Effects of Multiple Learning Mechanisms in a Cognitive Architecture

Dongkyu Choi (dongkyuc@uic.edu)
Stellan Ohlsson (stellan@uic.edu)
Department of Psychology
University of Illinois at Chicago
1007 W Harrison Street (M/C 285), Chicago, IL 60607 USA

Abstract

Human learning involves multiple sources of information. Their ability to adapt to changes in the environment depends on having such multiple learning modes. In this paper, we extend an existing cognitive architecture to have three distinct learning modes, in an effort to test the hypothesis that multiple learning capabilities bring synergistic effect in the overall performance. We show experimental results in a simplified route generation domain.

Keywords: cognitive architecture, learning, multiple modes of learning, learning from success, learning from failures, learning declarative knowledge, skill acquisition

Introduction

People acquire knowledge from various sources. They learn from their own success and failures, by observing situations around them, and by imitating others' behavior. Their ability to adapt to changing situations depends on such multiple modes of learning (Ohlsson, 2011). There have been many previous work on each of such learning modes, but we are particularly interested in their interactions and the synergy among them.

Since cognitive architectures (Newell, 1990) provide general framework for modeling cognition, they are suitable for our research on multiple learning modes and their interactions. In this work, we use one such architecture, ICARUS, that supports both learning from success (Langley & Choi, 2006) and learning from failures (Choi & Ohlsson, 2010; Ohlsson, 1996). We extended the architecture with a third mode of learning declarative knowledge in an effort to broaden the scope of our research, although we do not yet perform a lesion study in the current work with this learning mode turned off.

In the following sections, we first review some basic assumptions of ICARUS starting with its representation and memories and continuing to the inference and execution mechanisms. We then describe the three learning mechanisms in some detail and show experimental results that show synergy in a route generation domain. We also discuss related and future work before we conclude.

Representation and Memories

As with other cognitive architectures (Laird et al., 1986; Anderson, 1993), ICARUS makes commitments to a specific way to represent knowledge, infer beliefs, perform execution and learn new knowledge. In this section, we review ICARUS's representation of knowledge and the corresponding memories.

ICARUS makes distinctions in two separate dimensions. The first exists between concepts and skills. Concepts give ICARUS a language to describe its surroundings by enabling the system to infer beliefs about the current state of the world. Skills, on the other hand, are procedures that ICARUS believes to achieve certain concept instances. The second distinction lies between long-term knowledge and short-term structures. Long-term concepts and skills are general descriptions of situations and procedures, and ICARUS instantiates them for a particular situation at hand. Instantiated concepts and skills are short-term structures, in that they are constantly created and destroyed as the situation changes. These two distinctions result in four separate memories in ICARUS.

In its long-term conceptual memory, the architecture encodes concept definitions that are similar to Horn clauses (Horn, 1951). As shown in Table 1, concepts include a head and a body that includes perceptual matching conditions or references to other concepts. The first concept with the head, `(at ?location)`, matches against an object of type, `self`, and its attribute, `location`, in its `:percepts` field. The second concept, `(connected ?from ?to)`, matches against an additional object of type, `location`, and tests if its `accessible` attribute is not null and the two locations, `?from` and `?to`, are different. These two concepts do not have any reference to other concepts in their definitions, so they are *primitive* concepts. On the other hand, the third concept, `(not-dead-end ?location)`, matches against a `location` object and refers to two subconcepts in addition to a test. This makes the last concept a *non-primitive* one. This way, ICARUS builds a hierarchy of concepts that provides multiple levels of abstraction.

Another long-term memory stores ICARUS's skills that resemble STRIPS operators (Fikes & Nilsson, 1971). The head of each skill is the predicate it is known to achieve, making all skills indexed by their respective goals. Each skill has a body that includes perceptual matching conditions, some preconditions, and either direct actions to the world or a subgoal decomposition. Skills with no references to subgoals are *primitive*, while the ones with subgoals are *non-primitive*. Table 2 shows some sample ICARUS skills. The first skill that achieves `(at ?location)` has two preconditions, `(at ?from)` and `(connected ?from ?location)`, in its `:start` field and an action in its `:actions` field. Without any reference to subgoals, this skill is primitive. The second skill, however, is a non-primitive one that provides a subgoal decomposition to achieve `(at B)`. Namely, this skill instructs

ICARUS to consider two ordered subgoals, `(at W6)` and `(at B)`, to achieve the eventual goal.

Table 1: Some sample ICARUS concepts for a route generation domain. Question marks denote variables.

```
((at ?location)
 :percepts  ((self ?self location ?location)))

((connected ?from ?to)
 :percepts  ((self ?self location ?from)
             (location ?to accessible ?access))
 :tests     ((not (equal ?from ?to))
             (not (null ?access))))

((not-dead-end ?location)
 :percepts  ((location ?location))
 :relations ((connected ?location ?to1)
             (connected ?location ?to2))
 :tests     ((not (equal ?to1 ?to2))))
```

Table 2: Some sample ICARUS skills for a route generation domain. Question marks denote variables. The second skill is a part of a specific route that is learned.

```
((at ?location)
 :start    ((at ?from)
            (connected ?from ?location))
 :actions  ((*move-to ?location)))

((at B)
 :subgoals ((at W6) (at B)))
```

In addition, ICARUS has two short-term memories to store instantiated concepts and skills. While a short-term conceptual memory holds the current beliefs of the system, its short-term skill memory stores the selected skill instances indexed by their corresponding goals or subgoals. When ICARUS works on complex problems, information on goals and subgoals tends to dominate the short-term skill memory since it also serves as the goal stack for the system. Next, we explain the processes that generate contents of these short-term memories from their long-term counterparts.

Inference and Execution

ICARUS operates in cycles. On each cycle, it performs a series of processes as shown in Fig 1. The system first instantiates its long-term concepts based on the data from its sensors. The bottom-up inference of concepts creates beliefs in the form of instantiated conceptual predicates. The inference process starts with the perceptual information about objects in the world. The system attempts to match its concept definitions to the perceptual information and, when there is a match, it instantiates the head of the definitions to compute its current beliefs.

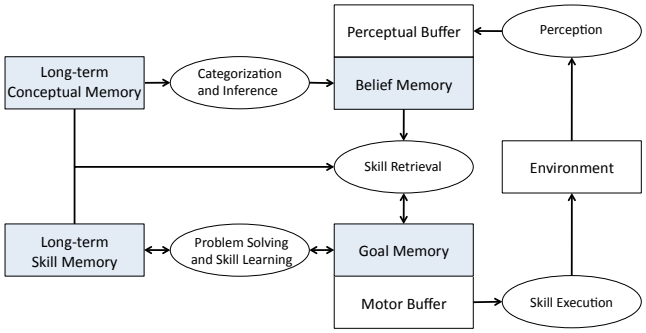

Figure 1: ICARUS's memories and the processed that work over them on each cycle.

Once the architecture computes all its beliefs, it starts the skill retrieval and execution process. ICARUS's goals guide this process, and the system retrieves relevant long-term skills based on the current beliefs. When it finds an executable path through its skill hierarchy, from its goal at the top to actions at the bottom, ICARUS executes the actions specified at the leaf node of the path. This execution, in turn, changes the environment, and the system starts another cycle by inferring beliefs from new data received from the environment.

Learning Mechanisms

The original ICARUS includes a single learning mechanism that acquires new skills from successful problem solving traces (Langley & Choi, 2006). It uses a version of means-ends problem solver to decompose its goals into subgoals and generate a solution trace. The system then uses it to compose a new skill for each subgoal. Our recent work added a second mechanism for learning from failures using a new constraint language (Choi & Ohlsson, 2010). Given constraints that are expressed as relevance–satisfaction condition pairs, the system revises its skills that cause violations of such constraints by adding new preconditions to them.

Although our previous observations showed that the two learning mechanisms yield some synergistic effects, we were concerned that the overall system assumes fully observable domains and this causes ICARUS's problem solver to be overly powerful. In response, we moved toward a partially observable domain and introduced a third learning mechanism to ICARUS that can handle this change. In this section, we review the two existing learning mechanisms briefly and explain the details of the new extension to ICARUS for learning declarative knowledge.

Learning from Problem Solving

When ICARUS hits an impasse with no executable skills for the current goal, it invokes its means-ends problem solver to find a solution. As shown in Figure 2, the system has two options, either using a skill definition to propose an unsatisfied precondition as the next subgoal (*skill chaining*), or using a concept definition to decompose the current goal into subgoals (*concept chaining*). By default, ICARUS gives priority to the former and proceeds to the latter only when there is no skill chains available.

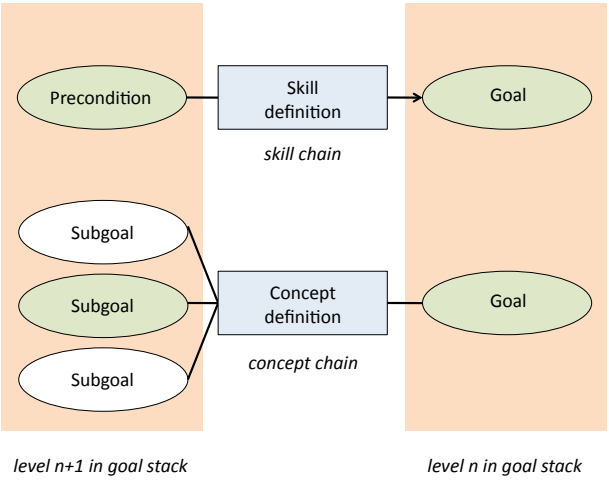

Figure 2: Two types of problem solving chains in ICARUS. For a skill chain, the system uses a skill definition to push the unsatisfied precondition as subgoal, while in a concept chain it uses a concept definition to decompose a goal into subgoals.

The architecture applies problem solving chains recursively until it finds a subgoal for which it can execute immediately. When such a subgoal is found, ICARUS proceeds with the execution to achieve it. Once the system satisfies the subgoal in the world, it learns a new skill from this experience by generalizing the situation and the procedures used.

Learning from Constraint Violations

ICARUS has the notion of constraints, expressed as pairs of relevance and satisfaction condition adopted from Ohlsson (1996). On every cycle, the system checks if the relevance conditions of each constraint is true in its belief state, and if so, it also checks the satisfaction conditions. When a constraint is violated, namely, when it is relevant but not satisfied, ICARUS invokes its constraint-based specialization mechanism to revise skills that caused the violation.

There are two different cases of violations. One is when a constraint has been irrelevant but it becomes relevant and not satisfied, and the other is when a constraint has been relevant and satisfied but it becomes unsatisfied. The system treats the two cases differently, using two distinct rules as shown in Table 3 to compute additional preconditions it adds to the corresponding skills. See Ohlsson (2011) for a more detailed description of this learning mechanism.

Table 3: Added preconditions computed differently based on the type of the constraint violation. C_r, C_s, O_a, and O_d denote relevance conditions, satisfaction conditions, add list, and delete list, respectively.

Type \ Revision	1	2
A	$\neg(C_r - O_a)$	$(C_r - O_a) \cup (C_s - O_a)$
B	$\neg C_r$	$C_r \cup \neg(C_s \cap O_d)$

Learning Declarative Knowledge

The newest learning mechanism in ICARUS outputs a different kind of knowledge. Instead of creating or revising skills, it remembers and maintains declarative knowledge in the form of ICARUS beliefs. The original architecture performs bottom-up inference of its beliefs from scratch on every cycle, but the extended system carries over some of previous beliefs while it still infers new ones based on the updated perceptual information on a cycle.

This process happens in a straightforward fashion. ICARUS first performs the bottom-up inference with updates from the environment. It then compares this belief state to the previous one and finds conflicting beliefs in the previous state that get removed. The rest of previous beliefs get added to the current belief state. What is crucial in this process is the mechanism for removing conflicting beliefs from the previous state. ICARUS uses negations in the definitions of concepts to find the beliefs that are, in some sense, opposed to new beliefs in the current state. However, this is not enough to find all conflicts, and it causes a catastrophic expansion of beliefs when it operates alone.

Therefore, the latest extension also includes a new field `:delete` in concept definitions that stores what is similar to a *delete list*. Since not all conflicting relations are explicitly expressed in the form of negated subconcepts, developers can manually add such relations in concept definitions. This is particularly useful to ensure uniqueness of some concept instances like an agent's current location. For example, in the definition for `(at ?location)` that shows the current location of the ICARUS agent in a route generation domain (see the first concept in Table 1), we can guarantee that only one instance of this concept exists in the belief state on a given cycle by including `(at ?other)` in the delete list for this concept. This process is shown in Figure 3.

Experiments

To prove the synergistic effect of having multiple learning mechanisms, we performed experiments in the route generation domain we have developed for our previous work (Choi & Ohlsson, 2010) with some modifications to make it partially observable. Instead of seeing all connections at all

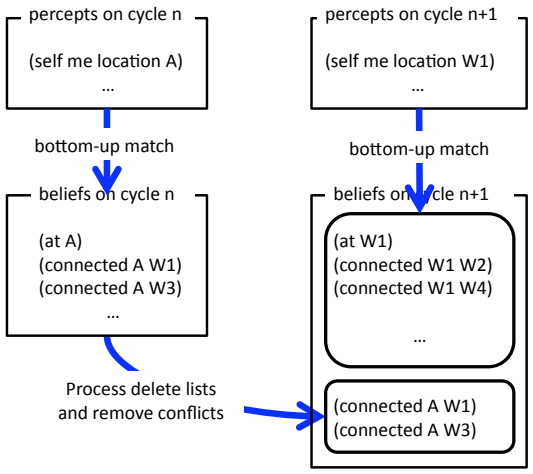

Figure 3: After inferring the current beliefs from the perceptual information on cycle n+1, ICARUS combines this result with previous beliefs that are both not in conflict with the current ones and not removed while processing the delete lists. In this example, only static beliefs for connectivity are maintained in the belief state for cycle n+1.

times, the agent can only see the connections from its current location to the neighboring ones. In this section, we describe our experimental setup and the results from our experiments in this domain.

Experimental Setup

In our route generation domain, the agent starts at a location on one side, and it has the goal to get to a target location on the other side. Using the connectivity information between various neighboring locations, an agent should traverse from its origin to the target. Although there are multiple possible routes in the environment, some of the routes might become unavailable for travel due to various reasons such as criminal activity or damage to a bridge. When this happens, the agent can encounter situations where it is unable to use routes it has learned before, requiring it to adapt to the new situation.

The domain is modified from its original form to give partial knowledge of the environment to ICARUS agents, restricting the available connectivity information to the visible ones from the agent's current location. We give only the basic concept and skill sets to the system at the beginning, along with a constraint. This means that the system knows how to operate in the world, but not at the level of expertise that enables it to satisfy the constraint at all times.

A typical run in this domain goes as follows. We give the system a goal to get to a target location, *B*, starting from the initial location, *A*. The two locations are connected by two alternate routes using waypoints *W1* and *W2*, respectively. From the location *A*, the agent sees connections to the neighboring locations, *W1* and *W2*. Without a complete connectivity information from the current location to the target, both execution and problem solving fails, and the system falls back to random exploration. This gets the agent to a waypoint, *W1*. From this location, the agent can see a direct connection to its target, which it takes by executing its skill for moving between neighboring locations (the first skill in Table 2).

Once the agent reaches the target, it is transported back to its origin for repeated trials. During the first trial, the system has remembered all the connectivity information it saw using declarative learning. Therefore, on the second trial, the problem solver can generate the route, *A - W1 - B*, from the beginning. It then takes the route by executing its skill twice for the two segments and achieves its goal. From this success, the agent learns the route as specific skills like the second one in Table 2.

Before continuing subsequent trials, we designate the waypoint *W1* as dangerous. This causes a violation of the constraint ICARUS is given, namely:

```
(at ?location) → (not (dangerous ?location))
```

which simply says that it should not be at a location that is dangerous. During the next trial, the system attempts to use the known route, *A - W1 - B*, but it realizes that taking this route would cause a constraint violation. In response, ICARUS revises its skill to include an additional precondition, which ensures that the location the skill takes it to is not dangerous. Then again, there is no executable skill from *A*, and the system finds an alternate route *A - W2 - B* through problem solving and learns a specific skill for this route. Starting from the next trial, the agent can simply execute its specific route skills to get to the target without any problem solving.

Experimental Results

We ran similar experiments at two different levels of complexity, with 100 simulated subjects for each. There are nine waypoints and four different routes between the origin and the target at the first level, while there are 12 waypoints and eight possible routes at the second level. There were four conditions, in which 1) we turn on all learning modes, 2) turn off learning from constraint violations, 3) turn off learning from problem solving, or 4) turn off both of these learning modes. Since turning off the declarative learning causes the system to fall back to exploration all the time, we did not include any conditions that involve turning off this learning mode.

Figure 4 summarizes main findings. The measure of computational effort is the total number of cycles per trial. The first four trials show the initial learning of a route to target. After the fourth trial, the learned route was declared out of bounds by marking some waypoints on that route to have become dangerous. The fifth trial is thus the one in which ICARUS discovers this change and faces the problem of adapting to it. The subsequent trials trace the discovery and learning of a novel route. The figure shows four curves for each complexity level, corresponding to the four learning conditions outlined above.

The curve at the top of these graphs shows the result for the fourth condition where only the declarative learning is active.

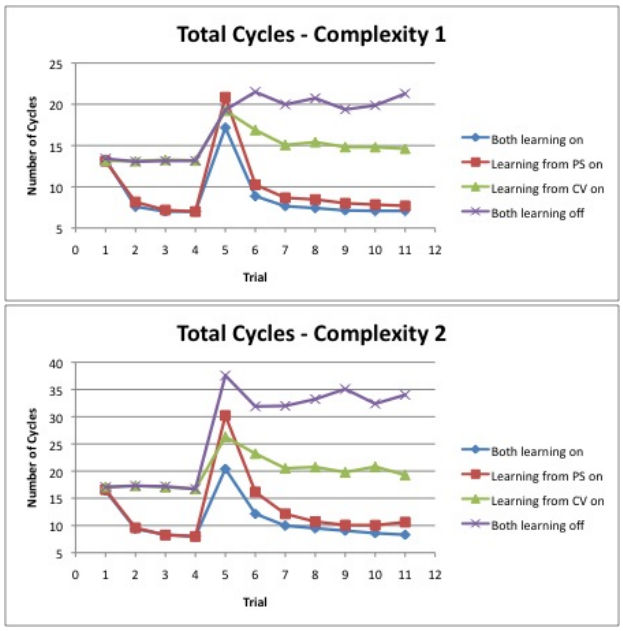

Figure 4: Number of cycles taken to reach the target location in situations with three different levels of complexity. Four conditions are shown in different shapes and colors consistently throughout the three graphs.

As one would expect, there is no change in effort across the four initial learning trials. Once the environment is changed, the system should perform search to find a path around the dangerous waypoints, so computational effort is higher on the fifth trial, and then stays high because the system does not acquire new skills from its experience.

The next curve marked with triangular shapes is for the third condition, in which learning from constraint violations is active along with the declarative learning. Since the system does not learn any specific routes under this condition, there is no noticeable difference from the above condition on the first four trials. However, after the environment is changed, the system revises its skill for moving to a neighboring location based on the expected constraint violation of being at a dangerous location. For this reason, it performs noticeably better than the above condition where only the declarative learning is turned on.

In the other two conditions where learning from problem solving is active, the system rapidly learns an initial route in the first four trials. There is no measurable difference between the two learning conditions with respect to the system's ability to learn an initial path, and the number of cycles required to traverse the landscape decreases about 50% from the first trial to the fourth. After the peaks at their fifth trials, the two conditions once again meet at roughly the same number of cycles in their steady states.

One crucial finding is that, on the fifth trial, the system adapts quicker to the changed environment when it runs with both learning from problem solving and learning from constraint violations than with either mechanism by itself. As shown in Figure 5, the synergy is substantial: the number of cycles is 17, compared to 19 for learning from constraint violations at complexity level 1 and the difference increases to 20 versus 26 at complexity level 2. These represents savings of 10% and 22%, respectively. When compared to learning from problem solving, the savings are even higher at 17% and 32%.

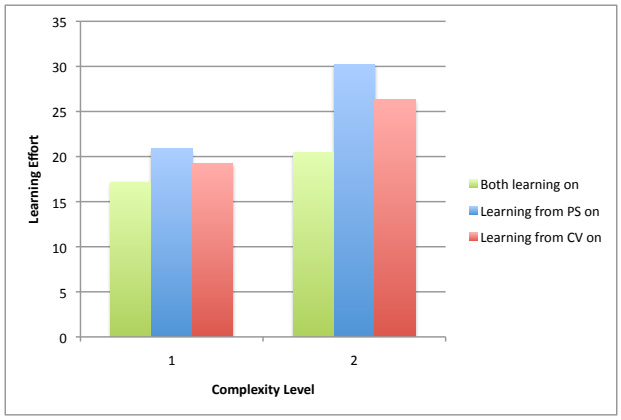

Figure 5: Learning efforts measured by the number of cycles at the fifth trial. The different between the two conditions shows the synergistic advantage of adding another learning mode as complexity increases.

Both the performance differences at the final steady state and the learning effort measured at the fifth trials suggest synergistic effects of having multiple learning mechanisms in a single system. We found that the performance of the system when the mutiple mechanisms are active is distinctly better than the performance with any one of the mechanisms.

Related and Future Work

This paper covers an ongoing research effort toward human-level variety of learning capabilities. In the current state, the system includes three different modes of learning, each of which has a vast amount of related work in the literature. First of all, learning from problem solving is closely related to previous research on macro-operators (Mooney, 1989; Shavlik, 1989) among work on explanation-based learning. The ICARUS approach shares the basic principle of composing knowledge elements into larger structures. However, it support disjunctions and recursions in the skill hierarchy, in addition to the simple fixed sequences learned in systems with macro-operators.

The mechanism for learning from constraint violations also has important similarities to previous work in explanation-based learning literature (Ellman, 1989; Wusteman, 1992). These methods assume a significant amount of domain theories presumed to be perfect. To augment this limitation, researchers worked on the similar problems of blame assignment and theory revision, although the exact formulations

were different from ours. Unlike most of these work, our approach includes explicit descriptions of constraints, which the system uses to detect failures and revise existing procedural knowledge accordingly.

In contrast to the two learning methods above, the topic of learning declarative knowledge is significantly less studied. Researchers agree on the fundamental differences between declarative and procedural knowledge (Anderson, 1976), and the both types of learning are popular research topics among neuroscientists in relation to particular brain regions (e.g., Weis et al., 2004; Quintero-Gallego et al., 2006). However, research for simulating declarative learning through computational means is not common. Chi and Ohlsson classified various types of changes to declarative knowledge as learning proceeds, but the work does not attempt to model them computationally. We extended ICARUS to support declarative learning, but the research is in a preliminary stage and requires further investigation.

Although our current work successfully shows the synergistic effects of multiple learning mechanisms in ICARUS, this research is still at an early stage. We plan to extend the architecture with yet another learning mechanism, possibly learning by analogy, to further verify our hypothesis of synergy among learning mechanisms. As this paper suggests, it is not our focus to implement a powerful single learning mechanism. Rather, we aim to build a collection of distinct learning capabilities that are written in a straightforward manner. Learning by analogy will not be an exception, and we plan to start with a simple mechanism that maps objects to similar objects or predicates to to related ones. We find research on representation mapping by Könik et al. (2009) as a good inspiration in this direction.

Conclusions

The human ability to adapt to changing situations depends on a variety of learning mechanisms. Therefore, an intelligent agent cannot be limited to a single learning mode to simulate human behavior properly. We extended ICARUS to support three different modes of learning to model this behavior. Our initial results in a route generation domain show synergistic effects of having multiple learning mechanisms, especially evident at higher levels of complexity in the environment. We plan to continue exploring additional types of learning capabilities in this framework.

Acknowledgments

This research was funded by Award # N0001-4-09-1025 from the Office of Naval Research (ONR) to the second author. No endorsement should be inferred.

References

Anderson, J. R. (1976). *Language, memory, and thought*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Chi, M. T. H., & Ohlsson, S. (2005). Complex declarative learning. In K. Holyoak & R. Morrison (Eds.), *The Cambridge handbook of thinking and reasoning* (pp. 371–399). Cambridge, UK: Cambridge University Press.

Choi, D., & Ohlsson, S. (2010). Learning from failures for cognitive flexibility. In *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society.* Portland, OR: Cognitive Science Society, Inc.

Ellman, T. (1989). Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, *21*(2), 163–222.

Fikes, R., & Nilsson, N. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, *16*(1), 14–21.

Könik, T., O'Rorke, P., Shapiro, D., Choi, D., Nejati, N., & Langley, P. (2009). Skill transfer through goal-driven representation mapping. *Cognitive Systems Research*, *10*(3), 270–285.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Langley, P., & Choi, D. (2006). Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, *7*, 493–518.

Mooney, R. J. (1989). The effect of rule use on the utility of explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 725–730). Detroit, MI: Morgan Kaufmann.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Ohlsson, S. (1996). Learning from performance errors. *Psychological Review*, *103*, 241–262.

Ohlsson, S. (2011). *Deep learning: How the mind overrides experience*. New York, NY: Cambridge University Press.

Quintero-Gallego, E. A., Gómez, C. M., Casares, E. V., Márquez, J., & Pérez-Santamaría, F. J. (2006). Declarative and procedural learning in children and adolescents with posterior fossa tumours. *Behavioral and Brain Functions*, *2*(9).

Shavlik, J. W. (1989). Acquiring recursive concepts with explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 688–693). Detroit, MI: Morgan Kaufmann.

Weis, S., Klaver, P., Reul, J., Elger, C. E., & Fernández, G. (2004). Temporal and cerebellar brain regions that support both declarative memory formation and retrieval. *Cerebral Cortex*, *14*(3), 256–267.

Wusteman, J. (1992). Explanation-based learning - a survey. *Artificial Intelligence Review*, *6*(3), 243–262.