: $\Theta \leftarrow \varnothing$
2: for each episode do
3:   $\mathcal{E} \leftarrow \varnothing$
4:   Initialize ground state $\bar{s}$
5:   Get $(s, \varphi)$ and $(\mathring{a}, \theta)$ according to Fig. 1
6:   while $s$ is not the goal abstract state do
7:     Take action $\mathring{a}\theta\varphi$, observe reward $r$ and successor
       state $\bar{s}'$
8:     Get $(s', \varphi')$ and $(\mathring{a}', \theta')$ according to Fig. 1
9:     if $e(s, \mathring{a}, \theta)$ is not in $\mathcal{E}$ then
10:      Add $e(s, \mathring{a}, \theta) = 0$ to $\mathcal{E}$
11:    if $\Theta(s, \mathring{a}, \theta)$ is not in $\Theta$ then
12:      Add $\Theta(s, \mathring{a}, \theta) = 0$ to $\Theta$
13:    $\Theta^0 \leftarrow \Theta(s, \mathring{a}, \theta)$
14:    if $\Theta(s', \mathring{a}', \theta')$ is in $\Theta$ then
15:      $\Theta' \leftarrow \Theta(s', \mathring{a}', \theta')$
16:    else
17:      $\Theta' \leftarrow 0$
18:    $\delta \leftarrow r + \gamma\Theta' - \Theta^0$
19:    $e(s, \mathring{a}, \theta) \leftarrow e(s, \mathring{a}, \theta) + 1$
20:    for each triple $(s, \mathring{a}, \theta)$ in $\mathcal{E}$ do
21:      $\Theta(s, \mathring{a}, \theta) \leftarrow \Theta(s, \mathring{a}, \theta) + \alpha\delta e(s, \mathring{a}, \theta)$
22:      $e(s, \mathring{a}, \theta) \leftarrow \gamma\lambda e(s, \mathring{a}, \theta)$
23:    $s \leftarrow s', \varphi \leftarrow \varphi', \mathring{a} \leftarrow \mathring{a}', \theta \leftarrow \theta'$
```

Figure 2. Θ(λ)-learning

The first task of Θ-learning is to get the applicable abstract actions automatically. Fig. 1 shows the algorithm. The agent finds the ground actions $\bar{A}$ based on the precondition of prototype actions firstly, selects valid actions $\bar{A}'$ based on term constraints, transforms to abstract actions A, and chose a pair $(\mathring{a}, \theta)$ based on Θ value function and the softmax policy [2]:

$$p((\mathring{a}, \theta)|s) = \frac{e^{\Theta(s, \mathring{a}, \theta)/T}}{\sum\limits_{\mathring{a}'} \sum\limits_{\theta'} e^{\Theta(s, \mathring{a}', \theta')/T}}.$$

The second task of Θ-learning is to evaluate the value function Θ through iterations. Fig. 2 shows Θ(λ)-learning method, which uses an eligibility trace decay parameter λ similar with Sarsa(λ) [2]. To prevent invalid substitutions from occurring in Θ which is initialized with an empty set in line 1, and valid substitutions are added in real-time in line 12. In line 20-22, triples of the episode are updated, and $\mathcal{E}$ is used to record the triples in line 10 and initialized with an empty set in the beginning of each episode in line 3.

V. STATES REFINEMENT

Section III gives an easy way to construct a complementary abstract state space based on the generating method and the expanding method mentioned above. However, to obtain a suitable state space, it also needs a sequence of the expanding conjunctions given manually by human users.

In this section, a state refinement method is proposed in which the expanding conjunctions are obtained automatically in the learning process, and the suitable state space is constructed finally. The process includes two points. The first is how to select a target state, and the other is how to get the expanding conjunction to split the target state. They will be described and resolved in section V-A and V-B respectively, and section V-C gives the model-free state refinement method.

A. Target State Selecting

A satisfiable state space has two basic features,

- its size is suitable, and
- it is well-distributed, which means that each state has approximate equal opportunity to be visited by the agent.

If the partition of states is well-distributed, the size only reflects the presicion of the learning results, and we can stop expanding if the results are acceptable. So, we will focus on the latter feature.

Assuming the original state space satisfies the latter feature, in oder that the one-step expanded state space should also satisfy this feature, the only way is that the state which has maximal visiting opportunity is selected as the target state to split.

For state s, if there exists an applicable action such that the successive state is also s, we call s self-loop state simply. If the successor states are often itself, we must say that the state will have a high visiting opportunity. In this paper, we consider the self-loop degree as the criteria depicting visiting opportunity and the formal definitions are listed as follows.

Definition 5 (Self-Loop Degree): In an MDP, $l(s) = \sum_a \pi(s,a)T(s,a,s)$ is called the self-loop degree of the state s.

Definition 6 (Self-Loop State): A state s is a self-loop state, if its self-loop degree is greater than 0 ($l(s) > 0$).

Thus, we can always select the state which has maximal self-loop degree as the target state to be splitted in each expanding step.

B. Expanding Conjunction Obtaining

As is mentioned in section III, an abstract state represents several ground states. If we want to split the abstract state s, we must concentrate on partitioning the ground states $\bar{S}(s)$. There are two ways to partition the ground states $\bar{S}(s)$,

- find two distinct characteristics from $\bar{S}(s)$ and partition $\bar{S}(s)$ to two sets based on them, or
- find one common characteristic of $\bar{S}(s)$, that is many states of $\bar{S}(s)$ follow it, and partition $\bar{S}(s)$ to two sets based on it.

Obviously, the latter is easier than the former, and it fits the expanding method mentioned above which just needs one expanding conjunction.

Intuitively, the common characteristic should be abstract, that is, it includes variables, and should exclude atoms in s. In addition, it should be related with the terms (especially the variables) of s and be simple as mush as possible.

Therefore, we must generalize the ground states $\bar{S}(s)$ with variables of s, exclude the fit-conjunction of s, and replace other constants with new variables if the constants are not significant for a certain task. For examples, all constants will be replaced with new variables for *stack* or *unstack* task, but $\mathtt{a}, \mathtt{b}$ will not be replaced for $\mathtt{on}(\mathtt{a}, \mathtt{b})$ task.

Before giving the formal definition of the common characteristic, some preliminary definitions are proposed as follows.

Definition 7 (Relevance Term): Two terms x, y are relevant iff there exists an atom a such that $x \in terms(a)$ and $y \in terms(a)$. The set of the relevance terms of term t is denoted as $T(t)$.

Definition 8 (Relation Closure): Conjunction A is the relation closure of term t if $a \in A$ iff $terms(a) \subseteq T(t)$. The relation closure of term t is denoted as $R^*(t)$.

Definition 9 (V-Equivalence): Given a set of variables V, two formula A, B are V-equivalence ($A \equiv_V B$) iff there is a substitution θ such that $A\theta = B$ and all terms of θ are in V.

It's easy to proof that $\equiv_V$ is a reflexive, symmetric, and transitive relation. That is, $\equiv_V$ is an equivalence relation.

Now, we give the formal definition of the common characteristic of $\bar{S}(s) = \{\bar{s}_1, \cdots, \bar{s}_m\}$, also the common characteristic of $s = \{s^\top, nots_1^\perp, \cdots, nots_n^\perp\}$. There needs an infinite set of variables V which holds $vars(s) \cap V = \varnothing$. We generalize each ground state $\bar{s}_i$ to s_i by replacing different constants with different variables of V if they are insignificant. The result is denoted as $S(s) = \{s_1, \cdots, s_m\}$ and holds $vars(s_i) \cap vars(s_j) = \varnothing$ if $i \neq j$.

Definition 10 (Characteristic Space): The characteristic space of the abstract state s is $\{R^*(t) | s_i \in S(s),\ t \in terms(s^\top) \cup terms(s_i), s_i \preceq_\theta s^\top, R^*(t) \subseteq s_i\theta^{-1} \smallsetminus s^\top\}$.

Definition 11 (Common Characteristic): The common characteristic of s is the V-equivalence class which contains maximal characteristics of s.

To here, the common characteristic of the target state is got. Then, it can be used as the expanding conjunction to split the target state in the expanding method mentioned above.

C. State Refinement Progress

If the model of the environment is known clearly, a suitable abstract state space can be constructed based on the above analysis. However, in most situations, the models are unknown or partially available. Despite this, the above analysis can also lead us to a model-free algorithm. Both the self-loop degree and the common characteristic can be obtained in the trial-and-error interactions between the agent and the environment.

Fig. 4 shows the model-free learning algorithm, and the state space will be refined in line 10 by calling the state refinement process listed in Fig. 3, where $l(s)$ means

```
Input: $s, s', \bar{s}', S, \Theta, \mathcal{N}, \mathcal{L}, \rho$
Output: $S, \Theta, \mathcal{N}, \mathcal{L}, \rho$
 1: if $s' = s$ then
 2:    if $l(s)$ is not in $\mathcal{L}$ then
 3:       Add $l(s) = 0$ to $\mathcal{L}$
 4:    $l(s) \leftarrow l(s) + 1$
 5:    Find the characteristics $C$ of $s$ via the equation (1)
 6:    for each $c$ in $C$ do
 7:       if $n(s, c')$ in $\mathcal{N}$ and $c' \equiv_V c$ then
 8:          $n(s, c') \leftarrow n(s, c') + 1$
 9:       else
10:          Add $n(s, c) = 1$ to $\mathcal{N}$
11: if $l(s) \geq \rho$ then
12:    Choose $c^+$ s.t $n(s, c^+)$ has maximum value in $\mathcal{N}$
13:    Split $s$ to two states $s_1, s_2$ in $S$ via the expanding
       method based on the expanding conjunction $c^+$
14:    Duplicate all $\Theta$ values of $s$ to $s_1$ and $s_2$
15:    Remove all $\Theta$ values of $s$
16:    $\mathcal{N} \leftarrow \varnothing$
17:    $\mathcal{L} \leftarrow \varnothing$
18:    $\rho \leftarrow \omega\rho$
```

Figure 3. Progress of the state refinement.

the self-loop count of state s, $n(s, c)$ means the size of the characteristic class c of state s, ρ is the threshold of splitting state, and ω is the rate of the threshold increasement. Both $l(s)$ and $n(s, c)$ are computed from the real ground states visited by the agent. In line **??** of Fig. 3, we use equation

$$C = \left\{ R^*(t) \middle| t \in terms(s^\top) \cup terms(s''), s'' \preceq_\psi s^\top, R^*(t) \subseteq s''\psi^{-1} \setminus s^\top \right\} \quad (1)$$

to compute the characteristics of $\bar{s}'$. Where s'' is a generalization of $\bar{s}'$ by replacing the different insignificant constants with different variables of V, and $s^\top$ is the fit part of s.

VI. Experiments

We implemented $\Theta(\lambda)$-learning and the state refinement process using SWI-Prolog. The objective was to learn an abstract policy and a suitable complementary states for the stack task of blocks world. The task requires that all blocks should be moved to one stack, and the blocks world is the prototypical toy domain requiring relational representations and has been experimented usually in RRL. Two standard predicates `on/2` and `clear/1` were used, and the other predicates such as `block/1`, `floor/1`, `above/2`, `height/2` reported by [6], [24] were not used in our model or algorithm, while "$\neq$" is the only constraint on variables. Total 500 states of blocks world were generated randomly for 6 to 10 blocks (more than 58 million ground states) using the procedure described by [25], which guarantees that the initial states are followed by the background theory of blocks world. The initial ground state of each episode was selected randomly from them.

```
 1: Construct the initial abstract state space $S$ via the
    generating method
 2: $\Theta \leftarrow \varnothing$, $\mathcal{N} \leftarrow \varnothing$, $\mathcal{L} \leftarrow \varnothing$
 3: for each episode do
 4:    $\mathcal{E} \leftarrow \varnothing$
 5:    Initialize ground state $\bar{s}$
 6:    Get $(s, \varphi)$ and $(\mathring{a}, \theta)$ according to Fig. 1
 7:    while $s$ is not the goal abstract state do
 8:       Take action $\mathring{a}\theta\varphi$, observe reward $r$ and successor
          state $\bar{s}'$
 9:       Get $(s', \varphi')$ and $(\mathring{a}', \theta')$ according to Fig. 1
10:       Call state refinement algorithm listed in Fig. 3
11:       if state space changed then go to line 4
12:       if $e(s, \mathring{a}, \theta)$ is not in $\mathcal{E}$ then
13:          Add $e(s, \mathring{a}, \theta) = 0$ to $\mathcal{E}$
14:       if $\Theta(s, \mathring{a}, \theta)$ is not in $\Theta$ then
15:          Add $\Theta(s, \mathring{a}, \theta) = 0$ to $\Theta$
16:       $\Theta^0 \leftarrow \Theta(s, \mathring{a}, \theta)$
17:       if $\Theta(s', \mathring{a}', \theta')$ in $\Theta$ then $\Theta' \leftarrow \Theta(s', \mathring{a}', \theta')$
          else $\Theta' \leftarrow 0$
18:       $\delta \leftarrow r + \gamma\Theta' - \Theta^0$
19:       $e(s, \mathring{a}, \theta) \leftarrow e(s, \mathring{a}, \theta) + 1$
20:       for each triple $(s, \mathring{a}, \theta)$ in $\mathcal{E}$ do
21:          $\Theta(s, \mathring{a}, \theta) \leftarrow \Theta(s, \mathring{a}, \theta) + \alpha\delta e(s, \mathring{a}, \theta)$
22:          $e(s, \mathring{a}, \theta) \leftarrow \gamma\lambda e(s, \mathring{a}, \theta)$
23:       $s \leftarrow s'$, $\varphi \leftarrow \varphi'$, $\mathring{a} \leftarrow \mathring{a}'$, $\theta \leftarrow \theta'$
```

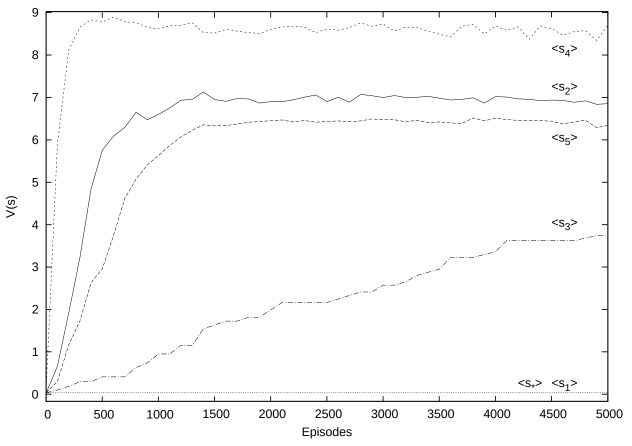
Figure 4. $\Theta(\lambda)$-learning with state refinement

A. $\Theta(\lambda)$-Learning

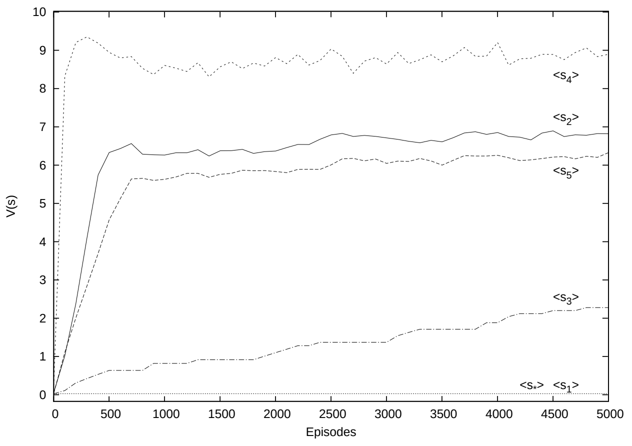
In the first experiment, the complementary abstract state space including six abstract states $\langle s_* \rangle, \langle s_1 \rangle - \langle s_5 \rangle$ listed in section III was used. It is constructed manually in a simple way by applying the generating method and the expanding method. It is also a well known state space in the planning community [25] for the blocks world. The agent gets reward 10 when it reaches the goal abstract state $\langle s_* \rangle$, while 0 when it reaches other states. The only prototype action $\langle \mathring{a}_* \rangle$ of the task have been listed in section III.

The discount factor γ was 0.9, the learning rate α was 0.015, and the temperature T of the policy softmax was decreased by 0.996 each episode starting with 1.0. Fig. 5 shows the evaluated optimal state value function of each abstract state for $\lambda = 0$, and $\lambda = 0.5$. The total running time for all 5000 episodes was less than 2 minutes on an 1 GHz Linux machine estimated using GNU Prolog's build-in predicate `user_time(Time)`.

It shows clearly that when $\lambda = 0.5$ the agent learns faster than when $\lambda = 0$ in the beginning, and both converge. The reason is that the non-zero reward is sparse, so bigger λ can influent more triples $(s, \mathring{a}, \theta)$ when a non-zero reward is received. It shows that using an eligibility trace parameter is a feasible method for solving such kind of tasks. Episode ends when the goal abstract state $\langle s_* \rangle$ is reached, so $V(s_*)$ is always 0. Abstract state $\langle s_1 \rangle$ represents a blocks world without blocks, so



(a) $\lambda = 0$



(b) $\lambda = 0.5$

Figure 5. Experiment results of $\Theta(\lambda)$-learning with (a) $\lambda = 0$ and (b) $\lambda = 0.5$.

$V(s_1)$ is always 0, too. Abstract state $\langle s_3 \rangle$ represents all blocks on the floor. For the stack task, it is seldom visited and all applicable abstract actions are equivalent, so $\langle s_3 \rangle$ converges slowly. The learnt abstract actions for $\langle s_2 \rangle, \langle s_4 \rangle, \langle s_5 \rangle$ are all $\mathtt{move(C, D, A)}$.

B. State Refinement

In this experiment, we test the state refinement process and discuss the final state space with that used in last experiment. The only prototype action is $\langle \mathring{a}_* \rangle$, and the initial abstract states of stack task are $\langle s_* \rangle, \langle s_1 \rangle$ and $\langle s_2' \rangle$ mentioned in section III, where $\langle s_* \rangle$ is the goal state, and the others are generated from it using the generating method. The agent gets reward 10 when the goal state is reached and the reward is used to adjust Θ values of preceding states.

The parameters were set as follows. The discount factor γ was 0.9. The learning rate α was 0.015. The temperature T of the policy softmax was decreased by 0.996 each episode starting with 1.0. The eligibility trace decay parameter λ was 0.5. The threshold of splitting state ρ was 200 and the increasing rate ω was 2.

Fig. 6 gives the learning curves for 1000 episodes. The total running time for all 1000 episodes was about

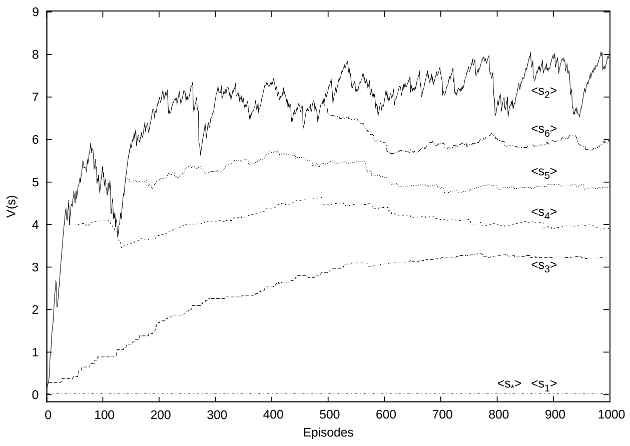


Figure 6. Experiment results of $\Theta(\lambda)$-learning with state refinement.

2 minutes on an 1.5GHz Linux machine. Episode ends when the goal abstract state $\langle s_* \rangle$ reached, so $V(s_*)$ has no chance changed and is always 0. Abstract state $\langle s_1 \rangle$ represents a blocks world containing 0 blocks, so $V(s_1)$ is always 0, too.

In the learning process, abstract states $\langle s_3 \rangle, \langle s_4 \rangle, \langle s_5 \rangle$ and $\langle s_6 \rangle$ are splitted from abstract state $\langle s_2' \rangle$ about episode 3, 40, 140, and 490, respectively. The expanding conjunctions are $\{\mathtt{on(D, E)}\}$, $\{\mathtt{on(E, F)}\}$, $\{\mathtt{on(F, G)}\}$, and $\{\mathtt{on(G, H)}\}$. Obviously, $\langle s_2' \rangle$ will be changed in each splitting step. The final abstract states are:

$$\begin{aligned}
\langle s_2 \rangle = \{ & \{\mathtt{on(A, B), clear(A), on(C, D), clear(C), C \neq A,} \\
& \mathtt{on(D, E), on(E, F), on(F, G), on(G, H)}\}, \varnothing\} \\
\langle s_3 \rangle = \{ & \{\mathtt{on(A, B), clear(A), on(C, D), clear(C), C \neq A}\}, \\
& \mathtt{not}\{\mathtt{on(D, E)}\}\} \\
\langle s_4 \rangle = \{ & \{\mathtt{on(A, B), clear(A), on(C, D), clear(C), C \neq A,} \\
& \mathtt{on(D, E)}\}, \mathtt{not}\{\mathtt{on(E, F)}\}\} \\
\langle s_5 \rangle = \{ & \{\mathtt{on(A, B), clear(A), on(C, D), clear(C), C \neq A,} \\
& \mathtt{on(D, E), on(E, F)}\}, \mathtt{not}\{\mathtt{on(F, G)}\}\} \\
\langle s_6 \rangle = \{ & \{\mathtt{on(A, B), clear(A), on(C, D), clear(C), C \neq A,} \\
& \mathtt{on(D, E), on(E, F), on(F, G)}\}, \mathtt{not}\{\mathtt{on(G, H)}\}\}
\end{aligned}$$

The final learnt states are very different from that given by human users in [24] described in section III. We find that the agent catches the core of stack task in the learning process. The target states to split and the expanding conjunctions reflect the status of the highest stack in the blocks world although the predicate `height` is not used. Apparently, the height of the stack is a more notable feature of the task rather than the number of the stacks in the blocks world, that is, moving all other blocks to the highest stack is enough for the task. So, it is foreseeable and understandable that all the learnt policies for every abstract states are $\mathtt{move(A, B, C)}$.

Another interesting thing is that the abstract states $\langle s_3 \rangle$–$\langle s_6 \rangle$ converge quickly but $\langle s_2 \rangle$ shakes although it trends to convergence. It implies that the shaky state should be splitted because it represents different kinds of ground states, some of which reaches the goal state quickly and some slowly.

VII. Related Work

Research on using rich relational representations for reinforcement learning has been a heat research topic during the past few years. Van Otterlo [15], [22] gave the definition of the RMDP in the ground level and defined the *CARCASS* which is an RL-based abstraction for RMDPs, and several abstract models have been proposed. Kersting and De Raedt [12] defined the *Logical MDP* (LOMDP) which is an abstraction over RMDPs by using variables in states and actions, and use Q-learning on the fixed abstract model. Kersting and Van Otterlo [14] introduced ReBel which uses a constraint logic language to regress preceding abstract states automated based on LOMDP for model-based learning. Wang et al. [19] developed a representation for decision diagrams suitable for describing value functions, transition probabilities, and domain dynamic of FOMDPs. Morales [13] introduced an *a priori* abstraction of RMDPs based on separate state and action spaces and uses Q-learning over the abstraction. Guestrin et al. [11] used probabilistic relational models and modeled class-based value functions assuming fixed relations.

Apart from the community of reinforcement learning, there has also been work on compact and elaboration tolerant representation of MDPs. Bacchus et al. [26] and Mateus et al. [27] propose probabilistic generalizations of the situation calculus. Boutilier et al. [28] suggest 2 stage temporal Bayesian networks (2TBNs) that augment Bayesian networks to represent stochastic effects of actions. Pooles independent choice logic [29] allows for independent choices and an acyclic logic program that gives the consequence of choices. Boutilier et al. [30] introduce a dynamic programming approach to first-order MDPs that are formulated in a probabilistic generalization of the situation calculus, and in [31] present a generalization of Golog, called DTGolog, that combines agent programming with decision theoretic planning. Probabilistic extensions of the action language A [32] has been proposed by Baral et al. [33].

Numerous work has contributed for the agent to grasp the environment based on its experience. For attribute-value representations in RL, there exists an incremental regression tree induction algorithm that was designed for Q-learning, i.e., the G-algorithm by Chapman and Kaelbling [34]. In the beginning, It tries to learn Q values for the entire environment as if it were one state. In the learning process, it gathers statistics based on individual input bits, and finds which bit cause a significant different Q value. The bit is used to split the decision tree. Then, the process is repeated in each of the leaves. This method was able to learn very small representations of the Q function. The U-Tree algorithm of McCallum [35] rely on an attribute value representation of states and actions, but allow the use of a (limited term) history of a state as part of the decision criteria. This history and the feature selection of U-trees allow generalization over similar states in the decision tree. Moore's PartiGame algorithm [36] is another solution to the problem of partitioning the deterministic high-dimensional continuous spaces. It divides the environment into cells, and in each cell, the actions available consist of aiming at the neighboring cells. The graph of cell transitions is solved for shortest paths in an online incremental manner, but a minimax criterion is used to detect when a group of cells is too coarse to prevent movement between obstacles or to avoid limit cycles. The offending cells are split to higher resolution.

VIII. Conclusions

In nLMDPs, the logical negation is introduced under the closed world assumption. So a complementary abstract state space can be constructed in an easy way by applying the generating method once and the expanding method several times in turn until its size is suitable for learning. Based on the complementary abstract state space and the definition of prototype action, the nLMDP is proposed, and a model-free Θ-learning method is implemented for evaluating the state-action-substitution value function. Experiments perform well and show the convergence results.

Based on the nLMDP, an refinement algorithm for states is proposed in this paper. A larger self-loop degree reflects the successive state is more likely to be itself, and it has more chance to be visited by agents. To keep the abstract states well-distributed, the state which has maximal self-loop degree is selected as the target state to be splitted. From the ground states represented by the target state, the characteristics of them can be computed. A set of the ground states can be separated from others based on the characteristics in common, and such characteristic is used as expanding conjunction to split the target state.

Guided by the above analysis, a model-free learning algorithm is proposed. The self-loop count and the common characteristic obtained in the trial-and-error interactions are used for the state refinement. In the experiments of the stack task of blocks world, although only the goal state is given by human users, the agent can catch the core of the task, i.e. the status of the highest stack, in the learning process, furthermore the agent can split the shaky states, leading to the intuitive state space. In a word, we propose a framework enabling the agent to grasp the environment while it is learning the policy. We expect that this framework will be useful to increase the intelligence of agents and decrease manual work in some domains, especially in the domain with many objects.

Future work includes more complicated applications and applying $\Theta(\lambda)$-learning and state refinement to multi-agent domains.

References

[1] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[2] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[3] P.-Y. Yin, B. Bhanu, K.-C. Chang, and A. Dong, "Integrating relevance feedback techniques for image retrieval using reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1536–1551, 2005.

[4] S. Fakih and T. Das, "LEAD: a methodology for learning efficient approaches to medical diagnosis," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 2, pp. 220–228, 2006.

[5] S. Misra and B. Oommen, "An efficient dynamic algorithm for maintaining all-pairs shortest paths in stochastic networks," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 686–702, 2006.

[6] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," *Machine Learning*, vol. 43, pp. 7–52, 2001.

[7] M. Van Otterlo, "A survey of reinforcement learning in relational domains," CTIT Technical Report Series ISSN 1381-3625, July 2005.

[8] P. Tadepalli, R. Givan, and K. Driessens, "Relational reinforcement learning: An overview," in *ICML'04 Workshop on Relational Reinforcement Learning*, 2004.

[9] S. Yoon, A. Fern, and R. Givan, "Inductive policy selection for first-order MDPs," in *UAI'02*, 2002.

[10] J. Cole, K. Lloyd, and K. Ng, "Symbolic learning for adaptive agents," in *The Annual Partner Conference, Smart Internet Technology Cooperative Research Centre*, 2003.

[11] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia, "Generalizing plans to new environments in relational MDPs," in *IJCAI'03*, 2003.

[12] K. Kersting and L. De Raedt, "Logical markov decision programs," in *IJCAI'03 Workshop on Learning Statistical Models of Relational Data*, 2003.

[13] E. Morales, "Scaling up reinforcement learning with a relational representation," in *Proceedings of the Workshop on Adaptability in Multi-agent Systems at AORC'03, Sydney*, 2003.

[14] K. Kersting, M. Van Otterlo, and L. De Raedt, "Bellman goes relational," in *ICML'04*, 2004.

[15] M. Van Otterlo, "Reinforcement learning for relational MDPs," in *Machine Learning Conference of Belgium and the Netherlands*, 2004.

[16] K. Driessens and S. Džeroski, "Combining model-based and instance-based learning for first order regression," in *Proceedings of the 22nd International Conference on Machine Learning*, 2005, pp. 193–200.

[17] J. Ramon, "Convergence of reinforcement learning using a decision tree learner," in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn Germany, 2005.

[18] A. Karwath and K. Kersting, "Relational sequence alignements," in *Proceedings of The 4th International Workshop on Mining and Learning with Graphs (MLG '06)*, T. Gärtner, G. C. Garriga, and T. Meinl, Eds., September 2006.

[19] C. Wang, S. Joshi, and R. Khardon, "First order decision diagrams for relational mdps," in *IJCAI 2007*, 2007.

[20] S.-H. Neinhuys-Cheng and R. de Wolf, *Foundations of Inductive Logic Programming, vol. 1228 of Lecture Notes in Artifical Intelligence*. Springer-Verlag, 1997.

[21] K. Clark, "Negation as failure." in *Logic and Data Bases*, 1977, pp. 293–322.

[22] M. Van Otterlo and K. Kersting, "Challenges for relational reinforcement learning," in *ICML'04 Workshop on Relational Reinforcement Learning*, 2004.

[23] N. Kushmerick, S. Hanks, and D. Weld, "An algorithm for probabilistic planning," *Artificial Intelligence*, vol. 76, pp. 239–286, 1995.

[24] K. Kersting and L. De Raedt, "Logical markov decision programs and the convergence of logical TD(λ)," in *Fourteenth International Conference on Inductive Logic Programming*, 2004, pp. 180–197.

[25] J. Slaney and S. Thiébaux, "Blocks world revisited," *Artificial Intelligence*, vol. 125, pp. 119–153, 2001.

[26] F. Bacchus, J. Y. Halpern, and H. J. Levesque, "Reasoning about noisy sensors and effectors in the situation calculus," *Artificial Intelligence*, vol. 111, no. 1-2, pp. 171–208, 1999.

[27] P. Mateus, A. Pacheco, and J. Pinto, "Observations and the probabilistic situation calculus," in *Proceedings of the 8th KR*, 2002, pp. 327–340.

[28] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research (JAIR)*, vol. 11, pp. 1–94, 1999.

[29] D. Poole, "The independent choice logic for modelling multiple agents under uncertainty," *Artificial Intelligence*, vol. 94, no. 1-2, pp. 7–56, 1997.

[30] C. Boutilier, R. Reiter, and B. Price, "Symbolic dynamic programming for first-order MDPs," in *Proceedings of the 17th IJCAI*, 2001, pp. 690–700.

[31] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun, "Decisino-theoretic, high-level agent programming in the situatino calculus," in *Proceedings of the 17th AAAI/12th IAAI*, 2000, pp. 355–362.

[32] M. Gelfoun and V. Lifschitz, "Representing action and change by logic programs," *Journal of Logical Programming*, vol. 17, no. 2-4, pp. 301–321, 1993.

[33] C. Baral, N. Tran, and L.-C. Tuan, "Reasoning about actions in a probabilistic setting," in *Proceedings of the 18th AAAI/14th IAAI*, 2002, pp. 507–512.

[34] D. Chapman and L. Kaelbling, "Input generalization in delayed reinforcement learning: An algorithm and performance comparisions," in *Proccedings of the 12th International Joint Conference on Artificial Intelligence*, 1991, pp. 726–731.

[35] A. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Computer Science Department, University of Rochester, 1995.

[36] A. Moore, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Machine Learning*, vol. 21, no. 3, pp. 199–233, 1995.

Song Zhiwei is currently a Postdoc at University of Science and Technology of China (USTC). He received his Ph.D. and BS degrees in computer science from USTC in 2006 and 2001, respectively. His research interests include agent learning, multi-agent systems, and natural computing.

Chen Xiaoping is currently a Professor of Computer Science at USTC. He received his Ph.D. degree in computer science in 1997 from USTC. His current research interests include AI logic, agent learning, and multi-robot systems.

Cong Shuang is currently a Professor in the Department of Automation at USTC. She received her Ph.D. degree in system engineering from the University of Rome "La Sapienza", Rome, Italy, in 1995. Her research interests include advanced control strategies for motion control, fuzzy logic control, neural networks design and applications, robotic coordination control, and quantum systems control.