Learning Planning Operators from Episodic Traces

David Ménager
Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS 66045 USA
dhmenager@ku.edu

Dongkyu Choi
Aerospace Engineering
University of Kansas
Lawrence, KS 66045 USA
dongkyuc@ku.edu

Mark Roberts, David W. Aha
Naval Research Laboratory, Code 5514
Washington, DC, 20375 USA
mark.roberts@nrl.navy.mil
david.aha@nrl.navy.mil

Abstract

Learning is an important aspect of human intelligence. People learn from various aspects of their experience over time. We present an episodic infrastructure for learning in the context of a cognitive architecture, ICARUS. After a review of this architecture, we formally define the architectural extensions for episodic capabilities. We then demonstrate the extended system's capability to learn planning operators using the episodic traces from two Minecraft-like scenarios.

1 Introduction

Learning is of central importance to intelligent agents. From the beginning of artificial intelligence back in 1950's, researchers have recognized that the learning process is intimately tied to the nature of intelligence (Simon 1980). In order to adapt to dynamic environments, intelligent agents must possess mechanisms that allow them to acquire a broad repertoire of relevant behaviors. For this reason, there has been a significant amount of research on learning domain models in a variety of manners. But we rarely find any theories that provide a complete account of how experiences are gathered and how knowledge is derived from such experiences over time.

Our research aims to provide an infrastructure for organizing and processing collected experience, which then establishes a foundation for an experiential learning in intelligent agents. We model human *episodic* capabilities (Tulving 1983) in the context of a cognitive architecture, ICARUS (Langley and Choi 2006), and attempt to bridge these capabilities with other learning modalities. In this paper, we begin our study with the experiential learning of planning operators including action and event models. This will produce agents capable of learning throughout their lives to develop low-level expertise and adapt to dynamic environments. Such agents will also be able to recover from incorrect or incomplete knowledge over time. Additionally, because ICARUS learns structured models, agents retain the advantage of explainability.

Our work is motivated by situated agents that learn in changing, dynamic environments. Certainly, robots are one kind of such agent, but this paper focuses on a simulated domain described in the next section. After a description of this illustrative domain, we review the ICARUS architecture by providing necessary definitions that contextualize the episodic extensions we describe next. Then we present some preliminary results in the domain. Finally, we will discuss related work before we conclude.

2 Illustrative Domain

To motivate our research on episodic agents and evaluate our system's capabilities, we use a simplified version of a popular open-world game, Minecraft (Johnson et al. 2016), where players attempt to survive in a continuous, dynamic world by collecting resources, forging tools, building structures, and fighting enemies. Consider a novice agent learning from an expert player who starts at the lower left corner of a room. There are resources scattered around the room and a craft desk nearby the player. The player should gather the resources to make a sword for protection, but there are zombies in this room that guard the resources. The player must be careful because she will lose health if a zombie attacks her.

The expert player starts by selecting a resource and moving north toward it. Once she is on the same row as the resource, the player moves east toward it until she is on the same column. Now the player is standing by the resource and picks up the resource to hold it. But there was a zombie in the same location, so the player's health was reduced while the player was standing there. Then she moves south and then west to the craft desk. When the player arrives there, she puts down the resource on the desk. After repeating this process several times, the expert player would have gathered all the resources necessary to build a sword and achieve its mission by crafting one. The novice observer stores in its mind all the situations the expert has encoun-

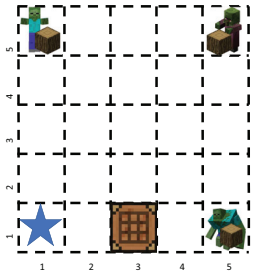

Figure 1: A 5x5 notional plot of Minicraft.

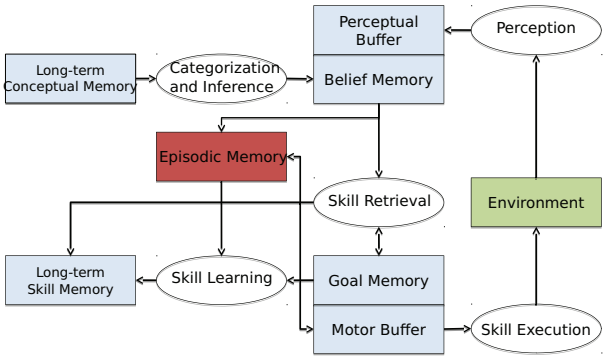

Figure 2: ICARUS cycle prior to episodic memory extension.

tered, and learns action and event models from them that it will be able to use to play the game.

We began our work by creating a grid world, *Minicraft*, that is inspired by the original Minecraft. Although simplified, this game captures enough dynamism to demonstrate the learning ability of our system. Figure 1 shows a notional view of Minicraft, which consists of four entities: resource, craftdesk, zombie, and the agent. The only entities with dynamic properties are the zombie and the agent. The agent begins at the star and moves one grid at a time while picking up or dropping resources and crafting items. Zombies, once placed on the map, are stationary, but provide dynamism to the world by decreasing the agent's health by one for every moment that the agent resides in the same grid as the zombie. All world dynamics, such as the effects of movement and action are unknown to the observer.

3 ICARUS Review

As a cognitive architecture, ICARUS provides a framework for modeling human cognition and programming intelligent agents. The architecture makes commitments to its representation of knowledge and structures, the memories that store these contents, and the processes that work over them. ICARUS shares some of these commitments with other architectures like Soar (Laird 2012) and ACT-R (Anderson and Lebiere 1998), but it also has distinct characteristics like the architectural commitment to hierarchical knowledge structures, teleoreactive execution, and goal reasoning capabilities (Choi 2011). Section 3.1 describes the key knowledge and memory structures of ICARUS, while Section 3.2 outlines how processes operate on these memories as part of a cognitive cycle.

ICARUS learns in the context of propositional states and action event models. Given a finite set of first order propositions P we define a propositional language $\mathcal{L}(P)$, and a finite set of labeled procedures, called *actions*, $\mathcal{A}$ such that $\mathcal{L}(P) \cap \mathcal{A} = \emptyset$.

3.1 Representation and Memories

ICARUS distinguishes two main types of knowledge: concepts and skills which represent semantic and procedural knowledge, respectively. Both have parameterized (i.e., lifted) variants that are grounded when variables are assigned to objects. Figure 2 shows the long-term and short-term memories of ICARUS, in which concepts and skills are stored. Paramaterized concept and skill definitions are stored in conceptual and procedural long-term memories, respectively. Instances of these definitions are stored in their respective conceptual or procedural short-term memories.

Concepts describe certain aspects of a situation in the environment. They resemble horn clauses (Horn 1951), complete with a predicate as the head, perceptual matching conditions, tests against matched variables, and references to any sub-relations.

Definition 1 (Concepts (C)) *A **primitive** concept is defined over P as $c_i = \langle \lambda, \epsilon \rangle$ where $\lambda \in P$ known as the concept head, ϵ denoting elements to pattern match in the world state S, where S is a subset of P. Let C_p be the set of primitive concepts. A **non-primitive** concept is defined over $P \cup C_p$ as $c_j = \langle \lambda, \epsilon, \gamma \rangle$ where γ denotes c_j's subrelations. We can further define non-primitive concepts over $P \cup C_p \cup C_n$, where C_n is the set of non-primitive concepts.*

Figure 3 shows example concepts for Minicraft. The first, `north-of`, is a primitive concept that describes the situation where a zombie is to the north of the agent, using perceptual matching and test conditions for *self* and *zombie*. The second, `on-horizontal-axis`, depicts a non-primitive concept where a zombie is on the same horizontal line as the agent. The third, `standing-by`, describes an even more abstract non-primitive concept where the zombie is standing right next to the agent.

Skills describe procedures to achieve certain concept instances in the environment. These are hierarchical versions of STRIPS operators (Fikes and Nilsson 1971) with a named head, perceptual matching conditions, preconditions that need to be true to execute, direct actions to perform in the

```
((north-of ?o1 ?self)
   :elements ((self ?self y ?y) (zombie ?o1 y ?y1))
   :tests ((> ?y1 ?y)))
((on-horizontal-axis ?o1 ?self)
   :elements ((self ?self) (zombie ?o1))
   :conditions ((not (north-of ?o1 ?self))
                (not (south-of ?o1 ?self))))
((standing-by ?self ?o1)
   :elements ((self ?self) (zombie ?o1))
   :conditions ((on-horizontal-axis ?o1 ?self)
                (on-vertical-axis ?o1 ?self)))
```

Figure 3: Three ICARUS concepts in the Minicraft domain.

```
((gather-resource ?o1)
   :elements ((self ?self) (resource ?o1))
   :conditions ((not (carrying ?any))
                (standing-by ?self ?o1))
   :effects ((carrying ?o1))
   :actions ((*pick-up-resource ?o1)))
((go-to ?o1)
   :elements ((self ?self))
   :conditions ((north-of ?o1 ?self))
   :subskills ((go-up-to ?o1))
   :effects ((standing-by ?self ?o1)))
((gather-resource ?o1)
   :elements ((self ?self) (resource ?o1))
   :conditions ((not (carrying ?any)))
   :subskills ((go-to ?o1) (gather-resource ?o1))
   :effects ((carrying ?o1)))
```

Figure 4: Three ICARUS skills in the Minicraft domain.

world or any sub-skills, and the intended effects of the execution.

Definition 2 (Skills (K)) *Given the finite set of actions $\mathcal{A}$, a skill defined over $C \cup S$ where C is the set of concepts and S is a propositional state, is a* primitive *skill if $k_i = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle$, where pattern match conditions $\epsilon \subseteq S$, preconditions $\gamma \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$, actions $\alpha \subseteq \mathcal{A}$, sub-skills $\sigma = \emptyset$, and effects $\eta \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$. Let K_p be the set of primitive skills.*

A skill defined over $C \cup S \cup K_p$ is a non-primitive *skill if $k_j = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle$, where $\epsilon \subseteq S, \gamma \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}, \alpha = \emptyset, \sigma \subseteq K_h$, and $\eta \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$. K_h is the set of non-primitive skills.*

Figure 4 shows example skills for Minicraft. The first, `gather-resource`, is a primitive skill that describes a procedure to collect a resource that is executable when the agent is not carrying anything and is standing next to the resource. This skill uses a direct action to pick up the resource and its intended effect is carrying the resource. The bottom two are non-primitive skills that use sub-skills: `go-to` uses a sub-skill `go-up-to` to achieve the goal of standing near the object, while `gather-resource` uses the two sub-skills above it to collect a resource.

3.2 The ICARUS Cognitive Cycle

The ICARUS architecture operates in a cognitive cycle repeating two steps: conceptual inference and skill execution. *Conceptual inference* is the process of creating concept instances (i.e., beliefs). At the beginning of each cycle, the system receives sensory input from the environment as a list of objects with their attribute-value pairs; this can be thought of as the world state and is represented as propositions. Based on this information, the architecture infers the concept instances (i.e., beliefs) that are true in the current state by matching its concept definitions to perceived objects and other concept instances in a bottom-up fashion.

In summary, Figure 2 shows concept definitions housed in the conceptual long-term memory are used to infer the beliefs of the system from the world state and are stored as concept instances in the conceptual short-term memory.

Definition 3 (Beliefs (B)) *Let C be the set of concepts. $\forall c = \langle \lambda, \epsilon, \gamma, \tau \rangle \in C$, $\exists$* belief *$b = \langle \lambda, \epsilon, \gamma, \tau, \beta \rangle$ where β represents bindings that ground b on the perceptual elements, ϵ. Let B be the set of all possible beliefs, and let $\mathcal{B} = 2^B$ be the set of all* belief states. *A* belief state *$s \in \mathcal{B}$.*

Skill execution proceeds after conceptual inference whereby ICARUS finds all the relevant skill definitions for the current goal(s) that are executable based on the current beliefs. ICARUS chooses a skill and sets it as its *intention* and executes it in the world.

Definition 4 (Intentions (ι)) *Let K be the set of skills. $\forall k = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle \in K$, there exists* intention *$\iota = \langle \epsilon, \gamma, \alpha, \sigma, \eta, \beta \rangle$ where β represents bindings that ground ι in the belief state.*

Each cycle may introduce changes in the environment, which may modify the sensory input for the next cycle, resulting in new beliefs and intentions. The architecture iterates in this manner until all of its goals are achieved or its operations are terminated for any other reasons.

4 Constructing Episodes

We now shift our attention to extending ICARUS with an episodic memory. In particular, we highlight the core data structures of ICARUS's Episodic Memory (Section 4.1), how it encodes episodes within that memory through a process called event segmentation (Section 4.2), and how it generalizes episodes over time (Section 4.3).

4.1 The Episodic Memory

The episodic memory in ICARUS is a long-term, cue-based memory that the agent uses to deliberately encode and retrieve episodes. The architecture organizes its episodic memory $E = \langle \rho, \mathcal{F}, \mathcal{T} \rangle$ in a compound structure composed of an episodic beliefs-action cache ρ, a concept frequency forest $\mathcal{F}$, and the episodic generalization tree $\mathcal{T}$.

Figure 5 shows how information is processed within the episodic memory and is discussed through this section. ρ acts as a storage for the agent's unprocessed history. We assume that the agent has sufficient memory to store the complete beliefs-action sequence. $\mathcal{F}$ records counts for the number of times concepts and their instantiations as beliefs have

occurred during the execution of the agent. $\mathcal{T}$ is the main data structure that organizes and stores episodes; the contents of $\mathcal{T}$ are used in the process of learning new skills. The elements ρ and $\mathcal{F}$ (Definitions 5 and 6), discussed next, facilitate the workings of the event segmentation and episodic encoding (Section 4.2). Generalization with $\mathcal{T}$ is discussed in Section 4.3.

Since episodes are built on top of sequences of beliefs, we introduce first the beliefs-action cache, which stores the moment-by-moment changes in belief, inferred from the world state, as well as the actions that were taken based on those beliefs.

Definition 5 (Beliefs-action cache (ρ)) *The* beliefs-action cache *ρ, is an ordered sequence of belief-action pairs. This cache stores a complete, detailed history of what the agent observed. Figure 5 shows that the contents of the belief memory are inputs to the beliefs-action cache.*

Once these traces are collected, they must be processed for *interesting* events, which are tracked in the concept frequency forest.

Definition 6 (Concept frequency forest ($\mathcal{F}$)) *Let X be a set of location predicates, and let $Y = \{x.first | x \in S\}$ be the set of object types. A* concept frequency tree *is a tree whose the root μ is a location predicate from X. The children of μ are all the concepts the agent has observed in that location. For each child concept, c, of μ, $\exists$ a set of types from Y, to specify concept disjunctions. Under each disjunction, j, there exists concept instances. Each node in the tree has a count field, denoting the number of times this node has been observed. A* concept frequency forest *is a collection of concept frequency trees.*

ICARUS uses $\mathcal{F}$ to model *expectation violation*. The agent sets two thresholds: one for positive expectations and one for negated expectations. Any belief with a conditional probability, given the location, is greater than the positive threshold is said to be *expected*. Any belief with a conditional probability, given the location, is less than the negated threshold is *not expected* to be in the state. A belief that violates an expectation is a *significant belief*, which prompt the system to create an episode. This is a primitive method for novelty detection that only uses spatial information, but we can further extend the novelty detection method to include the temporal domain as well.

The episode structure defined in Definition 7 represents the agent's experiences in the architecture. Once they are stored in memory, episodes are processed to abstract general rules that allow the agent to predict environmental dynamics.

Definition 7 (Episode (ε)) *An episode is a tuple $\langle B_s, B_e, \Sigma, \psi \rangle$, where B_s is the start state of the episode, B_e is the end state of the episode, Σ is the set of significant beliefs in B_e, and ψ is a count for the number of times the episode has occurred.*

During episodic encoding, the start and final states are taken from the ρ (i.e., the beliefs-action cache). In the current implementation, B_s and B_e are consecutive belief states, but our work does not require this. Our rationale is psychologically inspired. When humans perform low-level actions, kicking a soccer ball for instance, humans know that the effect is not always observed in their next cognitive cycle. The ball travels in time before it reaches the goal. This dynamic is readily understood by most humans. Modeling actions with temporally delayed effects is part of our future work.

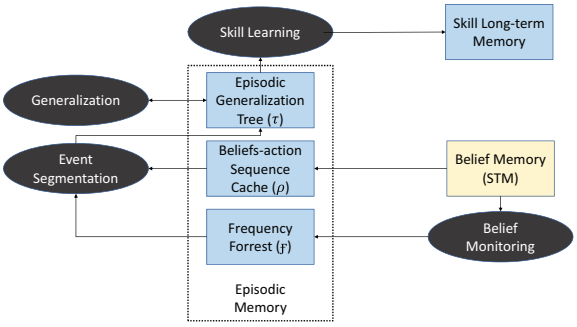

Figure 5: Block diagram depicting episodic memory components and information flow starting from the belief memory.

4.2 Episodic Encoding

Episodic Encoding in ICARUS is a two-step process. First, ICARUS operates on the ρ to returns a new episode ε. This is referred to as "Event Segmentation" in Figure 5. Once the episode exists, the second process places it into the episodic generalization tree. Algorithm 1 shows that encoding is triggered by the presence of one or more significant beliefs in belief state.

Algorithm 2 traces how episodes are inserted into the episodic generalization tree. Suppose the generalization tree contains several episodes. Γ is a list of sibling episodes under parent $\varrho \in \mathcal{T}$ If $\forall \varepsilon_i \in \Gamma, (\varepsilon_i, \varepsilon) \notin E$ then $(\varrho, \varepsilon) \in E$. That is ε becomes a child of ϱ. A new episode has successfully been encoded into the episodic memory. If $\exists \varepsilon_j \ni \varepsilon_j = \varepsilon$, then the counter for ε_j increments by one and ε is not inserted.

On every cycle, ICARUS records the belief state and executed actions into the episodic cache and updates $\mathcal{F}$. When the agent infers one or more significant beliefs, it encodes

Algorithm 1 CREATEEPISODE(ρ, loc, B_c)

```
ρ is beliefs-action cache
loc is current location
B_c is current belief state
B_prev ← last state in ρ
ρ ← ρ.add(B_c, a)
sigs ← GETSIGNIFICANTBELIEFS(B_c, loc)
if not NULL(sigs) then
    ε ← MAKEEPISODE(sigs, B_c, B_prev)
    T ← INSERT(ε, T)
```

Algorithm 2 INSERTEPISODE($\varepsilon, \mathcal{T}$)

```
queue ← ∅
temp ← root of 𝒯
match ← ∅
p ← ∅
while not NULL(temp) do
    match ← STRUCTURALEQ?(temp, ε)
    if match is exact match then
        temp.count ← temp.count + 1
        Try to learn from temp if count high enough
        BREAK
    else if match is bc of unification then
        temp.count ← temp.count + 1
        queue ← ∅
        queue ← temp's children
        Try to learn from temp if count high enough
        p ← temp
    temp ← queue.FIRST
    queue ← queue.POP
if null(temp) and match not exact then
    p ← p.ADDCHILD(ε)
    𝒯 ← GENERALIZE(p, ε)
```

a new episode. The root node of the generalization tree is the most general episode and is allowed to have an arbitrary number of children. Under the root, episodes are grouped according to *structural similarity*. Two episodes e_1, e_2 are *structurally similar* if their significant beliefs unify. By "unify" we mean that there must exist a binding set that transforms the significant beliefs of e_1 to those of e_2 and vise versa. This is a rigid generalization scheme that needs more consideration in future work. Each child is a *k*-ary tree where $k \in \mathbb{N}$. Episodes become more specific at each decreasing level of the tree according to structural similarity. At the leaf nodes exist fully instantiated episodes.

4.3 Episodic Generalization

ICARUS supports generalization of the episodic tree during encoding of episode, ε_i. Definition 8 shows that an episode hierarchy is induced by structural similarity. Two sibling episodes $\varepsilon_i, \varepsilon_j$ generalize iff $\exists$ episode ε_g such that $(\varepsilon_g, \varepsilon_i) \in E$ and $(\varepsilon_g, \varepsilon_j) \in E$, but $(\varepsilon_i, \varepsilon_g) \notin E$ and $(\varepsilon_j, \varepsilon_g) \notin E$. This means that ε_g unifies with its children, $\varepsilon_i, \varepsilon_j$, but its children cannot unify with it because they contain more specified bindings. If ε_g exists, ICARUS tests to see if it is still more specific than the parent of ε_i. If so, then ε_g's parent becomes ε_i's parent and ε_g's children become $\varepsilon_i, \varepsilon_j$. The count for a generalized episode is the summation of the count of its children.

Definition 8 (Generalization tree ($\mathcal{T}$)) *An* episodic generalization tree *is a tree (V, E) where V is a set of episodes, and E is a set of edges. For any $\varepsilon_i, \varepsilon_j \in V, (v_i, v_j) \in E$ if they are structurally similar. An episode is said to be generalized or* partially instantiated *if the bindings contain one or more unbound variables.*

The generalization tree naturally lends itself to the learning process as a result of generalization. For example, if person x drops a glass on the ground and it breaks, and person y drops a glass on the ground and it breaks as well, ICARUS forms a generalized episode that implies if anyone drops a glass on the ground, it will break. The ability to gain knowledge in this way is central to general intelligence. As the tree adds more episodes, they are sorted into increasingly sensible taxonomies. The resulting tree after insertion is ICARUS' best estimate of the ideal generalization tree. This organizational structure was inspired by the incremental concept formation literature (Gennari, Langley, and Fisher 1989). As episodes become more general, the skills ICARUS learns from those episodes are equivalently general. So, generalizing skills is performed within the episodic generalization tree, not the skill learning algorithm.

5 Skill Learning using the Episodic Memory

In previous work, ICARUS supported learning by observing problem solving traces that include goals, conditions, and the skills used (Nejati 2011). The system relied on the explanations it generated based on the given trace, and this process required, at the very least, primitive skills in ICARUS' memory. In the current work, we start with only the concepts that are sufficient to describe situations in the world but the agent does not have any skills in its knowledge base.

ICARUS starts as an observer and records the history of belief states and ground actions in its episodic memory. As its experience accumulates, the agent will insert an episode whose count surpasses a predefined threshold for model learning. At that moment, the system uses the actions from $B_s \rightarrow B_e$ as a search cue for collecting other episodes where that ordering of actions took place. This trace of episodes is then used in the rule induction algorithm, MLEM2 (Grzymala-Busse and Rzasa 2010). Although we are using MLEM2, this need not be the case. Any rule learning algorithm may be used as long as there is a transformation from ICARUS's representation of experience to the representation that the learning algorithm requires. After learning, the agent can seamlessly utilize the learned skills during problem solving.

5.1 Learning Action and Event Models

In order to learn models of the world, ICARUS must first retrieve experiences via a retrieval cue. The system generates an observation, as defined in Definition 9 for each episode that matches the cue. For the case of model learning, the retrieval cue is some subset of actions a_i from $\mathcal{A}$. As the episodes are examined, matches are collected into an episodic trace of evidence related to a_i.

Definition 9 (Observations (O)) *Let $o = \langle s_i, a_i, s_f \rangle$ be an* observation *from ρ, the beliefs-action cache, where $s_i, s_f \in \varsigma$ are respectively initial and final belief states, and $a_i \subseteq \Lambda$ be the set of actions that transformed s_i to s_f An* episodic trace*, O is a collection of observations.*

MLEM2 learns rules from data tables, therefore, once the episodic trace is obtained it needs to be transform O into a table. The x-axis for this table is an enumeration of all the

Belief	ℓ_b	$\ell_b \cap \{1,3,4\}$
(holding sword1)	{1,2,3,4,5}	{1,3,4}
(holding nothing)	{6}	$\emptyset$
(holding food1)	{7,8}	$\emptyset$
(next-to ?zombie)	{1,3,4,7}	{1,3,4}
(next-to tree1)	{5,2, 6,8}	$\emptyset$
(health good)	{1,2,3,4,5,6,7,8}	{1,3,4}

Table 1: Sample attribute and decision blocks.

unique beliefs in O, and the y-axis numbers each observation in O. Each belief, b on the x-axis has an associated list , $block_b = \{i | \langle s_j, a_j, s_k \rangle \in O[i], b \in s_j\}$ of the observation indices it appeared in. The last column of the data table is the list of the effects, fx for each associated observation. Table 1 summarizes the data table in a way that clearly shows each belief's block list. For example, the middle column states for the first row, that the (holding sword1) belief was present in observations 1 through 5.

For each effect, f in fx, the algorithm computes a list, $block_{fx} = \{i | \langle s_j, a_j, s_k \rangle \in O[i], f \in s_k\}$ of observation indices that it appeared in as well. MLEM2 tries to find, for each effect, conditions whose associated blocks cover the effect block. These coverings are what are the learned action and event models.

In this example, assume $a_i = ((*attack))$, and $fx = \{((zombie\text{-}dead\ ?zombie), \{1,3,4\}), ((wood\ wood1), \{2\})\}$.1 MLEM2 attempts to find local coverings of fx from the list of belief conditions. MLEM2 tries the pair $(b, block_b)$ whose listing, $block_b$ intersected with an uncovered effect $block^0_{fx} = \{1,3,4\}$ is the largest. If $block_b \leq block^0_{fx}$, then that condition becomes a rule that covers that effect. If $block_c \not\leq block^0_{fx}$ then other conditions need to be added to cover it. Once a rule has been found that covers all the cases of for an effect, the same process repeats for the uncovered effects in fx. In the example, the system learns the following rule: (next-to ?zombie) $\cap$ (holding sword) $\rightarrow$ (zombie-dead ?zombie).

In the ICARUS context, MLEM2 results are converted to action and event models, which are primitive skills. The left hand side of the rules become the preconditions, the right hand side would be the effects of the skill. The action information would capture what work needs to be done to realize the effects.

6 Experimental Setup

The goal with this research was to create an agent that could learn unknown domain dynamics from experience. Furthermore, we want a system that is flexible and continues learning over the course of its life to reflect the changes in the world's changing dynamics. We assume that the world is fully observable, and that the agent has a vocabulary that distinguishes belief states perfectly. Also, we assume effects come immediately after actions, and that the environment is not stochastic.

We tested on two scenarios. Each scenario has one expert with perfect concept and skill knowledge, and one observer with full observability of the state, perfect concept knowledge, but no skill knowledge (i.e., no knowledge of the domain dynamics). We are primarily interested in what action and event models the agent learns and know how they change in response to new evidence. In the first scenario, we place the expert at (1,1), and zombies and resources are at the other three corners. At (5, 1) there exists a craftdesk. The expert is tasked with collecting resources and placing them on the craftdesk. For the case of the expert, this problem is easily solved, but for the novice, we are interested in how well it learns the dynamics of the world. An example of an event model would be knowing that being next to a zombie reduces the agent's health, and an example of an action model would be learning about what happens to the state when the agent moves.

The second scenario extends the first with the zombies and resources have been randomly re-assigned to different corners. This makes for two different, but structurally identical scenarios. By doing this, we ensure that the agent constructs episodes that will generalize with the other episodes in its memory.

```
(achieve-bottom-horizontal-axis-and-more)
   :conditions ((at minicraft) (north-of r1 me)
               (north-of r3 me) (east-of r2 me)
               (east-of r3 me) (east-of craftdesk1 me)
               (north-of zombie2 me) (north-of zombie3 me)
               (east-of zombie1 me) (east-of zombie2 me)
               (good-health me) (on-ground r1)
               (on-ground r2) (on-ground r3)
               (on-vertical-axis r1 me)
               (on-vertical-axis zombie3 me)
               (on-horizontal-axis zombie1 me)
               (on-horizontal-axis craftdesk1 me)
               (on-horizontal-axis r2 me))
   :actions ((*move-up))
   :effects ((south-of ?r3 me) (south-of craftdesk1 me)
               (south-of ?zombie3 me)
               (bottom-of-horizontal ?zombie3)
               (bottom-of-horizontal ?r3)
               (bottom-of-horizontal craftdesk1))

(achieve-bottom-horizontal-axis-and-more)
   :conditions ((on-horizontal-axis ?r3 me)
                  (on-horizontal-axis craftdesk1 me)
                  (on-horizontal-axis ?zombie3 me))
   :actions ((*move-up))
   :effects ((south-of ?r3 me) (south-of craftdesk1 me)
               (south-of ?zombie3 me)
               (bottom-of-horizontal ?zombie3)
               (bottom-of-horizontal ?r3)
               (bottom-of-horizontal craftdesk1))
```

Figure 6: Learned action models for the `*move-up` action before (top) and after (bottom) generalization.

7 Results

We demonstrate that the agent is able to learn goal-directed, specific or generalized action and event models from experience. Because of the episodic memory, ICARUS agents have a mechanism for experiential learning which allows them to learn world dynamics in the form of ICARUS skills. The learned skills are continually revised according to evidence.

Figure 6 demonstrates how the action model for moving

```
(achieve-fair-health-and-more)
   :conditions ((good-health me) (on-ground r1)
                (healthy-standing-by zombie3))
   :actions (nil)
   :effects ((fair-health me) (slouching-by me zombie3))
```

```
(achieve-fair-health-and-more)
   :conditions ((good-health me)
                (healthy-standing-by ?zombie2))
   :actions (nil)
   :effects ((fair-health me) (slouching-by me ?zombie2))
```

Figure 7: Action models for the event model before learning (top) and after learning (bottom).

up changes with experience. The initial action model in Figure 6 (top) contains many irrelevant conditions, while the final version (bottom) contains no irrelevant conditions; not shown are intermediate versions. The same is true for the event model the agent learns for achieving (fair health). Figure 7 shows that the irrelevant condition is removed from the event model by the last refinement, where the event model also successfully generalizes the initial version (top) to the final version (bottom).

In our framework the system learns models based on the agent's interpretation of the ground truth. This is interesting because it clarifies certain properties of inference. Specifically, if an agent is lacking conceptual vocabulary to describe situations, its learned models will show evidence of stochasm. In other words, there will be cases where the same action occurred in identical belief states resulting in different effects.

8 Related Work

Earlier research in action recognition and learning aims to teach robots to recognize and perform human gestures (Yang, Xu, and Chen 1997). In that work the researchers used a discrete hidden Markov model to decode human intentions, and to learn the motor actions that controlled making gestures. Along this line, Liu et al. (2017) recently developed a multi-task learning system that hierarchically recognizes human actions. Also, another recent approach attempted to learn control policies for continuous, non-Gaussian stochastic domains (Wang et al. 2017). The work describes a reinforcement learning system that learns an incomplete policy for a discrete controller. Given the policy, a robot executes the action for the nearest state to the current one.

The main distinction from our work and these is that they do not learn action models in the way that we have defined them. The action models these systems learn are often limited to scenario-specific transition functions, and control policies. The semantic meaning of actions, however is still unknown to the agent, so planning with the notion of explicit goals is not possible. Moreover, when these system refer to action models they typically refer to modeling the human motor controls that produce gestures.

In addition to machine learning, researchers are also trying to learn operator descriptions that can be used in performance systems. As Langley and Simon point out, our goal is to understand and characterize the invariants of intelligence. Building systems that help explain how novices become experts in general is key to this endeavor. Wang et al. (1994) created a system built on PRODIGY (Carbonell et al. 1991) that incrementally learned planning operators based on STRIPS (Fikes and Nilsson 1971) via observation and practice. Expert demonstrations allowed the system to estimate initial versions of the operators. The agent refined its knowledge base by attempting to use learned operators to solve problems. The system was able to learn subgoal orderings for the operators, but the system could not learn operator decompositions, so operators were learned and stored in a flat structure. Gil et al. (1994) discussed how imperfections in domain knowledge do not always lead to planning or execution failures. They also presented a system that learns to refine imperfect operators by experimenting. The experimentation process can refine both operator pre and post conditions.

Another system, ALPINE provided methods for inducing abstraction hierarchies over operators (Knoblock 1990). Given a set of low-level operators, the system could induce abstraction hierarchies that reduced the search space.

Another interesting approach learned operators with associated numeric attributes to denote the utility of a particular operator (García-Martínez and Borrajo 2000). In this way the system favored more accurate operators. Walsh and Littman (2008) addressed the problem of efficiently learning STRIPS-like operators via experience. They define their own notion of an episode to be an initial state, s_0 goal state, and all state-action pairs following s_0 until the problem is solved or marked unsolvable. Their notion of episode, however, is not tied to a larger theory of episodic memory.

Lastly,Molineaux and Aha (2014) describe a surprise-driven method for learning event models. Given a problem, the system returned a plan of actions that would achieve the goal as well as a sequence of expected state changes caused by executing those actions. The system notices surprises when discrepancies exist between actual and expected state transitions. Discrepancies trigger an explanation module, DISCOVERHISTORY to hypothesize the cause of the discrepancies. When explanations fail, the system uses a variant of FOIL to learn an action model that repairs broken explanations.

In our work we addressed the problem of model learning from the vantage point of episodic memory for intelligent agents. Other research has investigated episodic memory. In the work most similar to ours, Nuxoll and Laird (2007) extended the Soar architecture (Laird, Newell, and Rosenbloom 1987) with episodic memory. They present results for action modeling in their work, but details about the learning mechanism are left out. There are also significant theoretical differences between the episodic memory in ICARUS and Soar. ICARUS has strong commitments to hierarchical organization of knowledge throughout the architecture, which helps support our theory for incremental learning. Soar, although it has had many successes, does not have such strict commitments to hierarchy. In their architecture episodes are stored in a flat container for experiences. Moreover, episodes

in ICARUS have temporal components, meaning that they contain a sequence of states, whereas Soar's episodes do not have any temporal dimension.

9 Conclusion

We presented a new extension to the ICARUS architecture that allows agents to learn goal-directed planning operators from episodic traces. Our results from the Minicraft domain showed that our theory incrementally learns skills in a specific-to-general manner, and also refines skills based on evidence. This evidence is collected from ICARUS episodic memory, a dedicated facility for constructing, storing and organizing experience.

Acknowledgments

The research presented in this paper was supported in part by the Naval Research Laboratory under contract number N00173-17-P-0837 and the Naval Research Enterprise Internship Program.

References

Anderson, J. R., and Lebiere, C. 1998. *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Carbonell, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Knoblock, C.; Minton, S.; and Veloso, M. 1991. Prodigy: An integrated architecture for planning and learning. *ACM SIGART Bulletin* 2(4):51–55.

Choi, D. 2011. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* 12:293–308.

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

García-Martínez, R., and Borrajo, D. 2000. An integrated approach of learning, planning, and execution. *Journal of Intelligent and Robotic Systems* 29(1):47–78.

Gennari, J. H.; Langley, P.; and Fisher, D. 1989. Models of incremental concept formation. *Artificial intelligence* 40(1-3):11–61.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *International Conference on Machine Learning*, 87–95.

Grzymala-Busse, J. W., and Rzasa, W. 2010. A local version of the mlem2 algorithm for rule induction. *Fundamenta Informaticae* 100(1-4):99–116.

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

Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 4246–4247.

Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In *AAAI*, 923–928.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1):1–64.

Laird, J. E. 2012. *The Soar Cognitive Architecture*. Cambridge, MA: MIT Press.

Langley, P., and Choi, D. 2006. A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*.

Langley, P. 1983. Learning search strategies through discrimination. *International Journal of Man-Machine Studies* 18(6):513–541.

Liu, A.-A.; Su, Y.-T.; Nie, W.-Z.; and Kankanhalli, M. 2017. Hierarchical clustering multi-task learning for joint human action grouping and recognition. *IEEE transactions on pattern analysis and machine intelligence* 39(1):102–114.

Molineaux, M., and Aha, D. W. 2014. Learning unknown event models. In *AAAI*, 395–401.

Nejati, N. 2011. *Analytical Goal-Driven Learning of Procedural Knowledge by Observation*. Ph.D. Dissertation, Stanford University.

Nuxoll, A. M., and Laird, J. E. 2007. Extending cognitive architecture with episodic memory. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 1560–1565.

Simon, H. A. 1980. Cognitive science: The newest science of the artificial. *Cognitive science* 4(1):33–46.

Tulving, E. 1983. Elements of episodic memory.

Walsh, T. J., and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *AAAI*, volume 8, 714–719.

Wang, Z.; Jegelka, S.; Kaelbling, L. P.; and Lozano-Pérez, T. 2017. Focused model-learning and planning for non-gaussian continuous state-action systems. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 3754–3761. IEEE.

Wang, X. 1994. Learning planning operators by observation and practice. In *AAAI*, 335–340.

Yang, J.; Xu, Y.; and Chen, C. S. 1997. Human action learning via hidden markov model. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27(1):34–44.