ARTICLE IN PRESS

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Cognitive Systems Research xxx (2011) xxx–xxx

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

An intrinsically-motivated schema mechanism to model and simulate emergent cognition

Action Editor: Ron Sun

Olivier L. Georgeon[a,b,*], Frank E. Ritter[b]

[a] Université de Lyon, CNRS, LIRIS, UMR5205, F-69622, France
[b] The Pennsylvania State University, University Park, PA 16802, USA

Received 4 April 2011; accepted 30 May 2011

Abstract

We introduce an approach to simulate the early mechanisms of emergent cognition based on theories of enactive cognition and on constructivist epistemology. The agent has intrinsic motivations implemented as inborn proclivities that drive the agent in a proactive way. Following these drives, the agent autonomously learns regularities afforded by the environment, and hierarchical sequences of behaviors adapted to these regularities. The agent represents its current situation in terms of perceived affordances that develop through the agent's experience. This situational representation works as an emerging situation awareness that is grounded in the agent's interaction with its environment and that in turn generates expectations and activates adapted behaviors. Through its activity and these aspects of behavior (behavioral proclivity, situation awareness, and hierarchical sequential learning), the agent starts to exhibit emergent sensibility, intrinsic motivation, and autonomous learning. Following theories of cognitive development, we argue that this initial autonomous mechanism provides a basis for implementing autonomously developing cognitive systems.
© 2011 Elsevier B.V. All rights reserved.

Keywords: Motivation; Autonomous learning; Cognitive development; Enactive cognition; Affordances; Constructivism; Cognitive architecture

1. Introduction

We introduce a model that simultaneously addresses three issues regarding cognition: intrinsic motivation, autonomously constructed internal state, and adaptive learning. The first issue, intrinsic motivation, is the question of implementing a system whose behavior is driven by inward forces, *impetus*, or *proclivity* (e.g., Blank, Kumar, Meeden, & Marshall, 2005; Oudeyer & Kaplan, 2007). Intrinsically motivated behavior contrasts with extrinsically motivated behavior that consists of performing a task, seeking a goal, or solving a problem as prescribed or defined by an external person.

The second issue is the question of autonomously constructing an internal state that reflects the agent's situation. Such an internal situational state allows the agent to adapt its behavior to the current context. In cognitive science, the agent's situational state is generally referred to as the agent's *representation of the situation* or, in human factors, *situation awareness* (Endsley, 1995). The term *perception* may also be used insofar as perception is taken as a cognitive construct rather than simple data received from the environment (e.g., Bajcsy, 1988). If autonomously constructed, the internal situational state reflects how the agent experiences the situation rather than how the designer models the situation. In other words, we expect an autonomous situational state to rest upon minimal ontological commitments made by the designer about the environment. Moreover, the designer should neither predefine the semantics of such situational states nor implement these semantics in the form of a reasoning mechanism based

* Corresponding author at: Université de Lyon, CNRS, LIRIS, UMR5205, F-69622, France.
E-mail address: olivier.georgeon@liris.cnrs.fr (O.L. Georgeon).

1389-0417/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cogsys.2011.07.003

ARTICLE IN PRESS

on a predefined ontology. Many authors (e.g., Dennett, 1991; Hurley, 1998; Pfeifer, 1996) have agued against cognitive modeling based on a predefined ontology. In essence, these arguments prompt us not to implement a "Cartesian theater" (Dennett, 1991) where the representation would be interpreted by a "homunculus" (Pfeifer, 1996). Instead, the internal state should be *directly operative* by being interwoven with the motivational mechanism that gives rise to behavior (Weill-Fassina, Rabardel, & Dubois, 1993).

The third issue is the question of adaptive learning through interaction. Most studies in adaptive learning distinguish between a learning phase where the knowledge is acquired, and a performance phase where the learning is assessed. Aha (1998) highlights two categories of learning algorithms according to how the computation is balanced across these two phases: eager learning and lazy learning. Eager learning algorithms compile input samples and use only the compilation to make decisions (e.g., reinforcement learning). Lazy learning algorithms perform little compilation and reuse the stored input samples to make decisions (e.g., schema mechanisms, case-based-reasoning). In agreement with Aha (1998), Wolpert and Macready (1997) argued that "there is no free lunch" in unsupervised learning algorithms: what is gained in performance is lost in generality, and vice versa. In this work, we investigate an autonomous developmental approach where the system gradually learns to process its input through its activity. In this case, learning is intertwined with performance. We expect the system to begin with lazy learning (making minimal assumptions about the environment), then gradually trade generality in favor of performance through the system's development.

This work investigates the hypothesis that these three issues (intrinsic motivation, self created state, adaptive learning) are intertwined, and that a system addressing them simultaneously will exhibit emergent cognition. The intuition behind this hypothesis is that the system's intrinsic motivation would provide criteria for autonomously assessing the system's learning; the system's autonomous learning would provide an evolving way to autonomously encode the agent's internal situational state; and the autonomous internal situational state, in turn, would offer a way to aggregate the agent's experience into knowledge suited to the agent's motivation. We named this hypothesis the triple-autonomy hypothesis: motivational autonomy, representational autonomy, learning autonomy.

With the triple-autonomy hypothesis, we want to contribute to an ongoing debate on the notion of autonomy in the cognitive sciences (e.g., Froese, Virgo, & Izquierto, 2007). Indeed, one could argue that even natural cognitive organisms are not fully autonomous because they incorporate cognitive biases that implement knowledge gained through phylogenetic evolution. In our case, we limit the knowledge pre-implemented in the agent to two aspects: (a) we predefine the *possibilities of interaction* that the agent has with its environment in the form of valued primitive interactions; (b) we implement a mechanism capable of learning and exploiting hierarchical sequences of interactions. We posit that the triple-autonomy hypothesis would be falsified if these kinds of prerequisites proved insufficient to implement agents that demonstrate emergent cognition. We nonetheless acknowledge that other innate biases might be needed to facilitate higher-level developmental stages, for example, to learn spatial regularities or object permanency.

With emergent cognition, we refer to cognitive development *ab-nihilo*—sometimes called *bootstrapping cognition* (e.g., Dennett, 1998). Different developmental theories have identified various stages in the cognitive development process. We retain here Piaget's (1937) idea of a sensorimotor earliest stage that precedes the discovery of persistent objects in the world and underpins later symbolic thought upon such objects. For the present work, though, the earliest stage can also fit the framework of phylogenetic evolution of animal cognition, as discussed for example by Sun (2004). Because such early stage mechanisms focus primarily on behavior organization rather than cognitive states, these mechanisms can also be related to the situated (e.g., Hutchins, 1995; Suchman, 1987), embodied (e.g., Varela, Thompson, & Rosch, 1991), and enactive (e.g., Noë, 2004; Stewart, Gapenne, & Di Paolo, 2008) theories of cognition. For these authors, early stage cognitive phenomena generally include processes such as learning and knowing something about how to interact with the environment, demonstrating emergent preferences and awareness of the situation, and acquiring sequential behaviors capable of generating anticipation of the consequences of actions. More specifically, enactive theories suggest a developmental design principle according to which, "as a result from action, the agent's perception of its environment may be altered in such a way that [the agent] will never again perceive that environment in the same way" (De Loor, Manac'h, & Tisseau, 2010, p. 330).

We report here a model as a proof-of-concept based on the triple-autonomy hypothesis that exhibits emergent cognition. We named this model the *intrinsically-motivated schema mechanism*. In its current state, we should be clear that we do not expect this model to account for symbolic reasoning or reflecting as it is done in human-like intelligence. Our motivation for this work comes from our belief that studying such early mechanisms can open the way to implementing autonomous higher-level intelligence, but, for the moment, we can only rely on arguments proposed by psychologists or philosophers to support this claim (e.g., Dennett, 1991; Piaget, 1970). Yet, we expect this algorithm to bring insights with regards to these theories, especially by informing our understanding of how agents can autonomously construct emergent representations of the situation that are grounded in the agent's activity (Harnad, 1990).

2. Implementation background

Our intrinsically-motivated schema mechanism mostly rests upon three different mechanisms of machine learning and cognitive modeling. The first mechanism is a mechanism of reinforcement learning based on intrinsic reward (Singh, Barto, & Chentanez, 2005). The second mechanism is a *hierarchical schema mechanism* inspired from Piagetian (1970) constructivist epistemology and from schema-based approaches to cognition (e.g., Arbib, 1992; Rumelhart & Norman, 1981) and hierarchy in systems (Simon, 1981). The third mechanism is a mechanism of episodic memory inspired by trace-based reasoning (Cordier, Mascret, & Mille, 2009; Mille, 2006). Besides these three sources of inspiration, the algorithm also implements an original view of the agent's inward drives that define the agent's intrinsic motivation.

The algorithm includes a notion of reinforcement learning in that the agent discovers the environment's structure through trial and error. Behaviors are weighted and their weights are incremented when the behavior is enacted in the environment. This weight operates as reinforcement in that the agent selects the most strongly weighted behaviors. The algorithm, however, differs from classical reinforcement learning in that the learning does not come from a reward that is given when the agent reaches a goal, with the reward subsequently backward-propagated to previous states. By definition, an intrinsically motivated agent has no pre-implemented mechanism that detects final goal achievement, nor is the agent exploring a pre-defined *problem space*. Therefore, the agent cannot attribute the classical *utility values* to transitions within such a problem space.

To report our agent's mechanism, we define the notion of *proclivity value* in correspondence to the notion of *utility value* in traditional reinforcement learning mechanisms. Subjectively, the notion of proclivity value corresponds to the agent's *intrinsic satisfaction* in an equivalent way as the notion of utility value corresponds to reward. The nuance resides in that intrinsic satisfaction merely comes from enacting the behavior while reward comes from the outcome of the behavior. To an external observer, our agent seems to *enjoy* enacting behaviors that have a positive proclivity value, and to *dislike* enacting behaviors that have a negative proclivity value, regardless of the behavior's outcomes. With our algorithm, nonetheless, the agent can learn to choose unsatisfying behaviors (negative proclivity values) in one context to reach other contexts where the agent can enact behaviors that are even more satisfying (positive proclivity values). In the rest of this paper, we use the terms *proclivity value* and *satisfaction value* equivalently, the former term corresponding to a programming perspective and the latter to an explanatory perspective. Overall, this learning mechanism results in increasing the agent's average satisfaction, that is, improving the capacity of the agent to enact the behaviors that have the highest proclivity.

The model also is related to Piaget's (1970) constructivist epistemology. Piaget's foundational intuition is that the distinction between the inner self and the external world is not innate but is learned by the subject: "Intelligence (and therefore knowledge) begins not with the knowledge of the self, nor with the knowledge of things as such, but with the knowledge of their interaction; intelligence organizes the world while organizing itself by simultaneously considering the two poles of this interaction".[1] Following this intuition, Piaget suggests that atomic elements of cognition are not symbols that represent individual things but schemes that represent individual interactions.

Piaget's theories have inspired a range of computer implementations called *schema mechanisms* (e.g., Arkin, 1987; Chaput, 2004; Drescher, 1991; Guerin & McKenzie, 2008; Holmes & Isbell, 2005; Perotto, Buisson, & Alvares, 2007; Stojanov, Bozinovski, & Trajkivski, 1997). These authors implemented schemes as triplets [perception$_1$, action, perception$_2$] and referred to them with the term *schema*. The agent randomly explores its environment and records schemas that mean that a certain action in a certain perceptual state (perception$_1$) would likely result in a certain perceptual outcome (perception$_2$). These authors, however, noted that this approach leads to a combinatorial explosion when the environment's complexity grows. We believe that this approach based on triplets diverges from Piaget's original views, in that this implementation of schemas presupposes the agent's perception of the world, whereas Piaget considered perception of the world as a construct arising from interaction.

In our work, we address the scalability issues of current schema mechanisms in three ways. First, we do not include the perception of the environment in our schemes but rather define the scheme's context in terms of interactions. Our schemes are modeled as a couple [interaction$_1$, interaction$_2$], meaning that, in the context of interaction$_1$, the agent can enact interaction$_2$. Describing the context in terms of interaction means that the agent learns to "see" its world in terms of *affordances* (Gibson, 1979) related to its own prior experience. We follow Gibson's definition of affordances as possibilities of interaction afforded to the agent by the environment. With this formalism, schemes natively encode entire sequences of interactions in a hierarchical fashion. To highlight this radical difference from classical schema mechanisms, we choose to keep Piaget's term *scheme* rather than use the term *schema*. Second, our agent limits the number of schemes by making a selection on the basis of the schemes' proclivity values. The agent computes higher-level schemes' proclivity values from primitive schemes' proclivity values, which are preset by the modeler. Third, we do not use our agent to solve a predefined problem but only expect it to construct satisfying behaviors to increase its average satisfaction through its interaction with its environment.

Finally, our implementation uses a notion of *episodic memory* because the algorithm involves a form of storage

[1] Translated by the authors from the French "L'intelligence (et donc l'action de connaître) ne débute ni par la connaissance du moi, ni par celle des choses comme telles, mais par celle de leur interaction; c'est en s'orientant simultanément vers les deux pôles de cette interaction qu'elle organise le monde en s'organisant elle-même" (Piaget, 1937, p. 331).

and reuse of the agent's history (in the form of hierarchical sequential schemes). Derbinsky and Laird (2009) have noted that implementing episodic memory in a cognitive system closely relates to the computer science domain of *learning through experience*. Learning through experience ranges from reusing declarative knowledge, in the case of case-based reasoning (Kolodner, 1992), to reusing procedural experience in the case of temporal case-based reasoning (Sanchez-Marre, Cortes, Martinez, Comas, & Rodriguez-Roda, 2005) or trace-based reasoning (Mille, 2006). In particular, Trace-based reasoning addresses the issue of modeling non-Markovian sequences—sequences that do not obey Markov's hypothesis that each item in the sequence depends only on a fixed number of previous items, with this number being known a priori (Putterman, 1994). Trace-based reasoning, however, usually follows a knowledge-representation approach that requires a human modeler to define the process of encoding episodes, and a human user to drive the process of reusing episodes.

Our approach differs from trace-based reasoning's knowledge-representation approach in that we neither initially endow our agent with knowledge of its environment, nor do we supply it with knowledge during its life. Instead, we propose a way for the agent to autonomously encode and reuse episodes based on the agent's intrinsic motivations. To be autonomous, the learning mechanism needs to automatically address three issues related to modeling non-Markovian sequences. First, it must determine appropriate start and end points for sequential episodes of interest; second, it must appropriately encode the contexts so that old episodes can be recalled based on context similarity; and third, it must organize episodes into appropriate hierarchical levels so that the appropriate level can be reused (an episode at a given level being a sequence of lower-level episodes). By automatically addressing these issues, our model advances theories of learning through experience in non-Markovian problems, moving towards an implementation of episodic memory within autonomous cognitive agents as we have defined them previously.

Sections 3 and 4 present the algorithm's implementation. Section 5 reports the behavior of the algorithm in two experiments. Although we did not yet intend to model reasoning processes, we have, nevertheless, implemented this algorithm in a cognitive architecture, namely Soar 9 (Laird & Congdon, 2009). Implementing this algorithm in a cognitive architecture allows us to compare it to other cognitive modeling approaches, which we do in Section 6. Finally, in the conclusion, Section 7, we discuss our results and the lessons learned for future work.

We have named our agent Ernest for Evolutionist pRagmatic self-orieNted lEarning in a conStructivist and boTtom-up approach, or simply because it is Ernest. From now on in this paper, we refer to Ernest with the pronoun *he* for easier reading.

3. Main concepts in the algorithm

Ernest's interactions with his world are represented using two kinds of objects: *schemes* and *interactions* that are hierarchically organized. Fig. 1 provides an example.

At its base level, Fig. 1 shows three example primitive schemes: turn "➘", touch forward "–" (this dash represents Ernest's antenna), and attempt to move forward "➔" (here, we note schemes within double quotes). Ernest is initialized with these primitive schemes as nodes in long-term memory. In Soar's terms, the lattice of schemes and interactions form working memory elements (WMEs) (i.e., extensions of the state ⟨s⟩). When Ernest is in a simulated world, the primitive schemes' effects are hard-coded in the environment and the environment returns a binary enaction status

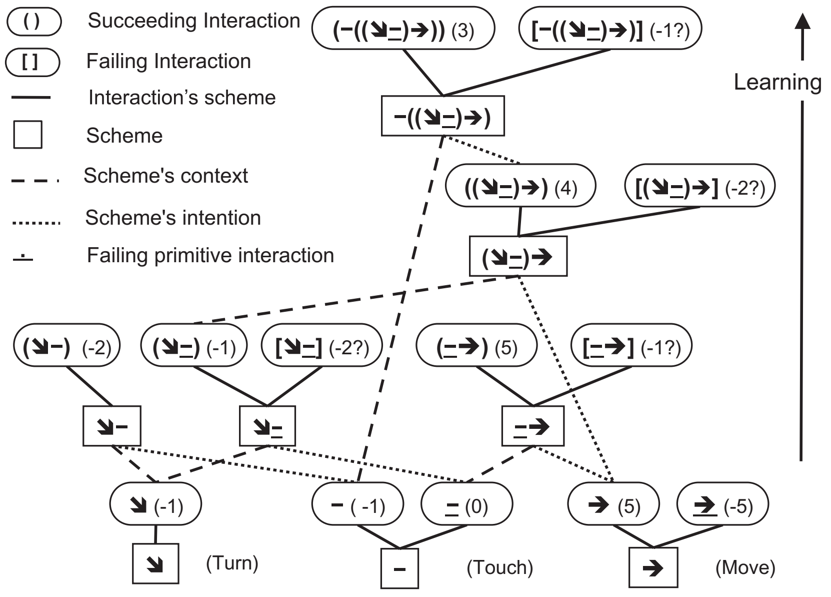

Fig. 1. Example hierarchical structure of schemes and interactions that arise over time.

to Ernest (failure or success). Enacted primitive schemes and their binary enaction feedback are the only information exchanged between Ernest and his environment.

In Fig. 1, primitive interactions are represented above the primitive schemes. Primitive failing interactions are represented with the scheme's symbol being underlined. For example, the interaction ➔̲(−5) corresponds to bumping into a wall while the interaction ➔(5) corresponds to moving and succeeding in moving forward. Each primitive interaction is associated with a proclivity value in Ernest's long-term memory (here, we note interactions without double quotes and followed by their proclivity value in parentheses, to differentiate them from schemes). Proclivity values are used to compute Ernest's impetus to select a scheme for enaction, as explained in Section 4.1 (scheme selection).

Primitive proclivity values are chosen and preset by the modeler according to the behavior he or she intends to generate. In our example, interaction ➔(5) means that Ernest *enjoys* moving forward, while interaction ➔̲(−5) means that Ernest *dislikes* bumping into walls. Similarly, interaction –(−1) means that Ernest touches a wall in front of him and slightly dislikes it while –̲(0) means that Ernest touches an empty square, leaving him indifferent. In these settings, touching a wall is considered a success and touching an empty square is considered a failure, which is an arbitrary choice that has no consequence on the agent's learning (but the proclivity values do).

As introduced in Section 2, higher-level schemes—also called composite schemes—consist of a sequence of two interactions "interaction$_1$ interaction$_2$", meaning that, in the context when interaction$_1$ was enacted, the agent can intend to enact interaction$_2$. Accordingly, we refer to interaction$_1$ as the scheme's *context interaction* and to interaction$_2$ as the scheme's *intention interaction*. A scheme's context interaction and intention interaction are always contiguous in time. Ernest learns composite schemes by associating context interactions with intention interactions as they are encountered during his activity. A composite scheme will, in turn, propose to enact its intention interaction if its context interaction matches again Ernest's situation. Scheme learning consists of adding the newly-learned scheme to Ernest's long-term memory, as a node and two edges pointing to its two sub-interactions, as depicted in Fig. 1. This way, entire sequences of interactions can be learned in a pairwise fashion. In the figures throughout the paper, the edge pointing to a scheme's context interaction is represented as a dashed line and the edge pointing to a scheme's intention interaction as a dotted line. For example, scheme "➘–̲" is learned when Ernest has performed the sequence of turning right and touching an empty square. So, scheme "➘–̲" indicates that, when Ernest has successfully turned right, he can expect to touch an empty square. Similarly, scheme "➘–" is learned when Ernest has successfully turned right and touched a wall, meaning that Ernest has also learned this sequence.

In addition, schemes have a weight that holds the expectation they generate. A scheme's weight corresponds to the number of times the scheme has been encountered, as detailed in Section 4.3 (learning mechanism). Consequently, over the course of Ernest's interactions, the relative scheme weights determine Ernest's overall expectations in specific contexts. For example, at a given point in time, if scheme "➘–" has been encountered three times (weight = 3) and "➘–̲" only twice (weight = 2) then the overall expectation generated by a simple right turn would be of touching a wall with a weight of (3 – 2 = 1). This expectation can, however, be balanced by other elements of context as we will see.

Once learned, composite schemes can be enacted as a whole sequence. Like primitive schemes, composite schemes can either succeed or fail when the agent tries to enact them, as further explained in Section 4.2. Each composite scheme can therefore again give rise to two higher-level interactions: its *failing interaction* and its *succeeding interaction*. Composite schemes' succeeding interactions are represented within parentheses (e.g., (–̲➔)(5)) and composite schemes' failing interactions are represented within square brackets (e.g., [–̲➔](−1?)). This parentheses and square brackets notation reflects the hierarchical structure of composite schemes and interactions.

While Fig. 1 displays how the schemes and interactions are stored in Ernest's long-term memory, it does not render the schemes' temporal structures. We illustrate these temporal structures with an example in Fig. 2.

Fig. 2 shows the enaction of scheme "–((➘–̲)➔)" (touch wall – turn right – touch empty – move forward) on the 84th decision cycle in our experiment. During this cycle, the intention to enact this scheme came from the activation of higher-level schemes (not represented in Fig. 2) that matched the previous context, resulting in this scheme being proposed and then selected. In the figures throughout this paper, the selected scheme is represented with double lines, and the enacted interaction with a wider gray line. A scheme's enaction consists of recursively following the scheme's hierarchical structure down to the primitive

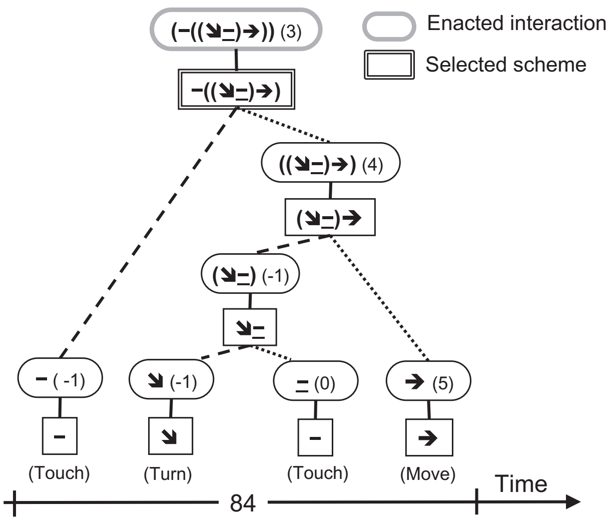

Fig. 2. Enaction of an example scheme.

schemes, and sequentially enacting the primitive schemes in the environment. For example, scheme –((➘–̲)➔)'s enaction consisted of successfully enacting scheme "–" (touch a wall), then successfully enacting scheme "(➘–̲)➔", consisting of successfully enacting "➘" (turn right) then enacting "–" with an expected failure status (touch an empty square), and finally successfully enacting "➔" (move forward). While a composite scheme's enaction results in a flat sequence of primitive schemes, the algorithm keeps track of the scheme's hierarchical structure to construct a hierarchically organized context (as detailed in Section 4.4, scope) and to support higher-level hierarchical learning. Notably, schemes do not encompass branching behaviors. Instead, branching behaviors are obtained during the scheme selection phase where higher-level schemes compete to propose different possible intentions.

Note that in Fig. 2, scheme "–" is represented twice in "–((➘–̲)➔)'s" temporal structure because it is enacted twice, whereas in Fig. 1, "–" is represented once in "–((➘–̲)➔)'s" storage structure because "–" corresponds to a single node in Ernest's long-term memory. In this example, because all sub-schemes in the hierarchy met their expected status, scheme "–((➘–̲)➔)" was successfully enacted. Therefore, the enacted interaction was (–((➘–̲)➔))(3).

Finally, we introduce the notion of the agent's *scope*. The *scope* is a data structure computed by the agent that represents the agent's current situation. As such, the scope constitutes the agent's short-term memory that is, in fact, a set of pointers pointing to interactions stored in long-term memory. The agent computes the *next scope* at the end of each decision cycle. The scope is a subset of all the interactions that have been completed during the previous decision cycle, as explained in Section 4.4 (scope assessment). Therefore, the scope corresponds to the agent's internal representation of its current situation in terms of interaction patterns. At the beginning of a decision cycle, the agent uses the current scope as a context to select the next scheme to enact, as detailed in Section 4.1 (scheme selection).

Overall, the concepts of scheme, interaction, and scope provide a novel way of representing the interaction between an agent and its environment. In traditional methods, the interaction is represented as a loop: perception-cognition-action (the "cognitive sandwich" as criticized by Hurley (1998)). Instead, in our approach, the scope represents perception as a construct created within the agent's cognitive system. As such, we expect the scope to better capture the entanglement of perception with intrinsic motivations and decision processes. Moreover, because we do not encode our ontological views about the environmental structure in the agent's perception, these views do not shape the agent's information processing. The agent is left alone to discover the structure of its environment through its experience.

4. Algorithm procedure

The algorithm follows two overlapping cyclical loops. These two loops are represented in Fig. 3. The highest-level

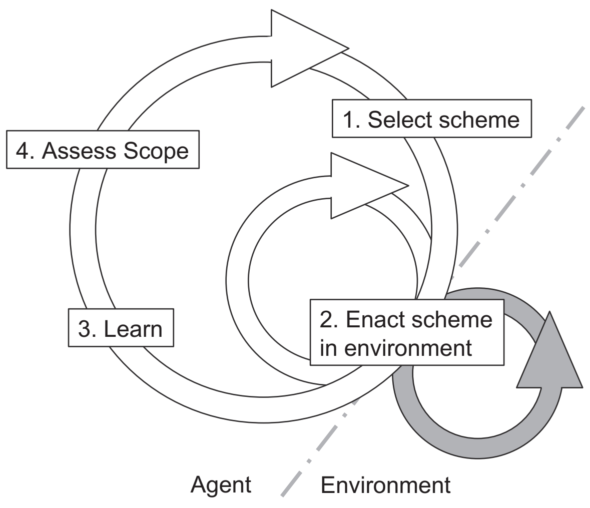

Fig. 3. Algorithm procedure.

loop (large white circle) consists of: 1: selecting a scheme for enaction, 2: trying to enact the selected scheme, 3: learning what can be learned from this trial, 4: computing the resulting scope, and finally looping back to step 1. We call this loop the control loop because it is at this level that the agent *decides* what scheme to try to enact. Step 2: (trying to enact a scheme) constitutes a nested loop that goes through the selected scheme's hierarchical structure and tries to enact each of its primitive interactions recursively, as introduced in Section 3. We call this loop the *automatic loop* (small white circle) because this loop enacts sub-schemes below the agent's decision process.

In Fig. 3, the gray circle represents the environment's loop. Each revolution of the automatic loop corresponds to a revolution of the environment's loop that returns the binary feedback on the enacted primitive scheme. In the control loop, the scheme's enaction constitutes only one step, regardless of the scheme's level in the hierarchy. Therefore, at the control loop level, all schemes are handled similarly to primitive schemes, making it possible to recursively learn hierarchically-organized higher-level schemes. The automatic loop returns control to the control loop either when the selected scheme has entirely been correctly enacted, or when the automatic loop is interrupted because one of the sub-interactions was not correctly enacted. The four control loop steps are described next.

4.1. Scheme selection

On each decision cycle, the scheme to enact is selected through an activation mechanism triggered by the current scope. This mechanism is illustrated by an example in Fig. 4.

Fig. 4 describes Ernest's behavior up to the point of Decision Cycle (DC) 56 in the experiment. In this figure, the scheme's weight is noted in the schemes' boxes. On DC 55, scheme "➔" was successfully enacted (move forward). Accordingly, scheme "–̲➔" was completed over

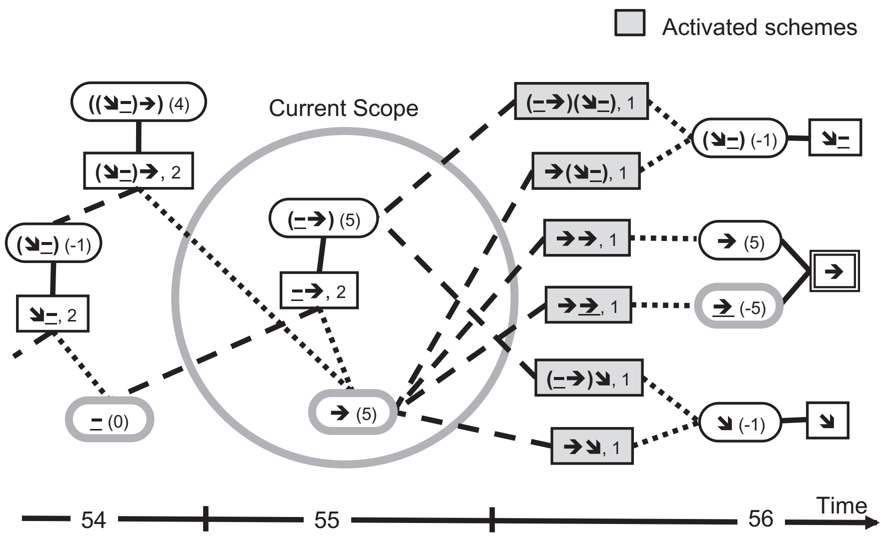

Fig. 4. Example selection of a scheme.

DC 54 and 55 (touch empty – move forward), and scheme "(➘–)➔" was completed over DCs 53 through 55 (turn right – touch empty – move forward). In this context, the scope after DC 55 was made of interactions ➔(5) and interaction (–➔)(5) (the scope can include several interactions as detailed in Section 4.4, scope assessment).

All schemes whose context interaction belongs to the current scope are activated. All the activated schemes create a weighted proposition for their intention interaction. The weight of this proposition is equal to the weight of the proposing scheme multiplied by the proclivity of the proposed intention. On DC 56, the activated schemes are the six schemes represented in gray in Fig. 4. All these schemes have a weight of 1 because they all have been recently learned and not yet reinforced. For example, scheme "➔➔" proposes to enact scheme "➔" next with a weight of $1 \times 5 = 5$. This proposition can be understood as Ernest having an impetus to try to move forward because he has previously succeeded once in this context and he enjoys that. Alternatively, scheme "➔➔" proposes to enact scheme "➔" with a weight of $1 \times (-5) = -5$. In this case, Ernest's impetus is counter-balanced by an *apprehension* to move forward because he has also previously bumped into a wall in this context once before, and he dislikes that. In addition, primitive schemes receive a default proposition with a weight of 0 if no higher-level scheme proposes them. This is the case of scheme "–" in our example (not shown in Fig. 4). This makes Ernest pick random primitive schemes when he has not yet learned to compute anticipations.

When a composite scheme is proposed, a little heuristic applies (in this example, scheme "➘–"). If its weight is greater than a threshold, called the *regularity sensibility threshold*, then it is effectively proposed for being enacted as a whole. If its weight is lower than or equal to the threshold and if its proclivity value is positive, then the scheme is not proposed but its proposition is propagated to its first (context) subscheme. In essence, this mechanism ensures that higher-level schemes are sufficiently rehearsed before being enacted as a whole. During each rehearsal, higher-level schemes are reinforced, which tends to pass the reinforcement from one hierarchy level to the next. A lower regularity sensibility threshold results in a faster adoption of potentially less satisfying higher-level schemes. A higher regularity sensibility threshold results in a slower adoption of potentially more satisfying higher-level schemes. The role of the regularity sensibility threshold is further discussed in Sections 4.4 (scope assessment) and 5 (experiment). In our example, scheme "➘–" has not reached the regularity sensibility threshold (in this experiment, 4) in DC 56, and its satisfaction is negative, so its proposition is not considered further at that point in time.

Finally, propositions are summed by schemes, and the scheme with the highest summed proposition is selected. If several schemes are tied, one is picked randomly. Notably, such stochasticity is not necessary in Ernest's algorithm; we have also implemented deterministic versions of Ernest that break the tie by simply picking the first scheme in the list of tied schemes. In our example, the total proposition for "➘" equals $1 \times (-1) + 1 \times (-1) = -2$. The total proposition for "➔" equals $5 - 5 = 0$. The highest summed proposition was therefore 0 (for "➔" and "–"). Scheme "➔" happened to be selected amongst these two (shown with a double lined box).

Then, scheme "➔" happened to fail, Ernest bumped into a wall, resulting in the enacted interaction ➔(−5) on DC 56 (shown in a gray outlined box). This experience caused scheme ➔➔'s weight to be incremented, and other schemes to be learned according to the learning mechanism detailed in Section 4.3 (learning).

This selection mechanism shows that proclivity values do not operate as a reward but rather as inward drives that either push the agent toward or restrain him from enacting certain behavior. Moreover, the reinforcement does not operate as a form of reward propagation but as a mechanism for experience counting. By driving the agent's behavior, this selection mechanism also shapes the agent's experience and consequently the agent's learning. The agent does not learn all that can be learned in the environment—which would be overwhelming—but only what its motivations make it experience. Moreover, this mechanism guarantees that higher-level regularities will only be constructed upon lower-level regularities that have been effectively tested.

By including several interactions at different levels in the scope, the selection mechanism is designed to capture different levels of regularities. Over time, the schemes that capture the most robust level of regularity afforded by the environment and that best fulfill the agent's satisfactions become prevalent. For example, the safest decision to try to move forward will not rest upon the last primitive step but rather requires exploiting regularities that cover the experience of several steps.

4.2. Scheme enaction

Once a scheme has been selected, Ernest tries to enact it. To do so, the algorithm recursively looks down the selected scheme's sub-interactions, as introduced in Section 3 and illustrated in Fig. 2. When a primitive sub-interaction is reached, its scheme is enacted in the environment. The agent then compares the feedback status to the expected status of the primitive interaction. If the received status matches the expected status, the primitive interaction is considered as correctly enacted, and the algorithm proceeds to the next interaction to enact. This enaction procedure continues until the end of the selected scheme, unless a sub-interaction does not meet its expectation at any level in the hierarchy.

If an expectation is not met, whether because the sub-interaction expects success but fails at some point, or the sub-interaction expects failure somewhere but accidentally succeeds, the enaction is interrupted and considered incorrect. In this case, an *actually enacted scheme* is constructed based on the part of the hierarchy that has been actually enacted before the interruption. This actually enacted scheme is associated with a success status to form the *actually enacted interaction* that Ernest will include in the next scope. In addition, the selected scheme that failed to complete is assigned a failing status to form a failing interaction that will also be part of the next scope. We call this failing interaction the *performed interaction*. Including both the actually enacted interaction and the performed interaction in the next scope ensures that the next scope effectively reflects the situation where Ernest ended up, both at the hierarchy level of Ernest's intention (the performed interaction) and at the hierarchy level where Ernest has felt back (the actually enacted interaction). Fig. 5 illustrates the scheme failing mechanism with an example.

On Decision Cycle 72 of our experiment, scheme "–➔" was selected for enaction (touch empty square – move

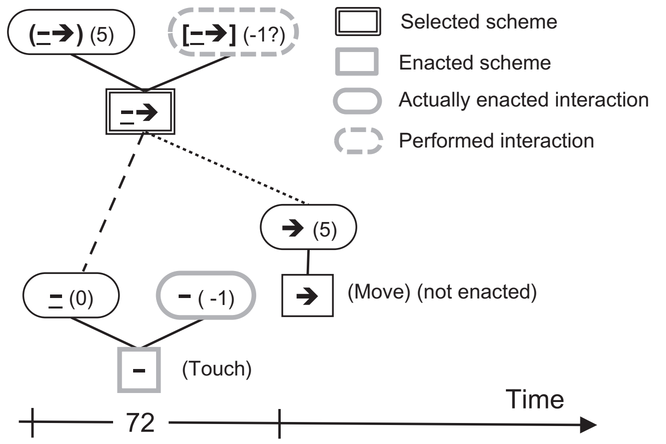

Fig. 5. Example of an interruption due to incorrect enaction of a scheme.

forward). At this point, scheme "–➔" has been learned during earlier decision cycles and is about to dominate scheme "➔" precisely because it brings less dissatisfaction when it fails, as we are going to see.

As it happened when Ernest tried to enact –➔'s first step (context interaction), Ernest touched a wall, an unexpected success. This unexpected feedback from the environment caused "–➔" to be interrupted. Consequently, in this decision cycle, the actually enacted scheme was "–" and the actually enacted interaction was –(−1). The performed interaction was [–➔](−1) made of selected scheme "–➔" associated with a failing status. Because scheme "–➔" was interrupted, Ernest did not suffer the dissatisfaction of bumping into a wall but only the dissatisfaction of touching a wall. Therefore, Ernest learned that the satisfaction value of the performed interaction [–➔] was equal to −1.

During forthcoming selection phases, higher-level schemes will generate balanced expectations for success or failure of scheme "–➔". Because scheme "–➔" yields less dissatisfaction in case of failure than scheme "➔", scheme "–➔" will tend to win forthcoming selection phases over scheme "➔". Hence, Ernest has learned to touch before possibly moving forward instead of directly trying to move forward. Note that this is more than just learning an IF THEN rule, this is adopting a proactive practice consisting of touching in specific contexts to ensure safer moves (see the comparison with bottom-up rule learning in the related work, Section 6).

Notably, a composite scheme's failing interaction cannot have a fixed proclivity value because a composite scheme can fail during any of its steps, each failure leading to a possibly different proclivity value. To address this problem, when the selected scheme fails more than once, its failing interaction's proclivity value is replaced by the average of both its previous value and of the actually enacted interaction's proclivity value. This is a simple way to have the failing interaction's proclivity value reflect some plausible proclivity value based on recent experience. Note that this proclivity value will be balanced by higher-level schemes' weight when generating a failure expectation during forthcoming selection phases.

In effect, this scheme enaction mechanism, associated with the selection mechanism, favors schemes that consist of going through unsatisfying interactions at the scheme's beginning to reach more satisfying or safer interactions at the end (in this example, touching before possibly moving forward). This is because the automatic loop does not rely on proclivity values to select sub-schemes. Therefore, Ernest is not limited to simple reflex interactions toward the highest immediate satisfaction; Ernest also learns to enact unsatisfying interactions to place him in situations where he can enact more satisfying interactions.

4.3. Scheme learning

This section describes how new schemes are learned in Ernest's long-term memory or reinforced if they already exist. This section uses the term learning in its computer science sense, meaning the recording of new data, as opposed to the rest of the paper that views learning as an emergent phenomenon demonstrated by the agent's behavior. The learning of a higher-level scheme occurs from the experience gained through trying to enact the selected scheme in a given context. The learning mechanism records higher-level schemes to memorize the association of elements of context (interactions belonging to the scope) with the enacted interactions. The proclivity value associated with a higher-level interaction is set equal to the sum of the proclivity values associated with its sub-interactions. This means that enacting a scheme as a whole is as satisfying as enacting all its sub-interactions individually. Fig. 6 illustrates this learning mechanism with an example.

On Decision Cycle 83, scheme "–➔" was successfully enacted, resulting in a current scope that included interactions ➔(5), (–➔)(5), and ((–((➘–)➔))(–➔))(8), represented inside a gray circle in Fig. 6. From this scope, DC 84's selection mechanism activated schemes "➔(–((➘–)➔))" and "(–➔)(–((➘–)➔))" (among other schemes not represented in Fig. 6). These activated schemes proposed to enact scheme "–((➘–)➔)". This proposition happened to win the selection phase, and this scheme happened to be successfully enacted, resulting in the enacted interaction (–((➘–)➔))(3). Such a scheme's enaction results in two learning mechanisms.

The first learning mechanism consists of creating or reinforcing schemes whose context interaction belongs to the current scope and whose intention interaction is the enacted interaction. In Fig. 6's example, the reinforced schemes are the activated schemes "➔(–((➘–)➔))" and "(–➔)(–((➘–)➔))" (in light gray in the figure). The learning consists of incrementing their weight from 4 to 5. The newly created scheme in this case is "((–((➘–)➔)) (–➔))(–((➘–)➔))" that is added to Ernest's long-term memory with a weight of 1. In addition, interaction (((–((➘–)➔)) (–➔))(–((➘–)➔)))(11) is added to Ernest long-term memory with a proclivity value set equal to the sum of its sub-interactions proclivity values: 8 + 3 = 11.

When the intended scheme is correctly enacted on step n, the number of schemes learned or reinforced with the first learning mechanism (noted L_n^1) is equal to the number of interactions in the current scope (noted S_{n-1} because it results from the previous cycle):

$$L_n^1 = S_{n-1}$$

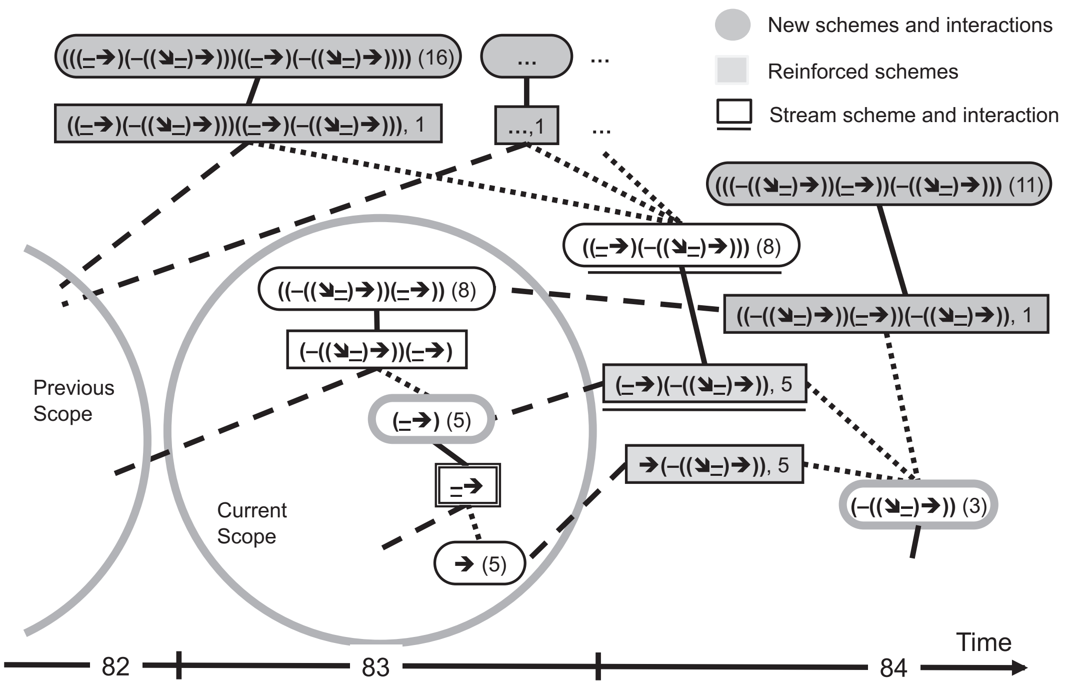

Fig. 6. Example of scheme learning.

When an intended composite scheme is incorrectly enacted, new or reinforced schemes are constructed both from the actually enacted interaction and from the performed interaction. Hence, in this case:

$L_n^1 = 2 \times S_{n-1}$

In effect, the first learning mechanism tends to append new interactions to the right side of already existing schemes. To extend schemes to their left side, a second learning mechanism applies. This second learning mechanism focuses on a particular scheme that we call the *stream scheme*. The stream scheme is the scheme that connects two consecutively enacted interactions (i.e., whose context interaction and intention interaction correspond to the two interactions that have been consecutively enacted). We also define the *stream interaction* as the stream scheme's succeeding interaction. At a given point in time, the stream scheme presents the particularity of being performed in the context of the previous scope (if the intended scheme is incorrectly enacted, the stream scheme is constructed from the actually enacted interaction). The second learning mechanism consists of creating or reinforcing schemes that memorize the association of elements of the previous context (interactions in the previous scope) with the stream interaction. Schemes learned with the second learning mechanism will propose again the stream scheme when situations similar to the previous context are encountered again, which helps higher-level schemes dominate lower level schemes when appropriate.

In DC 84's example, the stream interaction is ((=➔)(–((➘=)➔)))(8) (underlined in Fig. 6). Fig. 6 shows scheme "((=➔) (–((➘=)➔)))((=➔)(–((➘=)➔)))" (top left) as an example scheme learned with the second learning mechanism on DC 84. All the schemes learned with the second learning mechanism associate a context interaction that belonged to DC 82's scope (interactions not represented in Fig. 6) with the stream interaction. When scopes similar to DC 82's scope occur again, these schemes will propose to enact scheme "(=➔)(–((➘=)➔))" as a whole, which will result in this scheme eventually replacing scheme "–((➘=)➔)".

On step n, the number of schemes learned or reinforced with the second learning mechanism is:

$L_n^2 = S_{n-2}$

Summing the two learning mechanisms, the total number of learned or reinforced schemes on step n is given by the formulas:

$L_n = S_{n-1} + S_{n-2}$ (in case of correct enaction)

$L_n = 2 \times S_{n-1} + S_{n-2}$ (in case of incorrect enaction)

Notably, the enacted scheme is not reinforced ("–((➘=)➔)" on DC 84), nor are any of its sub-schemes. The reinforcement is not associated with a scheme enaction but with higher-level schemes learning. This highest-level reinforcement principle contributes to letting higher-level schemes possibly override lower-level schemes when higher-level schemes manage to capture higher-level regularities.

4.4. Scope assessment

The last step of the control loop consists of assessing the resulting *new scope*. As noted in Section 3, the scope is a set of interactions that were just completed. We call these interactions the *completed interactions*. Completed interactions are of three kinds: (a) the interactions that were just learned or reinforced; (b) the interaction that was just enacted, or in case of incorrect enaction, the actually enacted interaction plus the performed interaction; (c) the last subsequences of the interaction that were just enacted. For example, Fig. 7 shows the completed interactions on step 84.

As noted, on DC 84, interaction (–((➘=)➔))(3) was enacted meaning its intention interaction ((➘=)➔)(4) is also being just completed. Similarly, ((➘=)➔)(4)'s intention interaction ➔(5) is also just completed. Moreover, interactions on top of (–((➘=)➔))(3) are also being completed simultaneously. These are all the interactions learned or reinforced during the learning phase.

If all the completed interactions were included in the next scope, then after the next cycle, as many new schemes would be learned. Cycle after cycle, this process would lead to the construction of top-level schemes that would be data structures representing Ernest's entire experience since being instantiated. In this case, we can approximate the rate of memory growth by reckoning that all the learned schemes would be part of the next scope, that is, referring back to Section 4.3: $S_n = L_n$. Therefore, leaving apart the case of incorrect enactions, the number of learned or reinforced schemes on step n is approximated by:

$L_n = L_{n-1} + L_{n-2}$

This formula defines the Fibonacci sequence, which is known to grow exponentially. To prevent such combinatorial explosion, the algorithm restrains the scope's content. Metaphorically, this restriction can be understood as Ernest not being aware of his full prior history at each instant. Instead, Ernest's awareness only covers the current stream of his activity. His awareness of his current stream of activity is, however, framed by his full experience. At each moment, Ernest internally represents his situation as a set of interactions that are at the highest possible—while sufficiently confirmed—level in the hierarchy that he has learned thus far.

The heuristic we have implemented to restrain the scope consists of only including the enacted interaction plus the other completed interaction whose scheme's weight has passed the regularity sensibility threshold (introduced in Section 4.1). This means that Ernest becomes aware of regularities only when they have been encountered a sufficient number of times. Over Ernest's learning process, regularities pop up in Ernest's scope when they have sufficiently been encountered. These regularities form higher-level schemes that Ernest then tries to enact as a whole. These schemes' enactions open the way to even higher-level

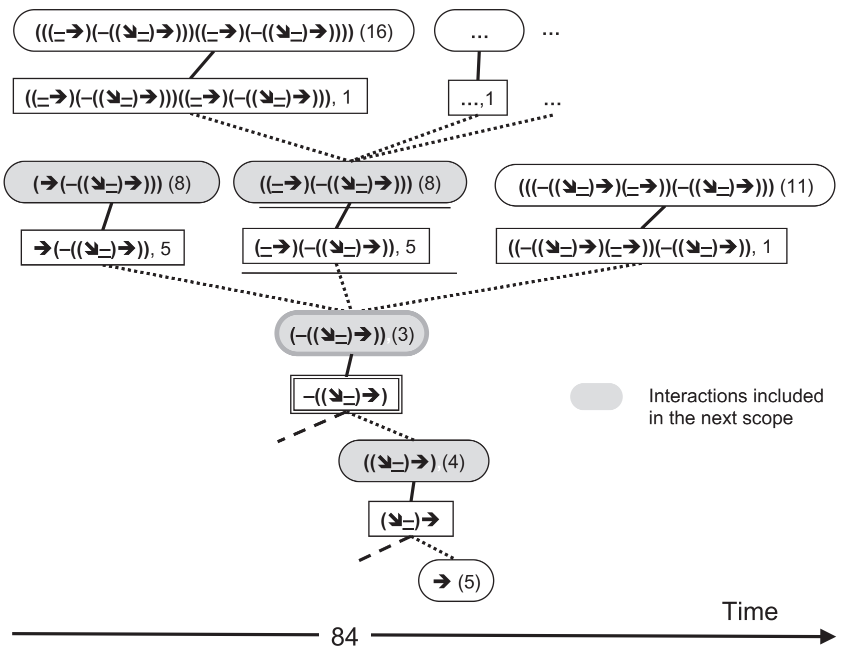

Fig. 7. Example of scope assessment.

regularities popping up in Ernest's scope and even higher-level schemes being learned.

In addition, only one sub-level of the actually enacted interaction is included in the scope. That is, referring back to Fig. 7, sub-intention interaction ((↘–)➔)(4) is included in the scope but not its sub-intention interaction ➔(5). Therefore, the scope construction mechanism works consistently across any level of the enacted scheme in the hierarchy; this enables the recursive learning of higher-level schemes. We can interpret this as Ernest being unaware of the interactions that he enacts at the lowest levels of the hierarchy; this again helps Ernest deal with the situation complexity. Following this heuristic, the current scope on DC 84 is thus made of interactions (➔(–((↘–)➔)))(8), ((–➔)(–((↘–)➔)))(8), (–((↘–)➔))(3), and ((↘–)➔)(4) (in gray boxes in Fig. 7). This scope covers 6 steps of interaction with the environment that encode the sequence *sense empty – move forward – sense wall - turn right – sense empty – move forward.*

In our experiments, the number of new learned schemes per cycle proved to range from 0 to 10, making the memory grow linearly with the number of cycles. An additional decay function could be implemented to delete unused schemes and keep the memory constant.

4.5. Implementation

Because we developed this algorithm to study and generate cognitive phenomena, it is natural to consider implementing it with a cognitive architecture. The most widely-used cognitive architectures, ACT-R (Anderson, 2007) and Soar (Laird & Congdon, 2009), were originally designed to model problem solving and symbolic processing rather than adaptive mechanisms. It might thus appear more natural to use cognitive architectures that are designed to support implicit knowledge, for example CLARION (Sun, Peterson, & Merrill, 1999), MicroPsi (Bach, 2008), CLA (Chaput, 2004), ICARUS (Langley & Choi, 2006), or ADAPT (Benjamin, Lyons, & Lonsdale, 2004). These later architectures, however, already implement specific learning algorithms, and we found that only Soar offered enough flexibility for our research on a new learning algorithm.

Moreover, the latest release of Soar, version 9, supports reinforcement learning (Soar-RL). While we are not implementing classical reinforcement learning, Soar-RL provides a mechanism that is useful to our approach, namely the *valued preference mechanism*. This mechanism provides an easy way to implement Ernest's scheme selection phase because it supports the weighted proposition of operators. Soar-RL then easily supports selecting the most strongly weighted operator. For these reasons, we have chosen Soar 9. The current version of the algorithm has 60 Soar productions,[2] making it a not overly complex Soar model.

5. Experiments

We developed two experiments where the agent was afforded hierarchical sequential regularities to learn and organize. We used the first experiment to illustrate the learning process in detail (Section 5.2) and to demonstrate the gain in Ernest's satisfaction over each run (Section 5.3). We use the second experiment for a qualitative analysis of Ernest's emergent behavior (Section 5.4).

[2] http://e-ernest.blogspot.com/2009/10/ernest72-soar.html

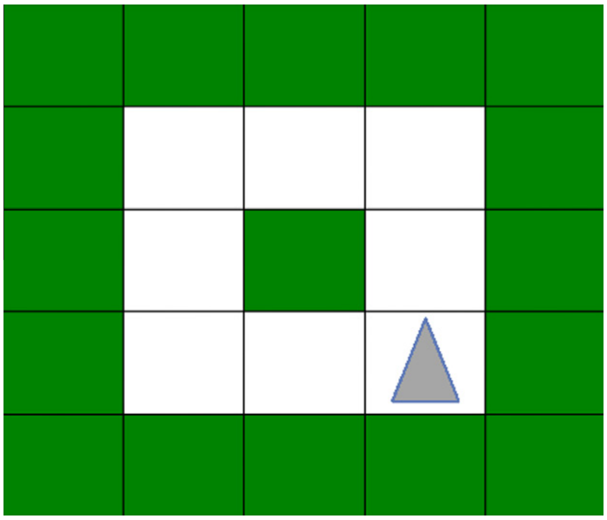

Fig. 8. The environment for experiment 1.

5.1. Simple loop experiment

Although the interaction's structure—resulting from coupling the environment with the agent's primitive schemes—is fundamentally sequential, the environment can be presented to external observers as a two-dimensional grid. This environment is represented in Fig. 8 and was implemented from Cohen's (2005) Vacuum environment based on Russell and Norvig's (1995) general vacuum cleaner environment.

In Fig. 8, the triangle represents Ernest's position and direction, white squares represent empty squares where Ernest can go, and filled squares represent walls. Ernest's primitive schemes and interactions are defined as described above ("➘" = turn 90° right (−1/NA), "–" = touch the square ahead (−1/0), "➔" = attempt to move one square forward (5/−5)). Additionally, we have primitive scheme "➚" consisting of turning to the left (−1/NA) (turning schemes "➘" and "➚" always succeed in this environment). Notably, the squares do not offer a unique identifier accessible to the agent, in contrast with many experiments with reinforcement learning mechanisms (e.g., Singh, Lewis, & Barto, 2009; Sun & Sessions, 2000). This lack of unique identifier for each agent's state makes the learning more challenging than in classical reinforcement learning experiments because the agent cannot attribute a *quality value* to transitions identified a priori. This additional difficulty explains the choice of an environment whose topological layout seems otherwise simpler than in traditional reinforcement learning experiments.

Our experimental settings offer a first notable regularity, namely that Ernest can increase his average satisfaction by touching ahead before trying to move forward, and by not moving forward when he touches a wall. Next, Ernest can increase its satisfaction by repeating sequences consisting of moving forward twice and turning once. Higher-level regularities consist of repeating this later sequence. The effects of this learning mechanism are described in detail in Section 5.2.

5.2. An example life of Ernest

Fig. 9 illustrates an example run. Videos of similar runs can be seen online.[3]

In Fig. 9, enacted schemes are represented at the lowest level in each line with a black outline. Learned schemes are represented on top of the enacted schemes with a gray outline. Failing higher-level schemes are represented as gray boxes (occurring at steps 68 and 72). The numbers from 1 to 91 indicate the control loop iterations (Decision Cycles).

At the beginning, Ernest acts randomly because he has not yet learned expectations. Every cycle, however, Ernest constructs or reinforces several schemes. For clarity, Fig. 9 only reports the construction and the reinforcement of the schemes that proved decisive in increasing Ernest's satisfaction in this run. Scheme "➘–" is constructed on step 8. Scheme "➘–" is then reinforced on DC 28, 34, and 49. Then scheme "➘–" passes the regularity threshold and Ernest attempts to enact it for the first time on step 68 but fails and enacts "➘=" instead.

Notably, a scheme "➘➘" is constructed on DC 19. This scheme is reinforced on DC 33, 42, and 43. It is then enacted twice on DC 44 and 45. It is, however, not used any further because other options prove more satisfying (its proclivity value is −2).

On DC 47, Ernest discovers the regularity *touch empty-move forward*, allowing him to generate schemes "=➔" and "(➘=)➔". After DC 47, scheme "=➔" always prompts Ernest to try to move forward after touching an empty square. Consequently, from then on, scheme "=➔" is quickly reinforced (rehearsed) in DC 55, 59, 63, and 71. Ernest tries to enact it for the first time on DC 72, but unsuccessfully. =➔'s failure resulted in falling back to –(−1), as detailed Section 4.2 and Fig. 5. From then on, Ernest has learned to always touch before moving forward. Scheme "=➔" is then successfully enacted on DC 74, 77, 80, 83, and 85.

On DC 69, scheme "(➘=)➔" passed the regularity sensibility threshold, and became included in the scope, which resulted in the learning of the fourth-order scheme "–((➘=)➔)". Then, on DC 73, the enaction of scheme "(➘=)➔" generated the learning of scheme "(=➔)(–((➘=)➔)". Scheme "–((➘=)➔)" is enacted for the first time on DC 84 as detailed in Fig. 2. Scheme –((➘=)➔)'s enaction generated the learning of "((=➔)(–((➘=)➔)))((=➔)(–((➘=)➔)))" as described in Section 4.3 and in Fig. 6. Scheme "(=➔)(–((➘=)➔)" starts to be enacted on DC 87.

After DC 87, Ernest keeps on performing the sequence touch empty – move forward – touch wall – turn right – touch empty – move forward. With this sequence, Ernest obtains a satisfaction of 8 with 6 primary steps, i.e., 1.33

[3] http://e-ernest.blogspot.com/2009/07/ernest-64.html

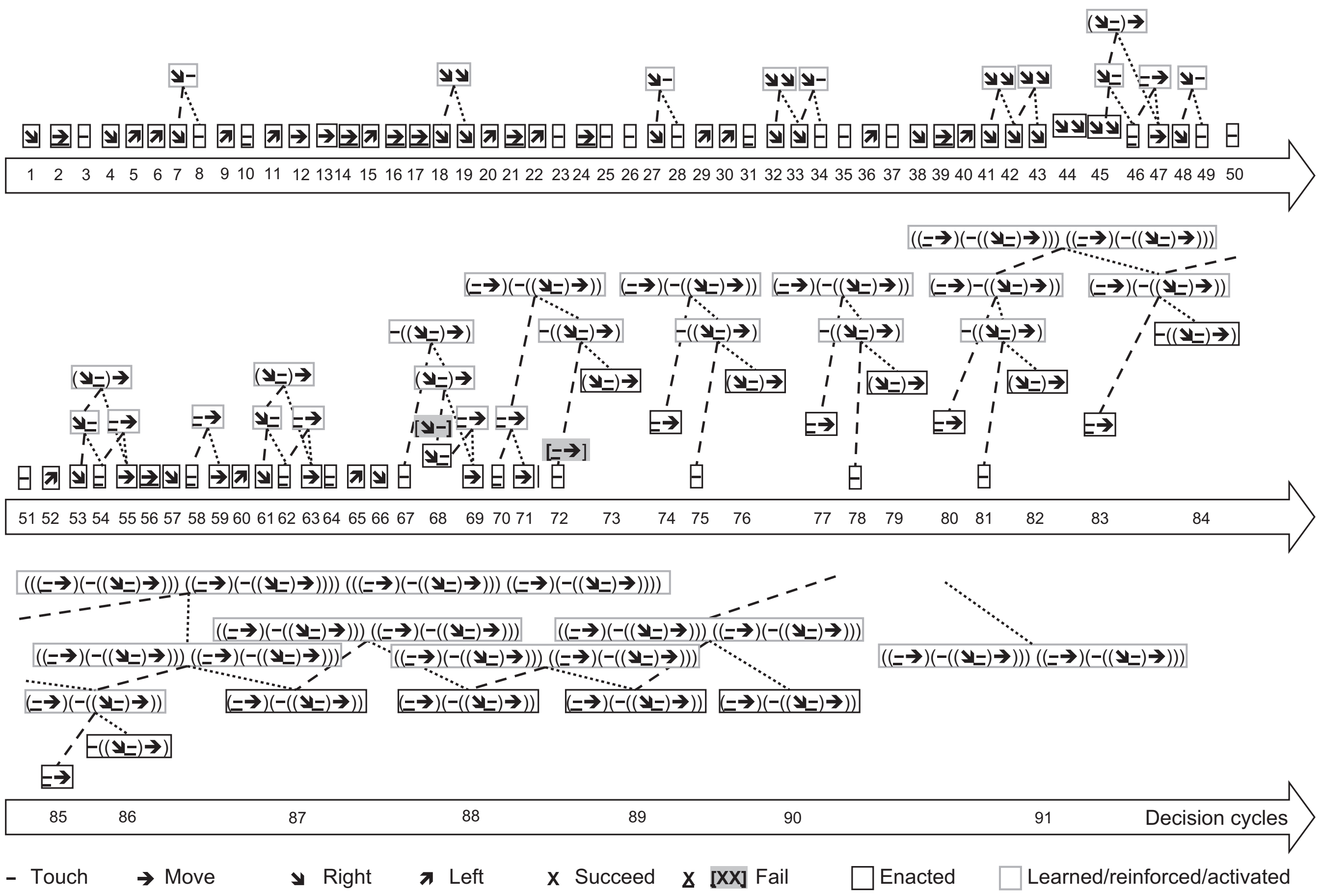

Fig. 9. An example run among the 18 reported in row 6 of Table 1.

per primary step. If we keep Ernest running, the repetition of this sequence generates higher-level schemes that keep on doubling this sequence.

In this example, Ernest did not learn the optimum sequence in the environment. In fact, Ernest has no way of knowing whether the stabilized sequence is optimum or not. Ernest only repeats a sequence when other actions appear less likely to bring satisfaction, based on what he has learned previously. In most instances, Ernest first learns to touch before moving, after which he begins to build other regularities based on this initial pattern.

The time before stabilization depends on the value of the regularity sensibility threshold. As noted in Section 4.1, schemes are not proposed for enaction until they pass this threshold. As said in Section 4.4, interactions are not included in the scope until their scheme has passed this threshold. Consequently, schemes do not contribute to higher-level learning until they have passed this threshold. As a tradeoff between satisfaction and the pace of learning, the regularity sensibility threshold was set to 4 in this experiment, meaning that a sequence had to be encountered 4 times before being considered for automatic enaction or retention in Ernest's situation awareness.

5.3. A hundred lives of Ernest

Experiment 1 was run 100 times, stopping each run when Ernest had reached a stable sequence, and clearing Ernest's memory between each run. The results are summarized in Table 1, the runs are aggregated by the proclivity value (satisfaction) of the final stable sequence (third column). The fourth column indicates the number of primitive steps that constitutes the stable sequence. The fifth column indicates Ernest's average satisfaction per step when he has reached the stable sequence. The last column reports the average number of decision cycles before reaching this sequence. Rows 1 through 6 report 86 runs where Ernest learned to circle the whole loop, achieving a satisfaction per step greater than or equal to 1.33. Among these rows, the first row represents 22 runs where Ernest found the optimum sequence: *move forward – move forward – turn* (this sequence may be implemented by different schemes). Rows 7–14 report 14 runs where Ernest has reached a stable sequence that results in him staying on one lap of the environment, with a satisfaction per step between 0.40 and 1.40.

The summary row shows that the average final satisfaction per step was 1.92. It was reached in an average of 72 decision cycles. In comparison, other experiments yielded

Table 1
Performance of Ernest over a hundred runs sorted by satisfaction of final scheme.

Rownumber	Runs	Satisfaction of final scheme	Steps of final scheme	Average satisfaction/step	Decision Cycles to stability
1	22	9	3	3.00	50
2	22	9	4	2.25	79
3	4	9	5	1.80	75
4	4	8	4	2.00	69
5	16	8	5	1.60	62
6	18	8	6	1.33	84
7	1	7	5	1.40	76
8	1	7	6	1.17	109
9	1	7	7	1.00	108
10	2	6	8	0.75	116
11	3	4	4	1.00	61
12	1	4	5	0.80	95
13	3	3	3	1.00	71
14	2	2	5	0.40	96
	100	7.98	4.49	1.92	72

an average satisfaction value per step of −0.93 without any learning and −0.38 with only the first-level scheme learning. This data demonstrates that, in all the runs, the hierarchical learning mechanism substantially increased the agent's satisfaction, compared to no or non-hierarchical learning.

These results vary depending upon the configuration of the primitive proclivity values and the regularity sensibility threshold. Our purpose was not to optimize this experiment; other values would lead to other learning results. For example, at the extreme, when Ernest is given a positive proclivity value for turning, he learns to keep spinning in place forever, which indeed gets him a good satisfaction but does not demonstrate much interesting behavior.

Table 2
Primitive interactions and their proclivity in the second experiment.

Interaction	Description	Proclivity
➔	Move forward	10
➔̲	Bump wall	−10
↘	Turn to the right toward an empty square	0
↘̲	Turn to the right toward a wall	−5
↗	Turn to the left toward an empty square	0
↗̲	Turn to the left toward a wall	−5
–	Touch a wall ahead	0
=̲	Touch an empty square ahead	−1
\	Touch a wall on the right	0
\̲	Touch an empty square on the right	−1
/	Touch a wall on the left	0
/̲	Touch an empty square on the left	−1

5.4. Ernest in other worlds

Because the mechanism works bottom-up, the learning of low-level regularities does not depend on the environment's overall complexity. To illustrate this, we placed Ernest in the more complex environment displayed in Fig. 10.

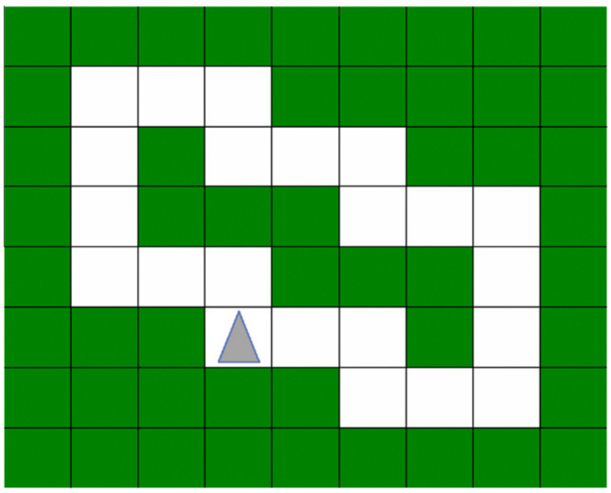
Fig. 10. The environment for experiment 2.

In Environment 2, Ernest was endowed with extra primitive schemes to touch the square to the right and to the left, and their associated proclivity values listed in Table 2 (the symbols \ and / represent Ernest antenna toward his right and left sides).

Ernest learned more elaborate perceptual behaviors in experiment 2 than in Experiment 1 because he could construct a better sense of his situation by touching on his sides. These behaviors can be seen in videos online.[4] Table 3 provides an example activity trace. Numbers from 1 to 205 indicate the first 205 decision cycles during which Ernest learned to circle this new environment. For each decision cycle, the actually enacted scheme is reported. If the actually enacted scheme was not the intended scheme, an exclamation mark (!) is appended.

Ernest completed a first tour on DC 171 and a second tour on DC 205. The sequence "=̲➔" was first encountered on DC 38 and 39. Scheme "=̲➔" was first incorrectly enacted on DC 128 and first correctly enacted on DC 133. In parallel, Ernest learned "↗➔" and "↘➔" that exploit the turning schemes' feedback to ensure safe move

[4] http://e-ernest.blogspot.com/2009/10/enrest-72.html

Table 3
Example activity trace in the second experiment.

```
1➔ 2\ 3➔̲! 4\ 5\ 6/̲! 7\ 8/̲ 9/̲! 10➘̲! 11➚̲! 12➔̲! 13/̲! 14/̲ 15➔̲! 16➚ 17➘̲! 18➘̲! 19➔̲! 20/! 21/ 22➘ 23➔ 24➘̲! 25\̲!
26/ 27➔̲! 28➚̲! 29- 30➔̲! 31➔̲! 32- 33➘̲! 34- 35➚̲! 36\ 37➚ 38-̲! 39➔ 40/ 41➘̲! 42/̲! 43/̲ 44- 45/̲! 46/̲ 47/̲ 48/̲ 49➚
50\ 51-̲! 52➔ 53- 54/! 55\̲! 56➚̲! 57➚ 58➔ 59➚̲! 60/̲! 61/̲ 62(/̲/̲) 63/̲ 64(/̲/̲) 65➚ 66➔ 67➔̲! 68➘ 69➔ 70- 71- 72(/̲/̲)
73➚ 74➔ 75/ 76\! 77-̲! 78➔ 79\ 80- 81\ 82- 83(/̲/̲) 84➚ 85➔ 86/ 87➚̲! 88/̲ 89➚ 90➔ 91\̲! 92- 93/! 94- 95\̲! 96➘
97➔ 98\ 99-̲! 100➔ 101- 102- 103/ 104\̲! 105➘ 106➔ 107\ 108- 109(/̲/̲) 110➚ 111➔ 112-̲! 113➔ 114- 115/! 116/ 117-
118\̲! 119➘ 120➔ 121- 122\! 123- 124/̲! 125(➚➔) 126-̲! 127➔ 128-! 129/! 130\̲ 131➘ 132➔ 133(-̲➔) 134(-̲➔) 135-!
136/! 137\̲ 138(➘➔) 139(-̲➔) 140-! 141/! 142\̲ 143(➘➔) 144-! 145(/̲(➚➔))! 146(-̲➔) 147-! 148/! 149(\̲(➘➔)) 150-!
151(/̲(➚➔))! 152(-̲➔) 153-! 154/! 155(\̲(➘➔)) 156-̲! 157➔ 158(-̲➔) 159-! 160/! 161(\̲(➘➔)) 162(-̲➔) 163-! 164/!
165(\̲(➘➔))  166-!  167(/̲(➚➔))!  168(-̲➔)  169-!  170/!  171(\̲(➘➔))  172-!  173(/̲(➚➔))  174(-̲➔)  175-!  176/!
177(\̲(➘➔)) 178-̲! 179➔ 180((-̲➔)/) 181(\̲(➘➔)) 182(-̲➔) 183-! 184/! 185(\̲(➘➔)) 186-! 187(/̲(➚➔)) 188(-̲➔) 189-!
190(/(\̲(➘➔))) 191(-(/̲(➚➔))) 192(-̲➔) 193-! 194(/(\̲(➘➔))) 195-̲! 196➔ 197((-̲➔)/) 198(\̲(➘➔)) 199(-̲➔) 200-!
201(/(\̲(➘➔))) 202-! 203((/̲(➚➔))(-̲➔)) 204-! 205(/(\̲(➘➔)))
```

forward (first enacted on DC 125 and 138). Due to the effects of learning, Ernest stopped bumping into walls after DC 67 and started exhibiting a more methodical explorative behavior shown by more consistent touching, turnings, and moving forward. On DC 128 through 131, we can see that he started applying a strategy consisting of, when he has touched a wall ahead, touching to the left, then, if there is a wall to the left, touching to the right, and, if there is no wall to the right, turning to the right. This strategy let him safely circle around his environment. This strategy is then implemented by the construction of schemes "/̲(➚➔)" and "\̲(➘➔)" first enacted on DC 145 and DC 149. This association demonstrates the emergence of the use of inexpensive behaviors (touching) as "perception" and the use of expensive behavior as "action" (turning). Then, on DC 203, the trace shows that Ernest started enacting a higher-level scheme consisting of turning and moving forward twice.

This trace also shows that the sequence representing Ernest's activity is non-Markovian because Ernest's behavior relies on regularities whose duration was not limited a priori. Moreover, Ernest is capable of reusing knowledge learned much earlier. For example, the behavior enacted during the last decision cycle (DC 205) still relies on the pattern ➘➔ that was first learned during DC 22 and 23.

It is not obvious to attribute emergent cognitive phenomena to Ernest when observing a printed report of his behavior, but we find it striking when watching the experiment running. In particular, observers can infer Ernest's growing awareness of his sides as he begins to associate touching with turning toward each side. Through empathy, observers can also infer basic subjective states such as enjoyment, pain, angst, hope, and growing knowledge of the environment. Moreover, across runs, Ernest instances differentiate themselves from each other through their choices and resulting experiences. While these basic cognitive phenomena are still rudimentary, we consider them as supporting the triple-autonomy hypothesis. A model whose implementation follows the principles of intrinsic motivation, autonomous situation awareness, and autonomous learning did generate behaviors that arguably exhibit early-stage cognitive phenomena.

Other experiments show that, when shifts in the environment occur during the course of Ernest's activity, he reuses low-level schemes that still apply and learns new higher-level schemes applicable to the new environment. However, studies in more complex environments show the current algorithm's limitations, which are discussed in the conclusion.

6. Comparison with related work

We first must note that our model uses Soar in a significantly different way than traditional symbolic cognitive models (Newell, 1990). In our case, although we use Soar's symbolic computation mechanism, we do not use *physical symbols* (Newell & Simon, 1975) to transcribe our own ontology of the environment into the agent (e.g., we do not define a symbol for walls). In this sense, our model can be seen as entirely situated at the sub-symbolic level even though it uses Soar's symbolic computation level. As such, our model constitutes an example showing that the distinction between symbolic and sub-symbolic levels is not tied to the architecture's commitments but rather can depend on the modeler's usage of the architecture.

Similarly, our work also differs from a bottom-up rule learning mechanism (e.g., Sun, Merrill, & Peterson, 2001). Certainly, to an external observer, Ernest seems to learn logical rules. For example, as discussed in Section 4.2, Ernest learns that if he touches a wall ahead, then he should not move forward. As opposed to rule learning mechanisms, however, our algorithm does not construct explicit logical rules in the agent's long-term memory, nor are we claiming that our agent has learned to process explicit logical knowledge. We do not expect our agent to learn logic but only to exploit regularities of interaction. This conforms to Piaget's (1937) developmental theory that states that logical knowledge processing will only arise after a long developmental process. This developmental process would imply finding persistent objects in the environment and representing them internally before eventually learning language and logic (mostly through social interaction).

Our work also differs from robotics studies that develop mechanisms for autonomously learning the correlation between actuators and sensors (e.g., Pierce & Kuipers, 1997). These studies conform to the perception-cognition-action loop by implementing a mechanism that transfers the sensors' signal to a perceptual buffer independently from the mechanisms that exploit the content of this buffer. Instead, we pre-encode the association of the agent's actions with the agent's inputs within primitive sensorimotor schemes. Our agent does not need to learn the relation between its actions and its inputs because these relations are predefined. What our agent needs to learn is the appropriate behavior to construct its perceptual representation of its situation. Our approach offers the advantage that sensorimotor schemes provide a place to implement the agent's intrinsic motivations. We, nevertheless, expect difficulties when implementing more complex perceptual behaviors (such as vision) and will consider using a hybrid approach in future studies, as suggested for example by Norman (2002).

As we mentioned in Section 2, our work also borrows from reinforcement learning principles. On the other hand, our work differs from reinforcement learning in that the reinforcement does not come from a reward that is given when a final goal is reached. Because of this difference, we could not use Soar's built-in reinforcement learning mechanism (Soar-RL, Laird & Congdon, 2009). Soar-RL's mechanism requires that all transitions between states in the problem-space be identified a priori, in the form of one or several operators. In this way, the reinforcement-learning algorithm can backward propagate the reward by modifying the operator *preference values*. Instead, we do not encode all the transitions a priori in our agent's model; rather our agent dynamically learns to identify contexts and transitions. This dynamic context recognition allows the agent to adapt to different environments. We believe that this approach better accounts for natural organisms that exist without uniquely identifying all the possible situations they might encouter (a problem often known as perceptual aliasing).

The algorithm also draws from work in the field of trace-based reasoning (Mille, 2006) to implement a mechanism that enables the agent to learn by reusing past sequential episodes of activity. With this mechanism, we bring an innovative answer to the issues related to non-Markovian sequence modeling introduced in Section 2—issues such as automatically delimitating episodes, organizing episodes in a hierarchy, and encoding context in a way that supports the appropriate reuse of episodes. By addressing these issues, our work differs from studies that require Markov's hypotheses, such as models based upon Bayesian statistics (e.g., Griffiths, Kemp, & Tenenbaum, 2008). Our answer to the issues of non-Markovian problems rests upon the idea that behavioral patterns should be constructed from the bottom up in a hierarchical fashion. Our work, nevertheless, differs from approaches of mere statistical hierarchical temporal learning such as Hawkins and Blakeslee's (2004) in that our agent interactively tests hypothetical patterns in the environment and makes a selection based on the agent's intrinsic motivation before learning higher-level patterns. These views relate to pragmatic epistemology (James, 1907), to evolutionist epistemology (Popper, 1972) that suggest that knowledge evolves on the basis of usage, and to constructivist epistemology (Piaget, 1970), that suggests that knowledge selection is driven by the subject's intrinsic motivations.

Our autonomous mechanism of learning through interaction offers a novel way of implementing episodic memory. In essence, the agent's encoding of its experience reflects the way the agent has learned to understand this experience. This approach also addresses the problem introduced in Section 1 of getting the agent to gradually learn to pre-encode its experience for future reuse. To this end, we can see Fig. 9 and Table 3 as views on Ernest's episodic memory in particular instances, where the agent's activity is represented in the form of increasingly elaborated patterns of behavior. This approach of episodic memory differs from Soar's built-in episodic memory (version 9.3). Soar's episodic memory does not encode the temporal structure of episodes but rather encodes *snapshots*; therefore, Soar's episodic memory cannot be as directly queried as Ernest's. Furthermore, Soar's manner of encoding does not evolve over the agent's experience, but rather consists of the raw content of the agent's working memory in each interaction cycle (corresponding to our automatic loop cycles that enact primitive schemes). Because of these differences, we could not use Soar's episodic memory.

The algorithm also draws lessons from genetic algorithms (e.g., Mitchell, 1996) by adopting an evolutionist approach. They, however, differ because genetic algorithms typically focus on phylogenetic evolution of the learning mechanism over generations of agents (e.g., Floreano, Mondada, Perez-Uribe, & Roggen, 2004), whereas our algorithm focuses on the ontogenetic cognitive development of each agent through the selection of the most useful knowledge. In the future, we nonetheless believe the phylogenetic approach can help us implement mechanisms to adapt our agent's inborn primitive proclivity values based on an evolutionary selection over generations of agents.

There are currently two major approaches for implementing intrinsic motivation in artificial agents. One approach consists of implementing motivation as behavioral rules that directly represent either emotions (e.g., Gadanho & Hallam, 1998) or drives (e.g., Sun, 2009). The second approach implements intrinsic motivation as curiosity and search for novelty (Blank et al., 2005; Oudeyer & Kaplan, 2007; Schmidhuber, 2010). We follow a third approach that implements an *inversion of reasoning* argument as some authors have argued for (e.g., Dennett, 1991). With the inversion of reasoning argument, Dennett postulates that humans do not "eat the cake because they find it sweet" but humans rather "find the cake sweet because they eat it". Humans have evolved with the tendency to enact this behavior, which defines their liking for sweetness. Our algorithm implements this view by incrementally organizing the agent's behavior around inborn proclivities. We consider that our algorithm illustrates this argument in the case of emergent cognition. We, nonetheless, imagine that the three approaches could be complementary in the case of higher-level cognition.

As for the test bed environment and for the experimental paradigm, our approach appears to be rather unique. We must note that our experiment substantially differs from maze solving experiments (e.g., Sun & Sessions, 2000) or from hierarchical sequence learning as depicted in the classical taxi cab experiment (Dietterich, 2000). In their experiments, the learning comes from a reward value that is given when the goal is reached, the learning requires that the agent has a pre-encoded way to uniquely identify each state in the environment, and the learning occurs over multiple runs (often thousands). In contrast, we have no final goal for the agent that would provide a reward, and states are not directly identifiable by the agent. Rather, the learning occurs through each run; and all the agent's memory is reinitialized between each run (including all forms of reinforcement, i.e., the schemes' weight). Our experimental paradigm also radically differs from those proposed in Soar's tutorial (e.g., the Eaters and TankSoar environments, Laird & Congdon, 2009), in that our approach does not encode the modeler's strategy and problem analysis. Because we could not find experiments related to our approach in the literature, we propose our experiments as an initial test paradigm for investigating the triple-autonomy hypothesis.

7. Conclusion

This study advocates approaching cognition by primarily focusing on interaction while conceiving of perception and representation as secondary constructs. In a proof-of-concept algorithm, we show that this approach offers a way to implement intrinsic motivation in the form of inborn proclivities associated with primitive possibilities of interaction. This approach also offers a way for the agent to construct a representation of the situation that is not tied to the modeler's ontological commitments about the environment. In addition, this approach offers a way to implement an autonomous learning mechanism where the agent learns to encode its experience to cope with the environment's complexity in compliance with the agent's intrinsic motivation.

In our experiments, the agent appears to an observer as if it learned to use certain schemes to inform its perceptions (schemes "–", "/", and "\" to *sense* the squares around in Section 5.4) and to determine subsequent actions based upon these perceptions. Therefore, the agent seems to learn to actively perceive its environment and pragmatically understand its perception simultaneously. By pragmatic understanding, we refer to a pragmatic epistemology according to which "meaning is use" (Wittgenstein, 1953). This result is original in that nothing in our agent initially differentiated perceptual behavior from action behavior except their cost (predefined proclivity values). Perceptual behavior emerged through the agent's activity, which also grounded the meanings of the agent's perceptions in its activity (Harnad, 1990). Once this perceptual behavior is learned, the agent perceives its environment in a new way, which makes new behavior possible, specifically, more systematic exploration. This conforms to the developmental design principle suggested by enactive theories of cognition (De Loor, Manac'h, & Tisseau, 2010) mentioned in the introduction.

Moreover, the way that the algorithm constructs a data structure in short-term memory (namely the scope) to represent the agent's situation can shed some light on the notion of situation awareness. One of the most accepted definitions of situation awareness (SA) is Endsley's (1995): "The perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future". Because the scope is usable by the agent for following its motivations, we argue that the agent understands the scope in a pragmatic sense. Because the scope activates schemes, the scope lets the agent construct a "projection of its current state in the near future". Our algorithm, therefore, offers an implemented model of these two aspects of Endsley's views. The scope is nevertheless not a representation of the agent's environment as we see it. Instead, the scope is a representation of the agent's situation in terms of the agent's possibilities of behavior. As such, the scope meets Gibson's (1979) ecological understanding of situation awareness. Gibson suggested that the environment is perceived in terms of interactions afforded to the agent by the environment, i.e., *affordances*. Our algorithm, therefore, also illustrates Gibson's views. This simulation generates emergent SA and acts like it has emergent SA, even though the representation is not explicit to an outside observer. As such, we consider this simulation not only as a model but also as an implementation of an emergent cognitive system.

In our implementation, the scope is, however, still rudimentary. While it covers a certain "volume of time" as Endsley called for, it does not cover the finding of distinct "elements of the environment [...] within a volume of space". Developmental theories suggest that finding interaction regularities constitutes a prerequisite toward finding persistent objects in the environment, but we still need to investigate the concrete mechanisms that will implement this passage.

Preliminary experiments in more complex environments indicate that the current algorithm faces three notable limitations. The first limitation concerns the management of a large number of learned schemes. Although the number of learned schemes is controlled as described in Section 4.4 (scope assessment), the algorithm, nevertheless, constructs several hundred schemes during each run in the environment described in Fig. 10. The algorithm stores schemes in Soar's declarative working memory and Soar is not optimized for a large working memory. This optimization strategy adopted by Soar eventually became a serious constraint on developing the algorithm. Our agent's knowledge operates both as procedural (when a scheme is enacted), and as declarative (when a scheme is part of the agent's situational representation in the scope). Soar's strong distinction between procedural and declarative knowledge, therefore, further compromises future developments in Soar. These limitations of Soar regarding the handling of schemes have also been noted by Benjamin et al. (2004) who proposed the ADAPT cognitive architecture as an extension of Soar that supports schema manipulation. Additionally, more advanced implementations will require mechanisms to reduce the number of schemes, for instance, by forgetting unused schemes or merging schemes that have similar primitive sequences.

The second limitation is that the algorithm is not good at finding spatial regularities. For example, if we replace the central wall square with an empty square in Fig. 8, the agent becomes less likely to find the most satisfying regularity, that of making a continuous circuit around its environment. We expected this limitation because we did not design this algorithm to learn spatial regularities, but moving the agent to more complex 2-D or 3-D environments will require addressing this issue in the future.

The third limitation is that the agent becomes quickly trapped in local optima, preventing it from exploring complex environments. As the agent continues to run, its recursive learning mechanism causes it to learn schemes representing increasingly long repetitions of a cyclical pattern. Our current experimental setup stops the agent when it detects these repetitions. To generate more interesting behaviors, future algorithms should implement other forms of intrinsic drives. For example, we can exploit the detection of cyclical behaviors to generate a sense of *boredom* that would incite the agent to break the cycle and move toward more exploration.

To move the agent to more complex environments, we are currently re-implementing the algorithm in Java. We have also implemented primitive interactions that react to distal properties of the environment (a sort of rudimentary vision) (Georgeon, Cohen, & Cordier, 2011). To help the agent find spatial regularities, we are now working on enhancing the agent's architecture with additional mechanisms to represent space (Georgeon, Marshall, & Ronot, 2011). The current results including the open source code are online.[5]

The algorithm currently works as an advanced adaptive mechanism but does not allow the agent to reflect upon or reason about knowledge. To move toward reflection, we envision making the agent capable of inhibiting its actions and only simulating them internally, without enacting them in the environment. We are now able to implement such an internal simulation mechanism because our agent's knowledge is associated with expectations, that is, our agent learns knowledge of its actions' effects. The agent can internally simulate different possible courses of action based on the expected outcomes associated with each action. The agent can then choose the course of action that has the greatest expected satisfaction based on these simulations. Understanding the scope as the agent's situation awareness, we anticipate such internal simulations would resemble a *stream of awareness*, in compliance with Cotterill's (2001) proposal that thought is an "internally simulated interaction with the environment", and Hesslow's (2002) argument that this simulation hypothesis can explain our experience of an inner world.

[5] http://e-ernest.blogspot.com/

Acknowledgments

We gratefully thank Richard Carlson, Jonathan Morgan, James Marshall, Amélie Cordier, and Mark Cohen for their much-appreciated comments on this report. Support for this study was provided by ONR (N00014-06-1-0164 and N00014-08-1-0481), DTRA (HDTRA 1-09-1-0054), and ANR (ANR-10-PDOC-007-01).

References

Aha, D. W. (1998). Feature weighting for lazy learning algorithms. In H. Liu & H. Motoda (Eds.), *Feature extraction construction and selection: A data mining perspective*. Norwell, MA: Kluwer Academic Publishers.

Anderson, J. R (2007). *How can the human mind occur in the physical Universe?* New York: Oxford University Press.

Arbib, M. (1992). Schema theory. In S. Shapiro (Ed.). *Encyclopedia of Artificial intelligence, 2nd ed.* (Vol. 2, pp. 1427–1443). New York, NY: John Wiley & Sons.

Arkin, R. (1987). Motor schema-based mobile robot navigation. *The International Journal of Robotics Research, 8*(4), 92–112.

Bach, J. (2008). *Principles of synthetic intelligence. Building blocks for an architecture of motivated cognition*. New York, NY: Oxford University Press.

Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE, 76*(8), 996–1005.

Benjamin, P., Lyons, D., & Lonsdale, D. (2004). ADAPT: A cognitive architecture for robotics. In *Sixth international conference of cognitive modeling* (pp. 337–338). Pittsburgh, PA: Lawrence Earlbaum.

Blank, D. S., Kumar, D., Meeden, L., & Marshall, J. (2005). Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems, 32*(2), 125–150.

Chaput, H. H. (2004). *The Constructivist Learning Architecture: A model of cognitive development for robust autonomous robots*. Unpublished doctoral dissertation, The University of Texas, Austin.

Cohen, M. A. (2005). Teaching agent programming using custom environments and Jess. *AISB Quarterly, 120*(Spring), 4.

Cordier, A., Mascret, B., & Mille, A. (2009). Extending case-based reasoning with traces. In *Grand Challenges for reasoning from experiences, Workshop at IJCAI* (pp. 23–32). Pasadena, CA.

Cotterill, R. (2001). Cooperation of basal ganglia, cerebellum, sensory cerebrum and hippocampus: Possible implications for cognition, consciousness, intelligence and creativity. *Progress in Neurobiology, 64*, 1–33.

De Loor, P., Manac'h, K., & Tisseau, J. (2010). Enaction-based artificial intelligence. Toward co-evolution with humans in the loop. *Minds and Machine, 19*, 319–343.

Dennett, D. (1991). *Consciousness explained*. The Penguin Press.

Dennett, D. (1998). *Brainchildren: Essays on designing minds*. Cambridge, MA: MIT Press.

Derbinsky, N., & Laird, J. E. (2009). Efficiently implementing episodic memory. In *8th international conference on case-based reasoning, ICCBR* (pp. 403–417). Seattle, WA.

Dietterich, T. G. (2000). An overview of MAXQ hierarchical reinforcement learning. In *SARA02 4th international symposium on abstraction, reformulation, and approximation* (pp. 26–44). London, UK: Springer-Verlag.

Drescher, G. L. (1991). *Made-up minds, a constructivist approach to artificial intelligence*. Cambridge, MA: MIT Press.

Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors, 37*(1), 32–64.

Floreano, D., Mondada, F., Perez-Uribe, A., & Roggen, D. (2004). Evolution of embodied intelligence. In F. Iida, R. Pfeifer, L. Steels, & Y. Kuniyoshi (Eds.), *Embodied artificial intelligence (Vol. 3139)* (pp. 293–311). Berlin: Springer-Verlag.

Froese, T., Virgo, N., & Izquierto, E. (2007). Autonomy: A review and a reappraisal. In *9th Conference on artificial life* (pp. 455–464). Lisbon, Portugal: Springer-Verlag.

Gadanho, S., & Hallam, J. (1998). Exploring the role of emotions in autonomous robot learning. In *AAAI fall symposium on emotional intelligence* (pp. 84–89). Orlando, FL: AAAI Press.

Georgeon, O., Cohen, M., & Cordier, A. (2011). A Model and simulation of early-stage vision as a developmental sensorymotor process. In *Artificial Intelligence Applications and Innovations*. Corfu, Greece.

Georgeon, O., Marshall, J., & Ronot, P.-Y. (2011). Early-stage vision of composite scenes for spatial learning and navigation. In *First joint IEEE Conference on development and learning and on epigenetic robotics*. Frankfurt, Germany.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Boston: Houghton-Mifflin.

Griffiths, T. L., Kemp, C., & Tenenbaum, J. B. (2008). Bayesian models of cognition. In R. Sun (Ed.), *The Cambridge handbook of computational cognitive modeling*. Cambridge, MA: Cambridge University Press.

Guerin, F., & McKenzie, D. (2008). *A Piagetian model of early sensorimotor development*. Paper presented at the Eighth International Conference on Epigenetic Robotics. Retrieved.

Harnad, S. (1990). The symbol grounding problem. *Physica D, 42*, 335–346.

Hawkins, J., & Blakeslee, S. (2004). *On intelligence*. New York, NY: Times Books.

Hesslow, G. (2002). Conscious thought as simulation of behaviour and perception. *Trends in Cognitive Science, 6*(6), 242–247.

Holmes, M., & Isbell, C. (2005). Schema learning: Experience-based construction of predictive action models. *Advances in Neural Information Processing Systems, 17*, 583–592.

Hurley, S. (1998). *Consciousness in action*. Cambridge, MA: Harvard University Press.

Hutchins, E. (1995). *Cognition in the wild*. Cambridge, MA: MIT Press.

James, W. (1907). Pragmatism.

Kolodner, J. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review, 6*(1), 3–34.

Laird, J. E, & Congdon, C. B (2009). *The soar user's manual version 9.1*. University of Michigan.

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

Mille, A. (2006). From case-based reasoning to traces-based reasoning. *Annual Reviews in Control, 30*(2), 223–232.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

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

Newell, A., & Simon, H. (1975). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM, 19*(3), 113–126.

Noë, A. (2004). *Action in perception*. Cambridge, MA: MIT Press.

Norman, J. (2002). Two visual systems and two theories of perception: An attempt to reconcile the constructivist and ecological approaches. *Behavioral and Brain Sciences, 25*(1), 73–144.

Oudeyer, P.-Y., & Kaplan, F. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation, 11*(2), 265–286.

Perotto, F., Buisson, J., & Alvares, L. (2007). Constructivist anticipatory learning mechanism (CALM): Dealing with partially deterministic and partially observable environments. In *Seventh international conference on epigenetic robotics*. Rutgers, NJ.

Pfeifer, R. (1996). Building fungus eaters: Design principles of autonomous agents. In *4th International conference on simulation of adaptive behavior* (pp. 3–12). Cambridge, MA: The MIT Press.

Piaget, J. (1937). *The construction of reality in the child*. New York: Basic Books.

Piaget, J. (1970). *L'épistémologie génétique*. Paris: PUF.

Pierce, D., & Kuipers, B. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence, 92*, 169–227.

Popper, K. (1972). *Objective knowledge*. Oxford: Oxford University Press.

Putterman, M. L. (1994). *Markov Decision Processes. Discrete stochastic dynamic programming*. New York, NY: Wiley-Interscience.

Rumelhart, D. E., & Norman, D. A. (1981). Analogical processes in learning. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 335–359). Hillsdale: NJ: Erlbaum.

Russell, S., & Norvig, P. (1995). *AI: A modern approach*. Englewoods Cliffs, NJ: Prentice-Hall.

Sanchez-Marre, M., Cortes, U., Martinez, M., Comas, J., & Rodriguez-Roda, I. (2005). An approach for temporal case-based reasoning: Episode-based reasoning. In *ICCBR*. Heidelberg: Chicago, IL: Springer.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation. *IEEE Transactions on Autonomous Mental Development, 2*(3), 230–247.

Simon, H. (1981). *The sciences of the artificial*. Cambridge: MIT Press.

Singh, S., Barto, A. G., & Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In L. Saul, K. Y. Weiss, & L. Bottou (Eds.). *Advances in neural information processing systems* (Vol. 17, pp. 1281–1288). Cambridge, MA: MIT Press.

Singh, S., Lewis, R. L., & Barto, A. G. (2009). Where do rewards come from? In 31st *Annual conference of the cognitive science society* (pp. 2601–2606). Austin, TX.

Stewart, J., Gapenne, O., & Di Paolo, E. (2008). *Enaction: A new paradigm for cognitive science*. Cambridge, MA: MIT Press.

Stojanov, G., Bozinovski, S., & Trajkivski, G. (1997). Interactionist-expectative view on agency and learning. *Mathematics and Computers in Simulation, 44*(3), 295–310.

Suchman, L. A. (1987). *Plans and situated actions*. Cambridge: Cambridge University Press.

Sun, R. (2004). Desiderata for cognitive architectures. *Philosophical Psychology, 17*(3), 341–373.

Sun, R. (2009). Motivational representations within a computational cognitive architecture. *Cognitive Computation, 1*(1), 91–103.

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive Science, 25*, 203–244.

Sun, R., Peterson, T., & Merrill, E. (1999). A hybrid architecture for situated learning of reactive sequential decision making. *Applied Intelligence, 11*, 109–127.

Sun, R., & Sessions, C. (2000). Automatic segmentation of sequences through hierarchical reinforcement learning. In R. Sun & C. L. Giles (Eds.), *Sequence learning* (pp. 241–263). Berlin Heidelberg: Springer-Verlag.

ARTICLE IN PRESS

Varela, F., Thompson, E., & Rosch, E. (1991). *The embodied mind: Cognitive science and human experience*. Cambridge: MIT Press.

Weill-Fassina, A., Rabardel, P., & Dubois, D. (1993). *Représentations pour l'action*. Toulouse: Octares.

Wittgenstein, L. (1953). *Philosophical investigations*. Malden, MA: Blackwell Publishing.

Wolpert, D., & Macready, W. (1997). No free lunch theorem for search. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.