Dynamic Epistemic Learning of Actions

Shuai Wang

ILLC, University of Amsterdam, The Netherlands
Email: shuai.wang@student.uva.nl

Abstract. This paper gives a first practical attempt on epistemic learning of actions. More specifically, this paper presents the learning of two types of actions, namely precondition-free actions and conditional actions. The corresponding two learning algorithms are designed and implemented with the use of modern logic reasoners. The evaluations indicate that these action learning algorithms are efficient and scalable.

Keywords: action learning, epistemic learning, scalable learning algorithms

1 Introduction

Action learning has been an important topic in robotics [8], Multi-Agent Systems (MAS) [4] and planning [3]. Statistical methods, such as Machine Learning (especially Reinforcement Learning) and Markov Decision Processes, have been playing key roles in action learning [3, 5]. However, until recently, little has been studied on epistemic learning of actions. This paper follows from the theoretical proposals in [2] and presents a first practical attempt of action learning in a formal approach [1]. More specifically, two learning algorithms corresponding to two types of actions are to be presented and evaluated. An action is a set of pairs of *precondition* and *postcondition.* A *context-free action* is an action with one precondition and one postcondition. The precondition limits the worlds the action can be performed while the postcondition represents the effect of the action. In contrast to context-free actions, a *context-sensitive* action may have several preconditions and their corresponding post conditions. A *before-world* is a formal representation of the world before the action was performed. A before-world is *valid* if it satisfies the precondition of the action. An *after-world* is a representation of the world following the effect of the action[1]. The set of the variables in the world is referred to as *domain.* An *event* consists of a before-world, actions and an after-world. If an event has several actions, it is assumed that they happened at the same time in parallel. This paper consider only one action at a time. An *observation* is a sequence of events. An (epistemic) *learning*

[1] This project is restricted to only the learning of one action at a time, the action name, the duration of the action, and even the agent the action belongs to will not be discussed in this paper. Moreover, it is always the case that once an action is performed, the after-world always satisfies the postcondition of the action.

function takes an observation of events and outputs an action. Two actions are equivalent if their precondition and the postcondition are the same.

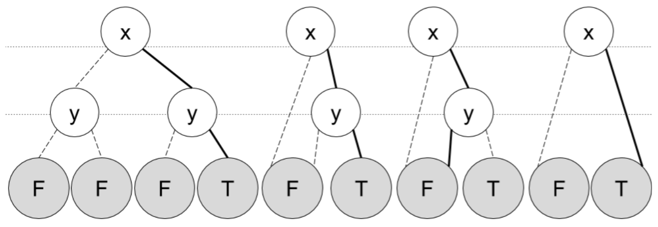
This paper presents the epistemic learning of two context-free actions with the help of modern reasoners. Some background of propositional logic and epistemic logic will be given in Section 2. The algorithms are to be presented in Section 3 followed by the implementation and evaluation in Section 4. This paper will be brought to an end by some discussions and some future work.

2 Background

2.1 Propositional Logic, Binary Decision Diagrams and Answer Set Programming

Propositional Logic is the logic with *propositional variable* interpreted to *true* ($\top$) or *false* ($\bot$). A *literal* is defined as a propositional variable or its negation (complement). A *disjunctive clause* is a disjunction of literals. A formula is in *Conjunctive Normal Form* (CNF) if it is a conjunction of disjunctive clauses. A propositional formula is interpreted to true or false. If we can assign each variable to true or false, i.e. find an *interpretation*, making all the clauses evaluates to true, we say the formula is *satisfiable* and the assignment as a *model* of it. These problem are named *Satisfiabllity Problems* (SAT problems). A formula is a *tautology* if it evaluates to true under all interpretations. The process of deciding if a formula is a tautology may be the process of constructing a *Binary Decision Tree* (BDT). A BDT is a binary tree with propositional variables as nodes and edges representing true or false (as the truth value of the variable of the node). The leaves of the tree are the results of the evaluation (with the interpretation obtained by visiting the nodes and edges from the root). A *Binary Decision Diagram* (BDD) is a more compact representation. BDDs are manipulated under the operators isomorphic to the operators of propositional logic ($\wedge$, $\vee$, $\neg$). Fig 1 shows the BDT of $x \wedge y$, the BDD of $x \wedge y$ and $x \wedge \neg y$ as well as their disjunction (which is equivalent to x).

Fig. 1. The BDT of $x \wedge y$, the BDD of $x \wedge y$ and $x \wedge \neg y$ as well as their disjunction



Remind that the problem of deciding if there is an interpretation making a formula evaluates to true is named a SAT problem. The program which works this out is called a *SATsolver*. Such interpretations are called the *models* of this formula. In similar fashion, an *answer set solver* computes the *stable models* (i.e. an answer set). An answer set problem can be reduced to a SAT problem by repeatedly adding clauses (with information about the current model) and get the new models. An answer set solver can be used as a SAT solver. *Answer Set Programming* (ASP) is a declarative way of using an answer set solver. An answer set solver is also known as an ASP solver.

2.2 Dynamic Epistemic Logic and Epistemic Action Learning

Propositional *Epistemic Logic* extends propositional logic by the use of the modal operator K. It gives a formal representation of the knowledge of an agent or an individual. $K\varphi$ is read as "it is known that φ (is true)" and $K_a\varphi$ is read as "agent a knows that φ (is true)". Different from Epistemic Logic where agents' knowledge does not change, *Dynamic Epistemic Logic* (DEL) studies the change of knowledge [9,10] where agents can make public or group announcements. The semantics of DEL is defined on *Kripke models*. An Kripke model $M = (W, E, V)$ over a set of propositions P is a representation of the worlds W, edges between the worlds E and the evaluation function V. If there is an edge between two worlds w_1 and w_2, the agent is considered unable to distinguish these worlds. The evaluation function V maps worlds to a set of propositions that are true in the world. An *update* of the agents' knowledge follows from the announcements. That is, edges or worlds are eliminated after certain updates. Announcements as updates and some corresponding model checking problems has been studied in [9]. Here we study how observations of events would change the agent's knowledge. For an agent and two propositions p and q. The Kripke model is $M = (W, E, V)$ where $W = \{w_1, w_2, w_3, w_4\}$ with $V(w_1) = \{p, q\}$, $V(w_2) = \{p\}$, $V(w_3) = \{q\}$ and $V(w_4) = \emptyset$. Initially the agent can not distinguish these worlds. That is to say, there are edges between any two the worlds (including the world itself). After a public announcement of $\neg p$, we eliminate all the worlds where p is the case, i.e. w_1 and w_2. Now that there are only worlds where p is true, we denote Kp. An Epistemic Learning problem is to remove all such uncertainty and keep only one world in the model.

Theorem 1. *A problem of Epistemic Learning with propositional updating information can be reduced to a SAT problem.*

It is easy to prove by regarding each world as a model of a SAT problem. Propositional updating information is taken as new clauses. The updated epistemic model (Kripke model) is with the worlds corresponding to the models. With this in mind, we look into action learning.

In this paper we discuss only non-trivial context-free actions (actions with exactly one precondition and one non-trivial postcondition). Let us take the set of all propositional variables $P = \{p_1, p_2, \ldots, p_n\}$ is the domain. For convenience, we define $-P = \{\neg p_1, \neg p_2, \ldots, \neg p_n\}$. Both conditions are taken as a consistent set of literals over P. The set of all consistent precondition over a domain P is the consistent subset of the power set of all literals $PRE \subsetneq \mathcal{P}(P \cup -P)$. While the set of all non-trivial consistent postcondition is $POST = PRE \setminus \{\emptyset\}$.Let us take the initial Kripke model is $M = (W, E, V)$. For every pair $\alpha = (i, j)$, s.t $i \in PRE$ and $j \in POST$, there is a world $w_\alpha \in W$ with $V(w_\alpha) = (i, j)$. Moreover, the agent cannot tell the difference between actions: $E = \{(w, w')|w, w' \in W\}$. An **Epistemic Action Learning** problem, is to update the model with observation of events until $|W| = 1$. The only world left represents the action learnt. This problem can also be considered a formal language learning problem with events as words. This paper is restricted to observations for one action at a time (and there is only one agent in the environment). An event is a structure of a before-world, an action and an after-world. A sequence of events is named an observation. This paper concentrates on the updates of the knowledge of actions, in other words, the learning of actions by observing a sequence of the performance of actions (i.e. events). More specifically, we propose two learning algorithms, corresponding to *precondition-free actions* and *conditional actions* respectively. Precondition-free actions are simply the actions with $\emptyset$ as their precondition. Actions with non-empty preconditions are harder to learn. In this paper we limit to only the case of *conditional actions.* That is, actions satisfy the following constraints:

1. The intersection of the precondition and the postcondition is empty.
2. The post-condition contains at least the negation of all the propositions in pre-condition.

Condition 1 says we do not consider actions like $a = (\{p_1, p_2\}, \{p_1, \neg p_2\})$. The following lemma shows that there is no learning function determine this action after observing a finite amount of events. In other words, the learning function would not be able to distinguish $a' = (\{p_1, p_2\}, \{\neg p_2\})$ with a. It is easy to see that any (valid) before-world w will always result in the same after-world after the action of a and a' since p_1 will stay the same as in the before-world[2]. Condition 2 following from Condition 1 to further specify that the action must contain the negation of all the literals in the precondition.

Lemma 1. *If the intersection of the precondition and the postcondition is not empty, the postcondition of the action is not deterministic (learnt) after an observation of any number of events.*

The output action a' is *safe* if the postcondition is exactly the same as the actual action and the precondition is stricter than the actual action a. By stricter, we mean the implication of the precondition of a' to the actual precondition is always true. That is, intuitively, we assume it impossible to perform the action if the agent has never observed such a world where the action was performed. An example is: $\{p, q\}$ is the actual precondition and $\{p, q, r\}$ is the precondition of a safe action learnt.

[2] This distinction might look trivial at a first glance, however, in situations allowing parallel actions (i.e. multiple actions in an event). This distinction would make a difference.

Lemma 2. *For any conditional action, it is possible to learn a safe action of it after a finite observation.*

Following this, we can easily get the following corollary:

Corollary 1. *For an conditional action with a precondition of the same size as the domain, it costs an observation of only one (valid) event to be learnt.*

3 Learning Algorithms for Context-free Actions

This section presents an encoding of the learning of a precondition-free action as a SAT/ASP problem[3] followed by the extension of this approach using BDDs for conditional action learning.

3.1 Precondition-free Action Learning

A precondition-free action is with $\emptyset$ as its precondition and a set of literals in the domain P as its postcondition. As for **initialisation**, for each propositional variable $p_i \in P$, we introduce three propositions p_i^+, p_i^- and $p_i^\times$ representing if action would change the proposition to true, false, keep its value in the after-world. Knowing that one and only one of these three propositional variable is true, these three variables are encoded as $p_i^+ \oplus p_i^- \oplus p_i^\times$. Next we **encode an event** regarding an action act, $e = (w_b, act, w_a)$. These literals are encoded as follows [4]:

- For each $p_i \in w_b \cap w_a$, we encode as $\neg p_i^-$
- For each $\neg p_i \in w_b \cap w_a$, we encode as $\neg p_i^+$
- For each $p_i \in w_b$ and $\neg p_i \in w_a$, we encode as p_i^-
- For each $\neg p_i \in w_b$ and $p_i \in w_a$, we encode as p_i^+

The process of **decoding** is obvious: deciding if a variable p_i is true (or false) in a world is equivalent to checking if p_i^+ (or p_i^-) is assigned to true in the model. The encoding of events is performed after observation of each event. The size of the model should reduce as more events are observed until there is one and only one model which represents the postcondition after decoding. The action is therefore learnt as a consequence. An ASP solver-based algorithm is given as Algorithm 1. In fact, with little change of the algorithm, we can obtain a faster algorithm with a SAT solver (as in Algorithm 2).

Take the conjunction of all the propositions $p_1, p_2 \ldots, p_n$ assigned to true in m as l. The model m is encoded as $\neg(p_1 \wedge p_2 \wedge \ldots \wedge p_n)$, which means there can not be an exact same model in later iterations.

With this algorithms, we get the following theorem:

[3] Since eventually we would like to replace the ASP solver by a parallel SAT solver, the encoding is more SAT-like.

[4] The encoding can be further optimised by comparing current event to be encoded with all past events. For the efficiency of encoding, this is not attempted in this paper.

```
Data: The domain $P$ and the action to be learnt $a$
Result: An learnt action $a'$ and the number of events observed $i$
initialise an ASP solver $s$;
initialise an event generator $eg$ of $a$;
initialise $P$ in $s$;
initialise an integer $i = 0$;
while the size of answer set obtained from s is more than 1 do
    generate one more event $e$ from $eg$;
    encode $e$ in s;
    i ++;
end
if the size of answer set is 1 then
    decode the model and construct the action learnt $a'$;
    return the action $a'$ and $i$;
else
    error;
end
```

Algorithm 1: Precondition-free action learning with an ASP solver

```
Data: The domain $P$ and the action to be learnt $a$
Result: An learnt action $a'$ and the number of events observed $i$
initialise a SAT solver $s$;
initialise an event generator $eg$ of $a$;
initialise $P$ in $s$;
initialise an integer $i = 0$;
initialise an empty model $m$;
while the SAT solver returns sat do
    record the model in $m$;
    encode the $m$;
    generate one more event $e$ from $eg$;
    encode $e$;
    i ++;
end
decode the (last) model $m$ and construct the action learnt $a'$;
return the action $a'$ and $i$;
```

Algorithm 2: Precondition-free action learning with a SAT solver

Theorem 2. *A problem of (propositional) Precondition-free Action Learning with propositional updating information can be reduced to a SAT problem.*

3.2 Conditional Action Learning

For conditional action learning, the algorithm of precondition-free action learning is extended by employing a BDD. More specifically, the BDD is initialised as $\perp$ and records all the valid worlds the action can perform (as in Algorithm 3).

```
Data: The domain $P$ and the action to be learnt $a$
Result: An learnt action $a'$ and the number of events observed $i$
initialise a SAT solver $s$;
initialise $P$ in $s$;
initialise an event generator $eg$ of $a$;
initialise an integer $i = 0$;
initialise an empty model $m$;
initialise a BDD $b$ as $\bot$;
while the SAT solver returns sat do
    record the model in $m$;
    encode $m$ in $s$;
    generate one more event $e$ from $eg$;
    encode $e$;
    $b := b \vee w_b$ ;
    i ++;
end
decode the (last) model $m$ and construct the action learnt $a'$;
return the action $a'$ and $i$;
```

Algorithm 3: Conditional action learning with a SAT solver

Since the algorithm terminates as soon as the postcondition is determined and the learning of a postcondition always terminates (after a finiite observation). We can prove that the algorithm always terminates:

Corollary 2. *The learning of a (propositional) Conditional Action with propositional updating information and a safe action as result always terminates.*

4 Implementation and Evaluation

The project was implemented in C++. The source code is available online at `https://github.com/airobert/PAAMS16`. In the implementation, the C++ API of Smodels [7] was used and events were randomly generated[5]. The encoding method is SAT-like to take full advantage of the performance of a SAT solver instead of an ASP solver (at a later stage). Both using Smodels as an ASP solver and as a SAT solver have been attempted at early stage of the project. As expected, generating the answer set explicitly and get the size of it is less efficient compared with using Smodels solely as a SAT solver. Therefore, we keep the latter in the final version. In this project, BuDDy was chosen as the BDD package due to its usability and efficiency [6]. The number nodes of BDDs were initialised to the size of the domain (BuDDy can add nodes by itself as it reasons) while the set variable number was initialised to $(\#domain)^2/20$ to avoid running out of variables.

[5] For the sake of efficiency, invalid events are not generated, i.e. a learning agent ignores these invalid events immediately. For this reason, we take the number of valid events as the #Observation in the evaluation of conditional action learning.

As for evaluation, two aspects were analysed: the efficiency and the scalability. Instead of calculating the size of the encoded clauses, the number of nodes of BDDs as well as the total memory taken, we use the number of iterations instead, i.e. the number of events observed (#Observation) as a rough representation of the scalability regarding these aspects. All the evaluation experiments were completed on a PC with one processor of two cores of 2.3 GHz (Intel Core i5) and a memory of 4GB.

For each domain, we take a division of 10 and a repetition of 20 for each division and then compute the average for each division. Indexed by the section the size of postcondition belongs to, Table 1 indicates that the average time taken changes little as the size of postcondition (#Post) of actions increases. The result indicates that the number of events needed to determine the postcondition is more related to the domain size than the size of the postcondition. Table 2 presents the efficiency of conditional action learning in this approach. Although it takes more time in conditional action learning but the scale-up is very small. Similarly, Figure 2 and 3, shows that the method is with good scalability. With these evaluations and comparisons, this approach is proved to be efficient and scalable. The evaluation of the relation between preconditon and postcondition is more complex. Figure 4 and 5 indicate that for a fixed domain (100 propositions in this test), and a certain #Pre, both the average time taken and the size of observation decreases as #Post increases.

Table 1. Efficiency Evaluation of Precondition-free Action Learning - Average Time/#Observation

	The Index of the Section (regarding #Post)									
	No.1	No.2	No.3	No.4	No.5	No.6	No.7	No.8	No.9	No.10
100	0.030/9.7	0.025/8.6	0.025/8.4	0.025/8.9	0.026/9.3	0.025/8.7	0.025/8.5	0.023/7.9	0.024/7.2	0.028/8.0
200	0.137/10.1	0.124/10.0	0.128/9.6	0.142/9.9	0.128/9.2	0.111/9.3	0.109/8.6	0.109/8.8	0.123/9.2	0.123/8.9
300	0.294/10.1	0.290/10.6	0.308/10.2	0.303/10.1	0.282/10.2	0.266/9.4	0.260/9.7	0.256/9.5	0.299/9.6	0.291/9.9
400	0.513/10.0	0.517/10.9	0.530/10.6	0.488/10.2	0.503/10.7	0.469/10.2	0.452/10.1	0.469/10.2	0.477/10.6	0.453/10.1
500	0.830/11.2	0.887/11.3	0.887/11.4	0.827/10.8	0.881/11.4	0.870/10.8	0.794/10.4	0.825/10.5	0.838/10.7	0.773/9.8
600	1.318/11.2	1.598/12.4	1.344/11.2	1.363/11.4	1.261/10.9	1.256/10.7	1.218/10.3	1.174/10.8	1.136/10.6	1.104/10.4
700	2.049/12.4	1.813/11.4	1.662/11.0	1.741/11.5	1.740/11.9	1.609/10.9	1.680/11.3	1.605/10.9	1.520/11.1	1.454/10.8
800	2.321/12.4	2.213/11.8	2.544/12.9	2.469/11.9	2.401/11.6	2.266/11.9	2.026/10.8	2.261/11.6	2.170/11.1	2.159/11.1
900	3.295/12.3	3.299/12.1	3.394/11.6	3.180/11.8	2.958/11.2	3.234/12.1	2.948/11.2	2.960/11.1	2.864/11.9	2.933/11.5
1000	4.103/12.2	4.106/12.4	3.819/11.9	3.625/11.9	3.993/12.5	3.964/11.8	4.334/12.7	3.658/11.3	3.151/11.0	3.361/11.9

5 Discussion, Future Work and Conclusion

As shown in Table 1, in fact, the actions with bigger postconditions are slightly easier to be learnt. A possible reason is that the changes are more significant making the difference between before-world and after-world more obvious. So far all the actions are context-free. However, in most real-life situations, actions are context-sensitive. An example would be to flip over a coin. There, we have two

Table 2. Efficiency Evaluation of Conditional Action Learning - Average Time/#Observation

	The Index of the Section (regarding #Post)									
	No.1	No.2	No.3	No.4	No.5	No.6	No.7	No.8	No.9	No.10
100	0.110/8.7	0.078/7.8	0.053/9.0	0.065/8.4	0.044/8.4	0.041/7.8	0.051/8.4	0.042/8.2	0.040/7.8	0.033/6.5
200	0.246/11.1	0.224/10.1	0.201/9.2	0.220/9.9	0.201/9.3	0.194/9.0	0.189/8.9	0.176/8.0	0.172/8.2	0.183/8.3
300	0.507/10.1	0.515/10.4	0.507/10.3	0.499/10.1	0.478/9.7	0.495/10.1	0.463/9.4	0.472/9.7	0.451/9.6	0.427/8.8
400	0.959/10.8	0.953/10.7	0.929/10.5	0.944/10.8	0.949/10.8	0.843/9.7	0.833/9.5	0.808/9.4	0.798/9.2	0.733/8.7
500	1.572/11.2	1.498/10.8	1.532/11.2	1.499/10.9	1.516/11.2	1.386/10.2	1.387/10.2	1.457/10.8	1.224/9.2	1.338/9.9
600	2.311/11.5	2.282/11.4	2.245/11.3	2.037/10.3	2.175/11.1	2.034/10.4	2.232/11.3	2.154/11.1	2.055/10.4	1.876/9.7
700	3.324/12.2	3.089/11.3	2.968/10.8	3.027/11.2	3.025/11.2	3.062/11.4	2.748/10.3	2.876/10.8	2.655/10.0	2.536/9.6
800	4.183/11.6	3.996/11.2	4.100/11.5	4.043/11.4	4.057/11.6	3.794/10.8	3.774/10.8	3.642/10.3	3.631/10.4	3.615/10.3
900	5.524/12.1	5.410/11.9	5.545/12.2	5.117/11.3	5.164/11.5	4.838/10.8	4.857/10.9	5.058/11.4	4.879/11.0	4.583/10.3
1000	6.717/11.8	7.055/12.5	7.097/12.7	6.630/11.8	6.864/12.3	6.543/11.8	6.221/11.2	5.671/10.3	6.550/11.9	5.600/10.2

Fig. 2. Scalability Evaluation - Average Time

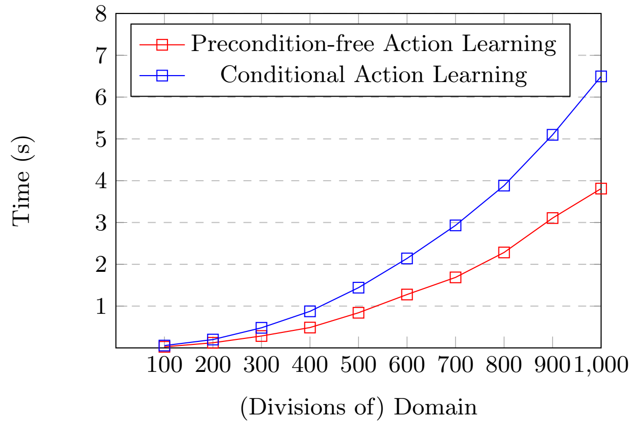
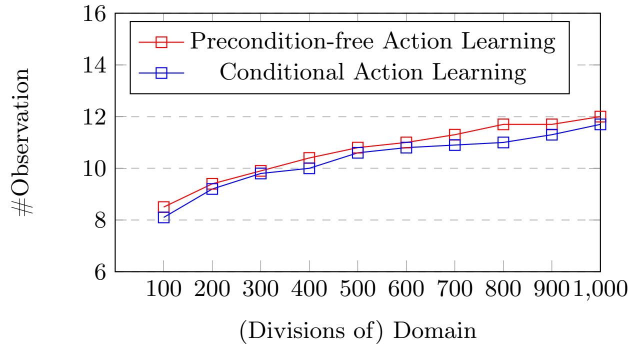


Fig. 3. Scalability Evaluation - Average #Observation



preconditions corresponding to two postconditions. The learning process requires

Fig. 4. #Pre-#Post Correspondence - Average Time

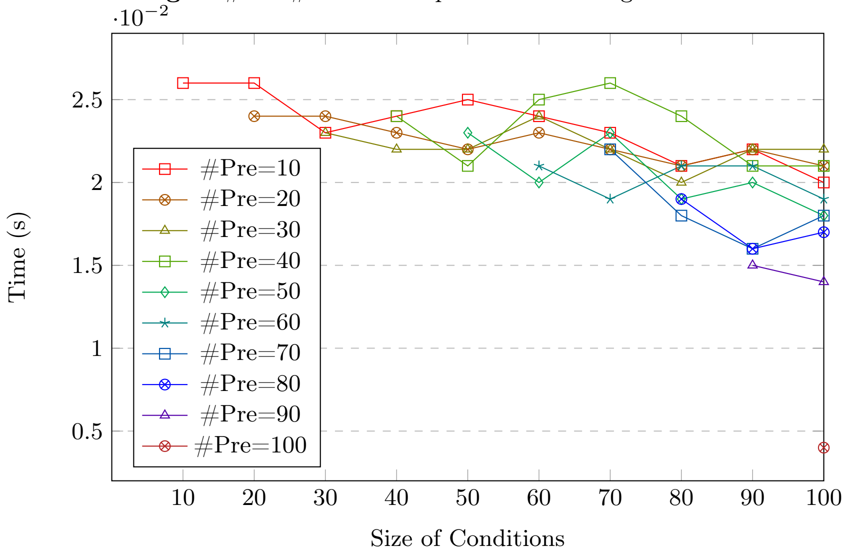
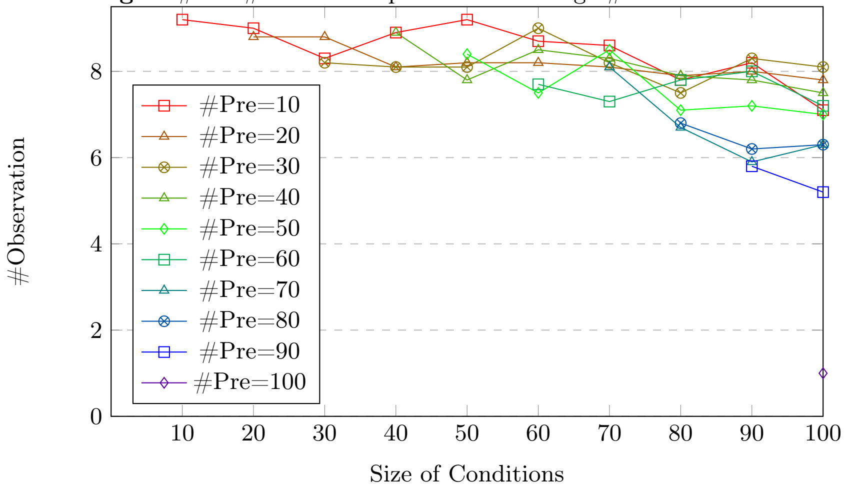


Fig. 5. #Pre-#Post Correspondence - Average #Observation



more analysis on the conditions and the events. Although such actions have been proved to be normalisable, it is hard to restrict with the lack of knowledge of the precondition [2]. Another way to deal with precondition is to record the invalid events as well. That is, a precondition is made of two BDDs: one to record the valid worlds, the other to record the invalid worlds. If a world does not get evaluated to true in the first BDD or false in the second BDD, the agent can either make a guess of the validity or wait for further information. This leads to another aspect of action learning, active learning. So far, we have shown that an agent can learn a certain action (passively) from events randomly generated. However, agents are autonomous and should be able to perform experiments by themselves. This leads to the next stage of this project, i.e. (re)active action learning. Figure 3 indicates that conditional action learning requires less events as observations. In fact, for a random environment, it would generate a significant number of invalid events. This was not captured by the current implementation due to efficiency reasons. Another work to be completed is to use a parallel SAT solver instead of an ASP solver to speed up the reasoning. In addition, it is to be studied if formal approaches are less beneficial when there are multiple actions performed in parallel. In conclusion, two algorithms of epistemic learning of actions were presented. Evaluation results confirmed the expected efficiency and the scalability of this approach. These optimistic results and the prototype implementation brings a platform for epistemic learning and planning loser to life.

References

1. Alexandru Baltag. A logic for suspicious players: Epistemic actions and belief–updates in games. *Bulletin of Economic Research*, 54(1):1–45, 2002.
2. Thomas Bolander and Nina Gierasimczuk. Learning actions models: Qualitative approach. In *Logic, Rationality, and Interaction - 5th International Workshop, LORI 2015 Taipei, Taiwan, October 28-31, 2015, Proceedings*, pages 40–52, 2015.
3. Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.
4. Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.
5. Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250. Citeseer, 1998.
6. Jørn-Lind Nielsen. Buddy-a binary decision diagram package. Technical report, Tech Report Technical University of Denmark, http://www. itu. dk/research/buddy, 2006.
7. Ilkka Niemelä and Patrik Simons. Smodels—an implementation of the stable model and well-founded semantics for normal logic programs. In *Logic Programming and Nonmonotonic Reasoning*, pages 420–429. Springer, 1997.
8. Dimitri Ognibene, Nicola Catenacci Volpi, Giovanni Pezzulo, and Gianluca Baldassare. Learning epistemic actions in model-free memory-free reinforcement learning: Experiments with a neuro-robotic model. In *Biomimetic and Biohybrid Systems*, pages 191–203. Springer, 2013.
9. Johan van Benthem, Jan van Eijck, Malvin Gattinger, and Kaile Su. Symbolic model checking for dynamic epistemic logic. In *Logic, Rationality, and Interaction - 5th International Workshop, LORI 2015 Taipei, Taiwan, October 28-31, 2015, Proceedings*, pages 366–378, 2015.
10. HP van Ditmarsch, J Ruan, and W van der Hoek. Model checking dynamic epistemics in branching time. *Formal Approaches to Multiagent Systems*, page 107.