Hierarchical Procedural Knowledge Learning Through Observation using Inductive Logic Programming, an Extended Abstract

Tolga Könik **John E. Laird**
konik@umich.edu laird@umich.edu
Artificial Intelligence Laboratory
ATL., Univ. Michigan, 1101 Beal Ave., MI, USA 48109

Creating intelligent agents that perform tasks on complex domains is a difficult and time-consuming process. (van Lent 2000) describes a framework to learn procedural knowledge from behavior traces of experts who perform tasks in domains such as Quake, a first-person perspective computer game. Since KnoMic, van Lent's specific implementation of this framework, uses a simple attribute-value based machine-learning algorithm, it would encounter difficulties in dealing with multiple objects of the same kind (i.e. multiple enemy creatures) or if the structured domain knowledge such as a playground map in Quake domain were the essential part of choosing and implementing the right strategy. We currently work on integrating a learning by observation framework with inductive logic programming (ILP) techniques to overcome these difficulties.

To reduce the complexity of the learning problem, the task performance knowledge we want to obtain is decomposed into a hierarchy of operators, each of which should encode the strategies to execute a sub-task (Figure 1). For example in Quake domain, if the agent decides to get an item *item1* in a different room by selecting the operator *get_item*(*Item*) with the instantiation *Item*= *item1*, to achieve the task it could select the sub-operator *goto_door*(*Door*), where *Door* should be instantiated with the door object on the shortest path from current room to the room where *item1* is in.

The learning by observation framework is depicted in Figure 2. Before training starts, the expert performs tasks on the environment using an interface, which captures the *behavior trace*, a symbolic representation of the changes in the environment as perceived from the expert's perspective. This representation contains, in addition to numeric sensors such as *self_x_coordinate*, structured sensors such as *self_current_room* pointing to a room object, which is part of a map structure of rooms, doors, items etc (Figure 4). In addition to that, the expert has to annotate time intervals in the behavior trace with the names and parameters of the operators he/she is executing. The interface should provide an abstract graphical representation such as the one in Figure 3, which could be used by the expert to select operator parameters that correspond to objects in the environment. This ensures that the selections of the expert correspond to the internal object representation of the interface. Any object that the expert might use in annotations as a parameter should be included in that representation.

In the next step, positive and negative examples are selected for different kind of concepts that can help to decide when to start and end operators. For example, a *goal_condition* concept of an operator will be used by the agent in determining when a selected operator should be terminated. For this concept, the negative examples are picked from regions of the trace where the expert continues to execute the operator, and the positive examples from where the expert has terminated the operator. Similarly, the positive examples of a precondition concept are selected from states just before the operator was selected and negative examples from states where other operators were preferred in the same context (when the same high-level operator is active). The examples of a concept for a given operator are selected only from regions where the parent operator is active. This simplifies the conditions by excluding conditions for selecting the high-level operator.

The background knowledge is composed of ground clauses describing the relations that hold at each state of the trace (i.e. *contains_item*(*state10*, *room2*, *item13*)) and some implicit clauses that describe domain knowledge such as how to find shortest path between two rooms. We have tested our initial ideas on a simplified model of Quake domain with randomly generated examples. Using Progol 4.4 (Muggleton 2000), we have obtained the precondition concept for *goto_door* operator (Figure 5). In this example, the learning system was able to model the decision using structured sensors (i.e. *current_room*/2), domain knowledge (i.e. *shortest_path*/3), specific task knowledge (room *r1* is connected to room *r2*), and higher level operators (representing context and intensions of the expert) including the objects that are mentioned in these (i.e. *active_get_item*/2). Once learning is finished, the learned Prolog clauses will be converted to rules of a Soar (Laird; Newell, and Rosenbloom 1987) agent, which should then interact with the environment similar to the way the expert does. We currently are working on building an experimentation interface between Aleph[1] and Soar and to start conducting experiments with real expert traces.

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/ machlearn/Aleph/

Our system extends van Lent's work (2000) by allowing parameters in operators, structured domain knowledge, and structured sensors. In (Klingspor; Morik, and Rieger 1996) a robot learns a hierarchy of concepts that correspond to abstract world states with temporal extend. For example it learns to detect *going_through_a_door* concept based on lower level concepts such as *being_in_front_of_the_door* and actions such as *move_forward* etc. In contrast, our system learns to select and execute operators such as *go_through_door*, based on higher-level operator concepts such as *get_item*. Also, we will not use actions in the background of learning, but we will learn when to select actions. The agents of (Matsui; Inuzuka, and Seki 2000) and (Inuzuka; Onda, and Itoh 2000) learn under which situations their actions will be successful by experimenting in the domain using an external criteria of success. (Benson and Nilsson 1995) describe a system that learns how actions of the agent change the environment. All of these systems learn what will happen if an agent behaves in a particular way. In contrast, our purpose is to model the decision-making expertise of when and how to exhibit behaviors.

Our main motivation in this research is to improve the performance of intelligent agents by using the strengths of ILP algorithms that attribute-value based machine learning algorithms lack but we also hope that the complex environments we plan to work on will create productive challenges for ILP community.

References

Benson, S. and Nilsson, N. 1995. Inductive Learning of Reactive Action Models. *Machine Learning: Proceedings of the Twelfth International Conference* . San Fransisco, CA: Morgan Kaufmann.

Inuzuka, N., Onda, T., and Itoh, H. 2000. Learning Robot Control by Relational Concept Induction With Iteratively Collected Examples. In Advances *in Robot Learning, Proceedings*, Lecture *Notes in Artificial Intelligence 1812*. Springer-Verlag Berlin.

Klingspor, V.; Morik, K. J., and Rieger, A. D. 1996. Learning Concepts From Sensor Data of a Mobile Robot. *Machine Learning*. 23(2-3):305-332.

Laird, J.; Newell, A., and Rosenbloom, P. 1987. Soar: An Architecture for General Intelligence. *Artificial Intelligence*. 33:1-64.

Matsui, T.;Inuzuka, N.; and Seki, H. 2000. A Proposal for Inductive Learning Agent Using First-Order Logic . In *The 10th International Conference on Inductive Logic Programming (ILP 2000).*

Muggleton, S. 2000. CProgol4.4: a Tutorial Introduction. In *Inductive Logic Programming and Knowledge Discovery in Databases*. Springer-Verlag.

van Lent, M. 2000. Learning Task-Performance Knowledge Through Observation . *Ph.D. diss.* Computer Science and Electrical Eng. Dept, Univ. Michigan.

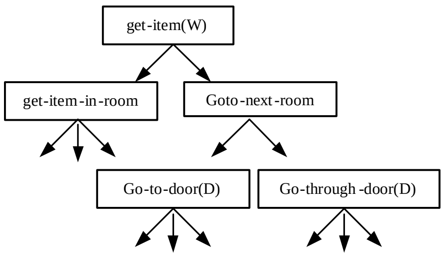

Figure 1. Hierarchical Task Decomposition

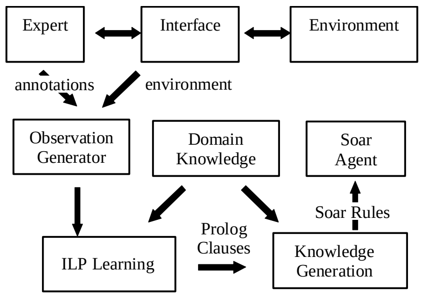

Figure 2. Learning By Observation Framework

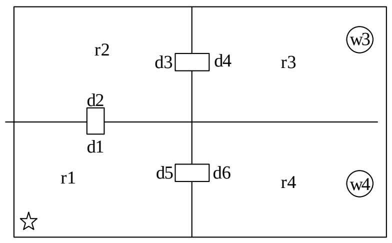

Figure 3. Map Interface

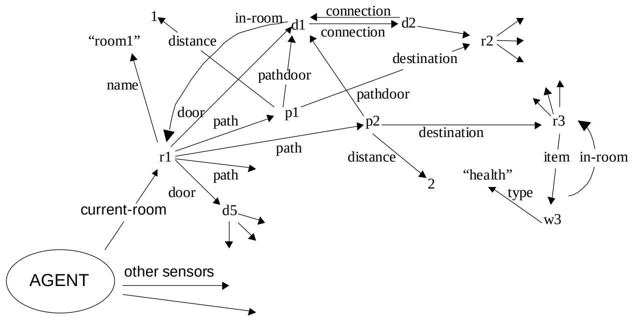

Figure 4. Map Structure in Quake Domain

```
precond_gotodoor(+STATE,_Door):-
    active_get_item(STATE, Item),
    current_room(STATE, CurrentRoom),
    shortest_path(STATE, CurrentRoom, Path),
    destination(STATE, Path, TargetRoom),
    contains_item(STATE, TargetRoom, Item),
    pathdoor(STATE, Path, Door).
```

Figure 5. Precondition Concept for goto_door Operator