Learning Conceptual Predicates for Teleoreactive Logic Programs

Nan Li, David J. Stracuzzi, and Pat Langley

School of Computing and Informatics, Arizona State University
Tempe, Arizona 85281 USA
{nan.li.3|david.stracuzzi|langley}@asu.edu

Abstract. Teleoreactive logic programs provide a formalism for describing conceptual and skill knowledge that is organized hierarchically. However, manual construction of the conceptual clauses is tedious and often requires expert knowledge. In this paper, we present an approach to defining new conceptual predicates from successfully solved problems. We provide experimental results that demonstrate these concepts improve the usefulness of skills learned from the same solutions.

1 Introduction

Teleoreactive logic programs encode both declarative and procedural knowledge into hierarchical first-order knowledge bases [1] using a syntax similar to the first-order Horn clauses in Prolog. The term "teleoreactive" [2] refers to the formalism's support for reactive execution of the goal-oriented skills over time. Teleoreactive logic programs are often created manually using expert knowledge, but this approach is both tedious and time-consuming.

There has been a growing body of work on learning teleoreactive logic programs, hierarchical task networks, and related structures [1, 3]. An important subtask involves acquiring the preconditions for the learned procedural clauses. These determine when specific clauses apply, and therefore guide the system to select procedures that take it toward the goal. In this paper, we report an approach to defining new predicates that encode these preconditions in ways that improve the behavior of learned skills over that of previous methods.

2 A Review of Teleoreactive Logic Programs

Teleoreactive logic programs incorporate ideas from traditional logic programming, but differ in that they carry out action over time. The formalism combines techniques from goal-driven and reactive control, and it incorporates constraints that make learning of hierarchical structures tractable. In this section, we briefly review the basic assumptions and operational procedures that Langley and Choi [1] introduced in their early work on this topic.

Programs in this framework distinguish conceptual and procedural knowledge. The conceptual knowledge base comprises a hierarchy of first-order Horn

Appearing in the *Proceedings of the Late-Breaking Papers Track of the Eighteenth International Conference on Inductive Logic Programming*, September 10-12, Prague, Czech Republic. Copyright 2008 by the authors.

Table 1. Sample conceptual clauses from freecell solitaire.

```
;; cards ?c1 and ?c2 are of different color, rank of ?c2 is one larger than ?c1
((stackable ?c1 ?c2)
   :percepts    ((card ?c1 color ?co1 val ?v1)
                 (card ?c2 color ?co2 val ?v2))
   :tests       ((not (equal ?co1 ?co2))
                 (= ?v2 (+ 1 ?v1))))

;; card ?c may be placed onto card ?dc, and ?cb is the card below ?c
((movable ?c ?dc ?cb)
   :percepts    ((card ?c) (card ?dc) (card ?cb))
   :relations   ((clear ?c) (clear ?dc)
                 (moved-onto ?c ?cb) (stackable ?c ?dc)))
```

clauses with negation that provide a vocabulary to describe the agent's environment. Each conceptual clause consists of a head, which states its predicate and arguments, and a body that describes the conditions under which the predicate is true, as Table 1 demonstrates. Procedural knowledge, stated as a set of skill clauses, is similar to hierarchical STRIPS operators [4]. Each skill clause has a head that refers to the skill's goal, a start condition that must be satisfied before it can execute, an action or subgoal field that describes how to achieve the goal, and an effects field that describes the situation after successful execution. The predicate in a clause head may appear in a subgoal, so the framework support recursive programs. Table 2 shows sample skill clauses from the freecell domain. Notice how predicates such as *movable* refer to the concepts defined in Table 1.

Teleoreactive logic programs perform two primary operations during each execution cycle. First, the interpreter carries out bottom-up inference to determine a belief state based on the agent's percepts and conceptual knowledge. Second, the interpreter retrieves the first unsatisfied top-level goal and attempts to find an applicable path through the skill hierarchy. Such a path starts from

Table 2. Sample skill clauses from freecell solitaire.

```
;; Clear card ?cb by moving the card ?c on top of it to card ?dc
((clear ?cb ?c ?dc)
   :percepts    ((card ?c) (card ?dc) (card ?cb))
   :start       ((movable ?c ?dc ?cb))
   :actions     ((*sendtocol ?c ?dc))
   :effects     ((clear ?cb) (moved-onto ?c ?dc) (clear ?c)))

;; Move card ?c on to card ?dc
((moved-onto ?c ?dc)
   :percepts    ((card ?c) (card ?dc) (card ?cb))
   :start       ((precondition-moved-onto_s8 ?c ?dc))
   :subgoals    ((movable ?c ?dc ?cb)
                 (clear ?cb ?c ?dc))
   :effects     ((effect-moved-onto_s8 ?c ?dc))
```

the agent's goal, which is an instance of a known concept, and descends through the hierarchy such that the preconditions of each skill clause match and the bindings of each subgoal unify with those of its parent.

If no applicable skill path exists, a problem solver decomposes the goal by chaining backward using domain knowledge. The problem solver only backchains over concept definitions and primitive skills, which refer to executable actions rather than subgoals. To chain off of a skill, the interpreter retrieves a skill that contains the current goal in its effect and attempts to achieve the preconditions for that skill. Similarly, when chaining off a concept, the system uses its definition to decompose the current goal into multiple subgoals.

Whenever the problem solver achieves a goal or subgoal, it constructs a new skill clause. The head of the skill is a generalized version of the goal that replaces constants with variables. If chaining off a skill achieved the goal, the new skill's subgoals are the precondition concept from the chained skill, plus the subgoals of the chained skill in order of execution. The precondition of the new skill is the precondition of the skill that achieved the first subgoal. If chaining off a concept achieved the goal, the new skill's subgoals are the subconcepts that were unsatisfied at the start of problem solving.

When the system encounters a similar situation in the future, its interpreter will test whether the new skill clause appears in an applicable path through the skill hierarchy, in which case it will execute that path. Experimental studies suggested that this approach to learning hierarchical skills rapidly replaced problem solving, which often required extensive backtracking, with reactive execution, which often led directly to the goal.

3 Learning and Using Conceptual Predicates

Langley and Choi's [1] skill-learning method produced encouraging results, but analysis suggested it has two drawbacks. First, skill clauses produced from solutions obtained via chaining off of concepts tend to have overly general preconditions because they ignore the preconditions of skill clauses that achieve the subgoals. Second, skill clauses constructed for goals achieved via skill chaining tend to have overly specific preconditions, since they consider only the particular primitive skills used to achieve the first subgoal and ignore other skill clauses that may achieve that subgoal.

We have addressed these issues by developing an extended approach which defines new conceptual predicates that better reflect a learned skill clause's applicability. Preconditions and effects define abstractions of the world before and after the skill executes. We can expand the system's knowledge about the world by constructing new predicates that encode these abstractions effectively.

The new approach introduces two kinds of terms: specialized predicates and generalized predicates. Each specialized precondition/effect is associated with a skill clause and describes the situations in which that skill applies/produces. The system uses specialized preconditions during execution to determine which skill to apply next. A generalized precondition/effect is associated with a goal

and encodes a disjunction over the specialized preconditions/effects from all skill clauses that achieve the goal. Our approach uses generalized preconditions and effects during learning to determine the specialized precondition and effect for a new skill that admits all possible uses of that skill, as detailed below.

The input to our method is a skill clause, built by the skill learner, which contains a goal and a list of subgoals. If the system achieved the goal by chaining off a concept, it first retrieves the generalized preconditions and effects associated with the subgoals. It then uses macro-operator composition [5] to compute the preconditions and effects by combining the generalized preconditions and effects of the subgoals. Finally, the system introduces new precondition and effect predicates using these two combined terms as their definitions. Similarly, if the system achieved the goal by chaining off a skill, the specialized precondition of the new skill is the generalized precondition of the first subgoal, S_1. Similarly, the specialized effect of the learned skill clause is the effect of the original skill.

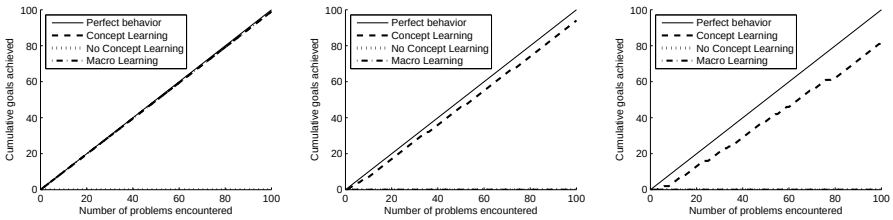
Irrespective of whether the problem solver used concept or skill chaining to achieve the goal, the algorithm updates the generalized predicates corresponding to the new specialized predicates by adding the precondition/effect predicates as disjunctive terms. This informs the interpreter that the given goal can be achieved in a new situation and lets it apply learned skill clauses to goals under circumstances in which it would otherwise have ignored them.

4 Experimental Evaluation

Our key claim is that expanding the representation of teleoreactive logic programs by defining new conceptual predicates improves the ability of learned skills to achieve goals and reduces their reliance on problem solving. To test this claim, we carried out an experiment that compared the new approach with the one that Langley and Choi [1] reported and with Mooney's [5] method for learning a non-hierarchical macro-operators. The latter determines preconditions using an analytic techniques similar to the one we described for computing specialized preconditions, but without introducing new predicates. We used the same inference, execution, and problem-solving modules in each case.

We presented each system with 100 randomly selected problems from the freecell domain [6]. For each problem, a system first tried to achieve the goal by executing its existing skills. If this failed, it called on the problem solver and learned new skill clauses, using one of the above three methods, whenever this achieved a goal or subgoal. We measured system behavior as the number of top-level goals achieved by execution without resorting to problem solving. Other metrics such as CPU time are secondary to our objectives. We provided the systems with initial knowledge bases (40 conceptual clauses and 13 primitive skills) sufficient to solve problems by execution that were one step away from the goal. We tested the system on problems with 8, 16 and 24 cards.

Figure 1 displays the cumulative number of goals achieved without problem solving by the three systems. Macro-operator learning fares the worst, while our predicate creation method learns more rapidly than Langley and Choi's tech-



(a) (b) (c)

Fig. 1. Cumulative number of goals achieved for the freecell solitaire domain on problems involving (a) eight cards, (b) 16 cards, and (c) 24 cards.

nique on tasks with 16 and 24 cards. Analysis of individual runs shows that the older approach often acquires overly general preconditions, which leads the system to execute skills that do not achieve the goal. The concept learner mitigates this drawback by learning more specific preconditions. Conversely, the macro-operator method collects too many relations among cards, which leads to overly specific preconditions that keep the interpreter from executing relevant skills. We conclude that, on average, the new mechanism creates more appropriate preconditions for skill clauses that reduce the chance of selecting irrelevant learned skills and increase the chance of selecting relevant ones.

5 Concluding Remarks

In this paper, we reviewed teleoreactive logic programs, along with an initial approach to learning them from problem solutions. We also identified some drawbacks with this scheme and described an extension for defining new conceptual predicates to produce more appropriate preconditions on learned skill clauses. An experiment demonstrated that the new mechanism learned more rapidly than either the initial technique, which formed overly general conditions, or a method based on macro-operators formation, which formed overly specific ones.

Our predicate creation algorithm has superficial similarities to work on predicate invention [7], but our approach is analytic while the latter was driven by empirical regularities. More closely related is research on representation change in problem solving and game playing [8, 9], which also relied on goal-driven analytical learning. However, our approach differs from these efforts by supporting incremental learning that is interleaved with problem solving, by acquiring recursive precondition concepts that aid generalization, and by working jointly with a method for constructing hierarchical skills that support reactive execution. Also relevant is recent work on learning hierarchical methods from problem solutions and action models [10–12], which shares many features but does not construct new conceptual predicates that extend the representation language.

The main claim of this paper is that the analytic creation of new conceptual predicates produces more appropriate preconditions for learned skill clauses,

which in turn let a teleoreactive interpreter achieve more goals through execution, without the need for problem solving. However, the results we have reported remain preliminary and suggest several avenues for additional research. For example, we should replicate out experimental studies in other domains, both to demonstrate generality and better understand the quality of the learned preconditions. We must also combine the predicate learner with an evaluation mechanism that lets the system determine which of the new concepts are useful. Finally, we should augment the framework to acquire skills for achieving these invented concepts, thus closing the loop on conceptual and procedural learning.

6 Acknowledgements

The authors would like to thank Dongkyu Choi and Tolga Konik for helpful discussions and suggestions concerning this work. This material is based on research sponsored by ONR under grant N00014-08-1-0069 and by DARPA under agreement FA8750-05-2-0283.

References

1. Langley, P., Choi, D.: A unified cognitive architecture for physical agents. In: Preceedings of the Twenty-First National Conference on Artificial Intelligence, Boston, AAAI Press (2006)
2. Nilsson, N.: Teleoreactive programs for agent control. Journal of Artificial Intelligence Research **1** (1994) 139–158
3. Ilghami, O., Nau, D.S., Muñoz Avila, H., Aha, D.W.: Camel: Learning method preconditions for HTN planning. In: Proceedings of the Sixth International Conference on AI Planning and Scheduling, Toulouse, France, AAAI Press (2002) 131–141
4. Fikes, R., Nilsson, N.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
5. Mooney, R.J.: A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding. Morgan Kaufmann, San Mateo, CA (1990)
6. Bacchus, F.: AIPS '00 planning competition. AI Magazine **22** (2001) 47–56
7. Muggleton, S.: Predicate invention and utility. Journal of Experimental and Theoretical Artificial Intelligence **6**(1) (1994) 121–130
8. Utgoff, P.E.: Shift of Bias for Inductive Concept Learning. PhD thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ (1984)
9. Fawcett, T.: Knowledge-based feature discovery for evaluation functions. Computational Intelligence **12**(1) (1996)
10. Reddy, C., Tadepalli, P.: Learning goal decomposition rules using exercises. In Fisher, D., ed.: Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN, Morgan Kaufmann (1997)
11. Nejati, N., Langley, P., Konik, T.: Learning hierarchical task networks by observation. In: Proceedings of the 23nd International Conference on Machine Learning, Pittsburgh, PA, ACM (2006)
12. Hogg, C., Muñoz-Avila, H., Kuter, U.: HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence, Chicago, AAAI Press (2008)