An Automatic Software Requirement Analysis Approach Based on Intelligent Planning Technology

Hong He, Dongbo Liu
School of Computer and Communication, Hunan Institute of Engineering, Xiangtan, 411104, China
Email: hhong1970@126.com

Abstract—**With the development of information technology, the scale of current software is growing dramatically. This motivates the needs of techniques for intelligent software requirements engineering, which allows for modeling and analyzing requirements formally, rapidly and automatically, avoiding mistakes made by misunderstanding between engineers and users, and saving lots of time and manpower. In this paper, we propose an approach to acquiring requirements automatically, which adopts automated planning techniques and machine learning methods to convert software requirement into an incomplete planning domain. By this approach, we design an algorithm called Intelligent Planning based Requirement Analysis (IPRA), to learn action models with uncertain effects. Furthermore, we obtain a complete planning domain by applying this algorithm and convert it into software requirement specification.**

Index Terms—**intelligent planning, quality of service, requirement analysis, software engineering**

I. Introduction

Software requirement is an abstract concept, which is represented as software requirement specification. Requirements serve to tie the implementation world of the developers to the problem world of the stakeholder [1, 2, 21, 22]. Most empirical studies of requirements have shown that misunderstanding and changing requirements cause the majority of failures and costs in software [19, 20, 23, 24]. Since software engineers usually have limited knowledge about related field, they have to focus on analyzing obtained business process, and possibly neglect some uncertain factors. That is the reason why some software can not be applied in practice. On the other hand, it is usually difficult for users to express their demands accurately and completely without necessary hint.

Therefore, more and more attention is paid on how to acquire requirement rapidly and accurately in software requirement engineering [2, 22, 24]. For example, acquisition of software requirements based on ontology [1] is one of hot topics, which focuses on inducing users to offer system information with normal situation examples. Since those examples are collected randomly, it is difficult to make sure that a group of situation examples can cover the whole system, and induce users to offer requirement information completely and exactly, therefore this method can not be applied generally.

In this paper, we focus on applying intelligent methods to acquire software requirement specification automatically, which will make great difference in practice to avoid incomplete information and misunderstanding. In traditional planning research, we normally assume that action models with conditional effects and probabilistic effects could be built manually by experience, but in fact, it is difficult even for experts. It requires that experts not only should grasp logic of domain, but also have enough prior knowledge. Therefore, we propose an algorithm called Intelligent Planning based Requirement Analysis (IPRA) to learn action models with conditional effects and probabilistic effects and apply this algorithm to acquire software requirement automatically. Compared with previous action model learning algorithms, IPRA make the following contributions: (1) obtained action models by IPRA could have uncertain effects, including conditional effects and probabilistic effects. In practice, effects of actions are usually uncertain and conditional, with multiple possibilities; (2) state information of the planning traces could be incomplete. It is difficult to obtain complete state information in reality. IPRA can be applied with incomplete state information.

The rest of this paper is organized as follows. In section 2, we introduce related work. In section 3 and section 4, we make problem definition and present the steps of algorithm IPRA in detail. In section 5, we construct experiments in four planning domains to estimate the error rates of learned action models by IPRA, and apply IPRA algorithm to acquire software requirement specification. In section 6, we summarize this paper and discuss our future works.

II. Related Work

Automated planning systems achieve goals by producing sequences of actions from given action models that are provided as input. In 1971, Fikes and Nils designed STRIPS system [3] to introduce definitions of STRIPS operators, which made significant difference in the research of automated planning. In 1991, Soderland and Weld [4] designed the first nonlinear planning system SNLP of the world. In 1996, Kautz [5] converted planning into SAT problem, which effectively solved partial planning problem and showed new direction of

© 2014 ACADEMY PUBLISHER
doi:10.4304/jsw.9.11.2861-2867

automated planning. In 1995, Avrim and Merrick [6] designed the first graph planner system Graphplan to solve planning problem, and proposed concept of graph plan. In 1998, Malik proposed Plan Domain Definition Language (PDDL) [7], then PDDL gradually became a general standard of representing domain models and was applied broadly in international planning competitions.

In recent ten years, researchers have proposed a series of planning algorithms to solve problems with uncertainty. Planning problems with uncertainty have become one of the most important research topics in artificial intelligent field. Artificial intelligence magazine organized a special version to introduce planning problems with uncertainty. As a direction of planning problem with uncertainty, probabilistic planning problems have contracted more and more attention. In international intelligent planning competition of 2004, researchers organized the first probabilistic planning competition. Younes and Littman [8] proposed PPDDL1.0 to solve probabilistic planning problems with uncertain effects and was applied in competition.

Recently, researchers have proposed some algorithms to learn action models. According to whether state information is complete, these algorithms could be divided into two parts. Some algorithms are, to learn action models from plan races with complete state information [9-16], which means for each action, we obtain the state information before and after it happens in advance, and then learn preconditions and effects of action model by statistics and reasoning. Gil et al. [9] build EXPO system, bootstraped by an incomplete STRIPS-like domain description with the rest being filled in through experience. Oates et al. [10] use a general classification system to learn preconditions and effects of actions. Schmill et al. [11] learn action models by approximate computation in relative domains. Wang et al.[12] propose an approach to learn action model automatically by observing planning traces and refine the operators through practice in a learning-by-doing paradigm. Pasula et al. [13, 14] present how to learn stochastic action models without conditional effects. Holmes et al. [15] model synthetic items based on experience to build action models. Walsh et al. [16] propose an efficient algorithm to learn action models for describing Web services.

III. Problem Description And Definition

A STRIPS-like planning problem with conditional effects and probabilistic effects can be defined as a four-tuple $<S,s_0,s_g,O>$, where S represents a set of states, and each state is a set of propositions; s_0 represents the initial state and s_g represents the goal state which is the final state following with a series of states transition, starting with initial state; O represents a set of action models with conditional effects and probabilistic effects. In this paper, we note O as a three-tuple $<a, PRE, CPEFF>$, where a represents an action schema with action name and parameters, *PRE* represents preconditions, *CPEFF* represents conditional effects and probabilistic effects.

Normally, *CPEFF* can formally be expressed as $<(p_{i1},c_i,e_{i1})...(p_{ij},c_i,e_{ij})\ldots(p_{in},c_i,e_{in})>$, where c_i represents the ith condition composed of literal and conditions $c_i\,(1 \le i \le k)$ are mutually exclusive, the corresponding jth effect is represented by e_{ij} with probability p_{ij}, which is a conjunction of literal and $\sum_{j=1}^{n} p_{i_j} = 1, p_{ij} \ge 0$. In the case when condition c_i is empty, conditional effects are exactly equal to probabilistic effects. If preconditions of an action are satisfied in state s, then the action can be applied in state s, and its effects can be selected according to conditions and probabilities. A possible action sequence is denoted as $<a_1,a_2,\ldots,a_n>$, transferring from initial state s_0 to goal state s_g. Furthermore, we call $(s_0,a_1,s_1,a_2,\ldots,s_n,a_n,s_g)$ as a planning trace, where the middle state s_i might be null, and a_i represents action schema.

Action model learning with conditional effects and probabilistic effects can be described as follows. Given planning traces set T, propositions set P as input, algorithm IPRA outputs all the action models with conditional effects and probabilistic effects in A. We show an example of action model learning with conditional effects and probabilistic effects in Table 1, which is is chosen from the domain slippery-gripper, an indeterminate planning domain.

TABLE I
AN EXAMPLE INPUT IN IP RA

Input: Predicates *P* *(block ?b) (gripper ?g) (gripper-dry ?g) (holding-block ?b) (block-painted ?b) (gripper-clean ?g)*			
Input: Action Schemas *A* *(pickup ?b ?g) (dry ?g) (paint ?b ?g)*			
Input: Plan Traces *T*			
	Trace 1	Trace 2	Trace 3
Initial state	*(gripper G) (block B) (gripper-clean G) (gripper-dry G)*	*(gripper G) (block B) (gripper-clean)*	*(gripper G) (block B) (gripper-clean)*
Action 1	(paint B G)	(pickup B G)	(pickup B G)
Observ 1		*not(holding-block B)*	*(holding-block B)*
Action 2	(pickup B G)	(dry G)	(paint B G)
Observ 2			
Action 3		(pickup B G)	
Observ 3			
Action 4		(paint G)	
Goal state	*(gripper-clean G) (holding-block B) (block-painted B)*	*not(gripper-clean G) (holding-block B) (block-painted B)*	*(gripper-clean G) (holding-block B) (block-painted B)*

IV. Framework Of Algorithm Ipra

The motivation of our algorithm IPRA is to transform the action model learning problem into weights learning problem in MLNs, and obtain action models with conditional effects and probabilistic effects. The frameworks of algorithm IPRA is shown as following: (1) Encode each plan trace as a set of propositions; (2) Generate candidate formulas, using A and P; (3) Apply

© 2014 ACADEMY PUBLISHER

MLNs to learn weights of all the candidate formulas; (4) Choose some of candidate formulas according to given threshold, and convert weighted candidate formulas to action models with conditional effects and probabilistic effects as output. In the following subsections, we will show a detailed description of each step of the algorithm IPRA.

A. Encode Plan Traces

In the first step of algorithm IPRA, we encode all the plan traces as a set of proposition databases *DBs* with plan traces *T* as input. Firstly, we use propositions to represent each state of plan traces. For example, consider domain slippery-gripper in table 1, which includes two objects *B* and *G*. Present state s_1, describing that *B* is a block, *G* is a gripper, and *G* is clean, can be represented as *(block B s_1)* ∧ *(gripper G s_1)* ∧ *(gripper-clean G s_1)*. Secondly, we can consider an action as transition of states, then action can be encoded as the conjunction of propositions. For example, the action *(pickup B G s_1)* in table 1 can be treated as transition from the state *(block B s_1)* ∧ *(gripper G s_1)* ∧ *(gripper-clean G s_1)* to the state *(holding-block B s_2)*, then the action *(pickup B G s_1)* can be encoded as: *(block B s_1)* ∧ *(gripper G s_1)* ∧ *(gripper-clean G s_1)* ∧ *(pickup B G s_1)* ∧ *(holding-block B s_2)*.

According to the above method, we can encode each plan trace into a conjunction of grounded literals, and then convert them into a database(*DB*), where each record in a *DB* is a ground literal, and records are related as conjunction. For the sake of simplicity, we use *i* to denote the state symbol s_i. As an example, we encode the plan traces in Table 1 as database, and the results are shown in Table 2. In the paper, we make open world assumption, which means the grounded literal not shown in Table 2 is considered as unknown.

TABLE II
ENCODINGS OF PLAN TRACES AS DATABASES

DB1	DB2	DB3
(gripper G 0) *(block B 0)* *(gripper-clean G 0)* *(gripper-dry G 0)* *(paint B G 0)* *(pickup B G 1)* *(gripper-clean G 2)* *(holding-block B 2)* *(block-painted B 2)*	*(gripper G 0)* *(block B 0)* *(gripper-clean G 0)* *(pickup B G 0)* *not(holding-block B 1)* *(dry G 1)* *(pickup B G 2)* *(paint G 3)* *not(gripper-clean G 4)* *(holding-block B 4)* *(block-painted B 4)*	*(gripper G 0)* *(block B 0)* *(gripper-clean G 0)* *(pickup B G 0)* *(holding-block B 1)* *(paint B G 1)* *not(gripper-cleanG 2)* *(holding-block B 2)* *(block-painted B 2)*

B. Generate Candidate Formulas

In STRIPS model, if a predicate is a negative effect of an action, then the predicate should be a precondition of the action; and a predicate can not be both positive effect and negative effect of an action. Considering the two characteristics, we describe an action model in two parts:

(1) Preconditions. If predicate *p* is a precondition of action *a*, then *p* must be satisfied when the action *a* is executed, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i) \rightarrow p(\bar{y},i), \quad (1)$$

where $\bar{x}$, $\bar{y}$ are parameters, and *i* is the state symbol. In formula (1), since $p(\bar{y},i)$ is a necessary condition, not a sufficient condition, we choose $p(\bar{y},i)$ from candidate formulas with weights bigger than some threshold as preconditions in action model.

(2) Conditional effects. If predicate *p* is a positive effect of action *a* with condition *c*, then *p* should be added to the next state after the action *a* when condition *c* is satisfied, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i) \rightarrow p(\bar{y},i)\hat{}p(\bar{y},i+1)\hat{}c(\bar{z},i) \quad (2)$$

where $\bar{x}$, $\bar{y}$, $\bar{z}$ are parameters, and *i* is the state symbol.

If predicate *q* is a negative effect of action *a* with condition *c*, then *q* is satisfied when *a* is executing and condition *c* is satisfied, but not satisfied after action *a*, which can be described formally as:

$$\forall i,\bar{x},\bar{y},a(\bar{x},i) \rightarrow q(\bar{y},i) \wedge \neg q(\bar{y},i+1) \wedge c(\bar{z},i) \quad (3)$$

where $\bar{x}$, $\bar{y}$, $\bar{z}$ are parameters, and *i* is the state symbol.

Similarly, suppose action *a* has a positive effect *p* and *a* negative effect *q* with condition *c*, then it can be described formally as

$$\forall i,\bar{x},\bar{y},a(\bar{x},i) \rightarrow \neg p(\bar{y},i) \wedge p(\bar{y},i+1) \wedge q(\bar{y},i) \wedge \neg q(\bar{y},i+1) \wedge c(\bar{z},i) \quad (4)$$

where $\bar{x}$, $\bar{y}$, $\bar{z}$ are parameters, and *i* is the state symbol, which means that effects of an action can be described as conjunction of some atomic formulas.

Applying formula (1) and (4), we can acquire candidate formulas of preconditions and conditional effects. For example, in slippery-gripper domain, candidate formulas of preconditions and conditional effects of action *pickup* are shown in Table 3 and Table 4.

TABLE III
CANDIDATE FORMULAS OF PRECONDITIONS BY (1)

ID	Formulas
1	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\ \ g\ \ i)$
2	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (block\ \ b\ \ i)$
3	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$
4	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (holding\text{-}block\ \ b\ \ i)$

TABLE IV
CANDIDATE FORMULAS OF CONDITIONAL EFFECTS BY (4)

ID	Formulas
1	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$ $\hat{}\neg(holding\text{-}block\ \ b\ \ i)\hat{}(holding\text{-}block\ \ b\ \ i+1)$
2	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$ $\hat{}(holding\text{-}block\ \ b\ \ i)\hat{}\neg(holding\text{-}block\ \ b\ \ i+1)$
3	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow \neg(gripper\text{-}dry\ \ g\ \ i)$ $\hat{}\neg(holding\text{-}block\ \ b\ \ i)\hat{}(holding\text{-}block\ \ b\ \ i+1)$
4	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow \neg(gripper\text{-}dry\ \ g\ \ i)$ $\hat{}(holding\text{-}block\ \ b\ \ i)\hat{}\neg(holding\text{-}block\ \ b\ \ i+1)$
5	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (holding\text{-}block\ \ b\ \ i)$ $\hat{}(gripper\text{-}dry\ \ g\ \ i)\hat{}\neg(gripper\text{-}dry\ \ g\ \ i+1)$
...	...

C. Learn Weights of Candidate Formulas

According to reference [17], Markov Logic Networks *L* consists of a set of pairs (F_i,ω_i), where F_i is a formula in first-order logic and ω_i is a real number. With a

© 2014 ACADEMY PUBLISHER

finite set of constants $C=\{c_1,c_2,\cdots,c_n\}$, it defines a Markov network $M_{L,C}$ as following steps: (1) $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1, if the grounded predicate is true, and 0 otherwise; (2) $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is ω_i associated with F_i in L.

We apply Alchemy system [18] to learn weights of candidate formulas, by using weighted optimized pseudo log-likelihood. For each atomic formula, if it appears in DBs, then it corresponds to $x_i=1$, otherwise 0. As mentioned in step 2, we can obtain the candidate formulas of preconditions and effects of actions by (1), (4), then learn weights of all the candidate formulas by MLNs. For example, the weights of candidate formulas in Table 3 and 4, are shown in Table 5 and Table 6.

TABLE V
WEIGHTS OF CANDIDATE FORMULAS FOR PRECONDITIONS

ID	Weights	Formulas
1	0.3	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\ \ g\ \ i)$
2	0.5	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (block\ \ b\ \ i)$
3	-0.4	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$
4	-0.2	$\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (holding\text{-}block\ \ b\ \ i)$

TABLE VI
WEIGHTS OF CANDIDATE FORMULAS FOR EFFECTS

ID	Weights	Formulas
1	0.77	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$ $\wedge\neg(holding\text{-}block\ \ b\ \ i)\wedge(holding\text{-}block\ \ b\ \ i+1)$
2	0.12	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\text{-}dry\ \ g\ \ i)$ $\wedge(holding\text{-}block\ \ b\ \ i)\wedge\neg(holding\text{-}block\ \ b\ \ i+1)$
3	0.44	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow \neg(gripper\text{-}dry\ \ g\ \ i)$ $\wedge\neg(holding\text{-}block\ \ b\ \ i)\wedge(holding\text{-}block\ \ b\ \ i+1)$
4	0.47	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow \neg(gripper\text{-}dry\ \ g\ \ i)$ $\wedge(holding\text{-}block\ \ b\ \ i)\wedge\neg(holding\text{-}block\ \ b\ \ i+1)$
5	-0.3	$\forall i.b.(pickup\ \ b\ \ g\ \ i) \rightarrow (holding\text{-}block\ \ b\ \ i)$ $\wedge(gripper\text{-}dry\ \ g\ \ i)\wedge\neg(gripper\text{-}dry\ \ g\ \ i+1)$
...	...	...

D. Obtain Action Model

In the candidate formulas of preconditions, we choose those formulas with weights bigger than some threshold as a set and convert the set into the preconditions of action model. Similarly, we can choose some candidate formulas of conditional effects and calculate their corresponding probabilities. Finally, we can obtain action model with probabilistic conditional effects. Weight of a formula in MLNs reflects the level of truth, which means the higher weight, the more formulas with true value after instantiation. At the beginning, we need to decide a threshold of the weights. For example, we set the threshold to be 0, then we can choose all the formulas with weights bigger than 0 in Table 7, as shown below.

$$\begin{cases}\forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (gripper\ \ g\ \ i)\\ \forall i,b,g,(pickup\ \ b\ \ g\ \ i) \rightarrow (block\ \ b\ \ i)\end{cases}$$

Therefore, predicates $(gripper\ \ g\ \ i),(block\ \ b\ \ i)$ are the preconditions of action $(pickup\ \ b\ \ i)$.

Similarly, we choose those formulas under the same condition, with weights bigger than 0 in Table 6, and calculate their corresponding probabilities, then we can acquire the action model of $(pickup\ \ b\ \ i)$ with probabilistic and conditional effects as shown in Table 7.

TABLE VII
THE ACQUIRED ACTION MODEL

Action	pickup(?b ?g)
Preconditions	*block(?b), gripper(?g)*
Probabilistic conditional effects:	*<(0.87 (gripper-dry ?g) (and (holding-block ?b))), (0.13(gripper-dry?g) (and(not(holdingblock ?b)))) > <(0.48(not(gripper-dry?g))(and(holdingblock? b))), (0.52 (not (gripper-dry ?g)) (and(not(holding-block ?b))))>*

V. EXPERIMENTS EVALUATION

A. Datasets and Evaluation Criteria

To evaluate the algorithm IPRA, we collected plan traces from the following planning domains: slippery-gripper, blocks-world, zenotravel, logistics-strips. These domains have the characteristics we need to evaluate in IPRA algorithm: all the four domains have uncertain effects. Using probabilistic planner Probabilistic-FF, we generated 20-100 planning traces from the three domains, as training data of learning action models with probabilistic effects. We consider the given action models in the above web-page as correct ones, and then use the correct action models to evaluate the error rates of learned action models.

We define the error rates of our algorithm as follows:

(1) Error rates of preconditions: let the number of all the possible preconditions in action models be N_{pre} ,the set of preconditions of learned action models be T'_{pre} ,and the set of preconditions of correct action models be T''_{pre} .If a precondition belongs to T'_{pre} , not T''_{pre} , then the number of errors in preconditions denoted by E_{pre} ,adds one; similarly if a precondition belongs to T''_{pre} ,not T'_{pre} , E_{pre} adds one. Then the number of errors in preconditions can be expressed as $n_{pre}=\left|T'_{pre}\cup T''_{pre}-T'_{pre}\cap T''_{pre}\right|$ .Thus error rate of preconditions can be calculated as $P_{pre}=\frac{n_{pre}}{N_{pre}}$.

(2) Error rates of effects: since the learned action models have probabilistic effects, then we calculate the error rates of effects in a different method. Suppose for action a, the correct action model with probabilistic effects has m effects, and the corresponding probability

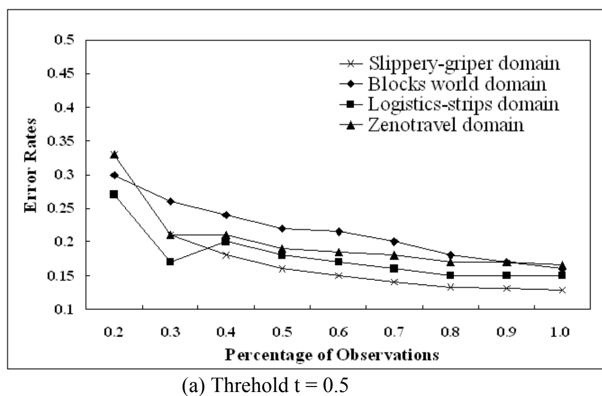
© 2014 ACADEMY PUBLISHER

is $p_i, 1 \le i \le m$,and $\sum_{i=1}^{m} p_i = 1$ .Suppose the learned action model has n effects, and the corresponding probability is $q_j, 1 \le j \le n$,and $\sum_{j=1}^{n} q_j = 1$. We compare the ith effect e_i in the correct action model with the jth effect f_j in the learned one. Let the number of atomic formulas belonging to e_i , not to f_j ,be n_{miss} ; let the number of atomic formulas belonging to f_j , not to e_i ,be n_{extra} .The number of errors is denoted by $n_{ij} = n_{miss} + n_{extra}$ .Therefore, we can calculate the average number of errors of action a as $n_{effect} = \sum_{i=1}^{m} p_i \sum_{j=1}^{n} n_{ij} q_j$ .Let the number of possible errors be N_{effect} ,then the error rate of action a can be calculated as $P_{effect} = \frac{n_{effect}}{N_{effect}}$.

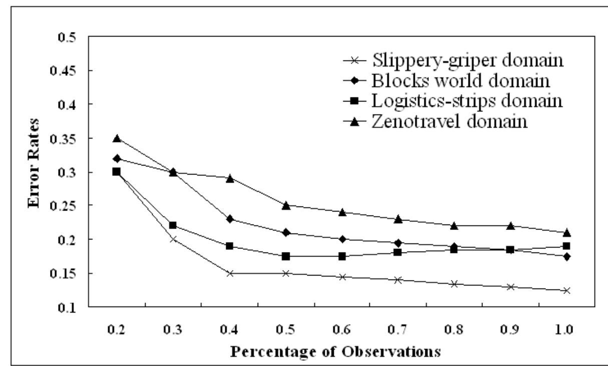
Furthermore, for action a, the error rate of action model with probabilistic effects can be defined as $R(a) = \frac{1}{2}(P_{pre} + P_{effect})$.Here we assume that the error rates of preconditions and effects were equally important, and the range of error rate $R(a) \in [0,1]$ .Moreover, the error rate of all the action models A in a domain is defined as $R(A) = \frac{1}{|A|} \sum_{a \in A} R(a)$,where $|A|$ is the number of A's elements.

B. Accuracy and the Observed Intermediate States

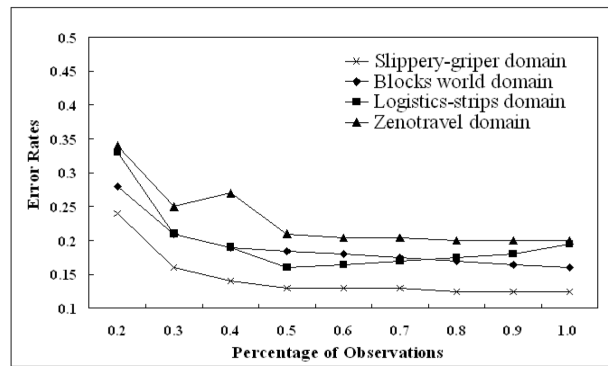
To simulate partial observation between two actions in a plan trace, from the plan traces, we randomly select observed states with specific percentage of observations 1/5, 1/4, 1/3, 1/2, 1. For each percentage value, e.g. 1/3, we randomly select an observation within three consecutive states in a plan trace. We run the selection process three times. IPRA generates learned action models each time, and meanwhile error rates are calculated. Finally, we calculate an average error rate on the plan traces. The results of these tests are shown in Figure 1.



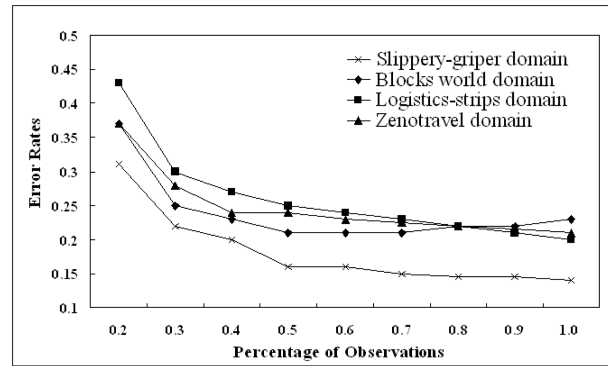
(a) Threhold t = 0.5



(b) Threhold t = 0.01



(c) Threhold t = 0.5



(d) Threhold t = 0.001

Figure 1. Error Rates of Learned Action Models in Different Domains

Figure 1 shows the performance of the IPRA algorithm with respect to different threshold values t used to select the candidate formulas, which are set to be 0.001, 0.01, 0.1 and 0.5, respectively. From the results, we find that error rate is sensitive to the choice of threshold. Generally, thresholds shall not be set to be extremely smaller or bigger. A bigger threshold will miss out some useful formulas, meanwhile a smaller threshold will cover some formulas with noise. From these experiments, it is shown that when the threshold is set to be 0.1, the mean average accuracy is optimal. Furthermore, the error bars representing the confidence intervals, show that our algorithm performance is stable.

The result also shows the relationship between the accuracy of learned model and percentage of observed

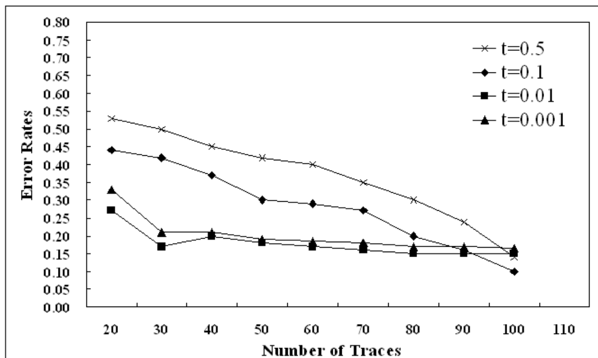
© 2014 ACADEMY PUBLISHER

intermediate states. In most cases, the more observations we have, the lower the error rate will be, which is consistent with our intuition. However, there are some cases, e.g., when threshold *t* is set to be 0.5, and there are only 1/4 of the states observed, the error rate is lower than the case when 1/3 of the states are given. These cases are not consistent with our intuition, but they are possible, since when more observations are obtained, the weights of their corresponding formulas go up and the weights of other formulas may go down in the whole learning process. Thus, if the threshold *t* is still set to be 0.5, some formulas which were chosen before are missed out, and the error rate will be higher. Thus, we conclude that in these cases, we need to reduce the value of the threshold correspondingly to make the error rate lower.

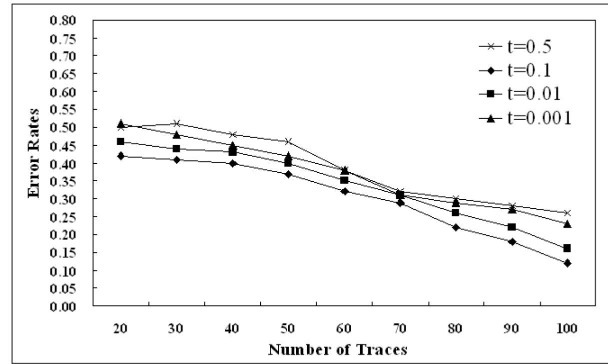
When threshold is set to 0.1, comparing the error rates of the four domains, it is obviously observed that the error rate of more complicated domain (with more predicates and actions) is generally higher than that of other domains, while with the increase of the number of plan traces, the error rate will decrease to about 10%. The reason is that in those complicated domains, a large number of predicates and actions will result in more candidate formulas of preconditions and conditional effects. In this case, if we don’t have enough number of plan traces, then the noise in the experimental result will be quite serious. Therefore, in those complicated plan domains, the number of plan traces should be at least 100.

C. Plan Traces in Action-model Learning

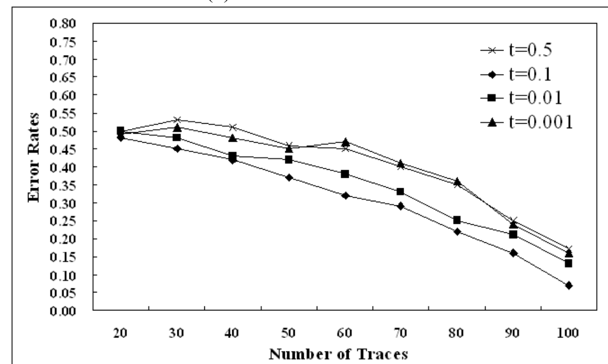
To see how error rate are affected by the number of plan traces, we used different number of plan traces as the training data to evaluate the performance. In experiments, we assume that each plan trace had 1/5of fully observed intermediate states. These observed states were randomly selected. The process of generating state observations is repeated five times, where each time an error rate is generated under different selections. Figure 2 shows that error rates are affected by the number of given plan traces.



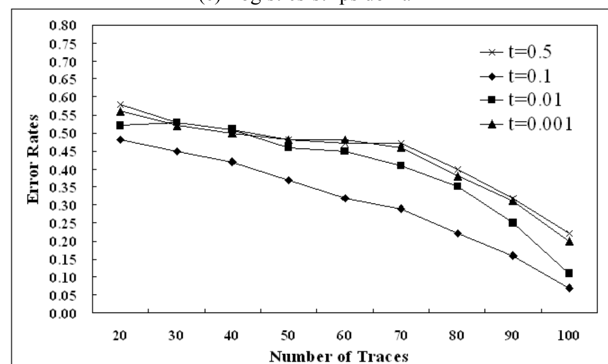
(a) Slippery-griper domain



(b) Blocks world domain



(c) Logistics-strips domain



(d) Zenotravel domain

Figure 2. Error Rates of different Number of Plan Traces (Observed Intermediate States is 1/5)

Generally, error rate decreases when the number of plan traces increase. When the number of plan traces is smaller, the error rate is higher. When the number of plan traces increases to some extent, the error rate decreases rapidly, but eventually it goes down slowly. It means that the difference between learned action models and correct ones is obvious when information is limited, but the difference will decrease when enough information is available. It can be speculated that learned action models will be approximate to correct ones, when enough number of plan traces is available.

When threshold is set to 0.1, comparing the error rates in the four domains, it is obviously observed that the error rate of the more complex domain (with more predicates and actions) is generally higher than the others. With the increase of the number of plan traces, the error rate will decrease to lower than 10%. The reason is that in those complex domains, a large number of predicates and actions will result in more candidate formulas of preconditions and effects. In this case, if we have not

© 2014 ACADEMY PUBLISHER

enough number of plan traces, then the noise in the experimental result will be quite serious. Therefore, in those complicated plan domains, the number of plan traces will be more than 100.

VI. Conclusion

In this paper, we adopt methods of automated planning and machine learning to translate software requirements into partial planning domain, formally described by PDDL language. Then we build up an action model learning algorithm to obtain complete planning domain and requirements specification. The proposed method can be used to acquire software requirement automatically. In future, we are planning to improve IPRA algorithm to apply it in the problem of system re-configuration at runtime.

References

[1] Zhao Y, Dong J, Peng T. Ontology Classification for Semantic-Web-Based Software Engineering. IEEE Transactions on Services Computing, 2009, 2(4):303~317.
[2] Drouin N, Badri M, Touré F. Analyzing Software Quality Evolution using Metrics: An Empirical Study on Open Source Software. Journal of Software, 2013, 8(10): 2462~2473.
[3] Fikes R, Nils J. N. Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 1971,2(3):189~203.
[4] Soderland S, Weld D. Evaluating nonlinear planning. Technical Report TR 91-02-03. University of Washington CSE, 1991.
[5] Kautz H, McAllester D, Selman B. Encoding plans in propositional logic. In Proceedings of the 5th International Conference of Principles of Knowledge Representation and Reasoning, 1996.1084~1090.
[6] Avrim L. B, Merrick L. F. Fast planning through planning graph analysis. In Proceeding of the 14th International Joint Conferences on Artificial Intelligence, 1995:1636~1642.
[7] Malik G, Adele H, Craig K, Drew M, Ashiwin R, Manuela V, Daniel W, David W. PDDL-the planning domain definition language, http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0203/pddl.pdf,1998.
[8] Smith D, Weld D. Confor m ant graphplan. Proceeding of 15th National Conference on Artificial Intelligence, 1998.
[9] Weld D, Anderson C, Smith D. Extending graphplan to handle uncertainty and sensing actions. Proceedings of 15th National Conference on Artificial Intelligence, 1998.
[10] Chen Y. Constrained partitioning in penalty formulations for solving temporal planning problems. Artificial Intelligence, 2009, 170(3):187~231.
[11] Philipple L. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. Artificial Intelligence, 2009,143(2):151~188.
[12] Omid M, Steve H, Anne C. On the undecidability of probabilistic planning and related stochastic optimization problems. Artificial Intelligence, 2013, 147:5~34.
[13] Younes H, Littman M. L, Weissman D. The first probabilistic track of the international planning competition. Journal of Artificial Intelligence Research, 2010,24:851~887.
[14] Zhou JP, Yin MH, Gu WX, Sun JG. Research on Decreasing Observation Variables for Strong Planning under Partial Observation. Journal of software, 2009, 20(2):290~304.
[15] Yan SY, Yin MH, Gu WX, Liu XF. Research and advances in probabilistic planning. CAAI Transactions on Intelligence Systems, 2008, 3(1):9~22.
[16] Yolanda G. Learning by experimentation: Incremental refinement of incomplete planning domains. In Proceeding of the Eleventh International Conference on Machine Learning(ICML 1994), 1994. 87~95.
[17] Thomas J. W, Michael L. L. Efficient learning of action schema and web-service descriptions. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008), 2008. 714~719.
[18] Stanley K, Parag S, Matthew R, Pedro D. The Alchemy System for Relational AI. University of Washington, Seattle, 2005.
[19] Tao C, Li B, Gao J. A Systematic State-Based Approach to Regression Testing of Component Software. Journal of Software, 2013, 8(3):560~571.
[20] Pervez Z, Khattak A M, Lee S, Lee Y K. Achieving Dynamic and Distributed Session Management with Chord for Software as a Service Cloud. Journal of Software, 2012, 7(6):1403~1412.
[21] Wu K D, Liu W, Jin Z. Managing Software Requirements Changes Based on Negotiation-Style Revision. Journal of Computer Science and Technology, 26(5):890~907, 2011.
[22] Peng X, Yu Y, Zhao W. Analyzing evolution of variability in a software product line: From contexts and requirements to features. Information and Software Technology, 53(7):707~721, 2011.
[23] Perini A, Susi A, Avesani P. A Machine Learning Approach to Software Requirements Prioritization. IEEE Transactions on Software Engineering, 39(4):445~461, 2013.
[24] Portillo-Rodriguez J, Vizcaino A, Piattini M. Tools used in Global Software Engineering: A systematic mapping review. Information and Software Technology, 54(7): 663~685, 2012.

Hong He received his B.S. degree at Wuhan University of Technology in 1996, and M.S. degree at Xiangtan University in 2006. Currently, he works in Hunan Institute of Engineering as an associate professor. His research interesting is grid computing, cloud computing, distributed resource management.

Dongbo Liu received his master degree in Hunan University in 2004. Now he works in Hunan Institute of Engineering and is a Ph.D candidate in Hunan University. His research interests include distributed intelligence, multi-agent systems, high-performance application. He is now a student member of CCF in China, and worked as Senior Engineer in HP High-performance Lab.

© 2014 ACADEMY PUBLISHER