Constructing Phenomenal Knowledge in an Unknown Noumenal Reality

Olivier L. Georgeon, Florian J. Bernard, Amélie Cordier

Constructing Phenomenal Knowledge in an Unknown Noumenal Reality

Olivier L. Georgeon, Florian J. Bernard, Amélie Cordier
Université Lyon 1, LIRIS, UMR5205, F-69622, France.
olivier.georgeon@liris.cnrs.fr, florian.bernard3@etu.univ-lyon1.fr, amelie.cordier@liris.cnrs.fr

Abstract
In 1781, Immanuel Kant argued that cognitive agents ignored the underlying structure of their world "as such" (the *noumenal reality*), and could only know *phenomenal reality* (the world "as it appears" through their experience). We introduce design principles to implement these theoretical ideas. Our agent's input data is not a direct function of the environment's state as it is in most symbolic or reinforcement-learning models. The agent is designed to discover and learn regularities in its stream of experience and to construct knowledge about phenomena whose hypothetical presence in the environment explains these regularities. We report a proof-of-concept experiment in which the agent constructs categories of phenomena, and exploits this knowledge to satisfy innate preferences. This work suggests a new approach to cognitive modeling that focuses on the agent's internal stream of experience. We argue that this approach complies with theories of embodied cognition and enaction.

1 Introduction

Most artificial intelligence and machine learning models represent artificial agents in a perception/cognition/action loop. For example, to Russell & Norvig [1, p. iv], "*the problem of AI is to build agents that receive percepts from the environment and perform actions*". Many theoreticians of cognition, however, criticized this cognitive loop model [e.g., 2]. We reviewed some of these criticisms in a previous paper [3]. Here, we introduce an alternative modeling approach that addresses the design of cognitive agents from the other side. Rather than beginning with the agent/environment dualism, we begin with a monist perspective (inspired by Spinoza's monist philosophy) in which the agent and its universe are considered a single system. Figure 1 represents a series of chunks of experience that this monist system experiences successively during its existence. These chunks of experience may have a positive, a negative, or a null valence, as we shall explain below.

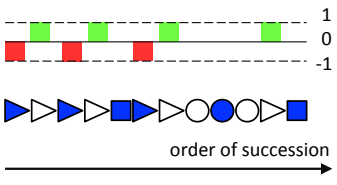

Figure 1: A monist system's stream of experience. White/blue triangles/squares/circles represent chunks of experience that the system experiences one after another. We do not know what these chunks of experience mean, feel like, or are about. The system ignores the shapes and colors of experiences; they are only intended to the reader. Top: the valence of each chunk of experience displayed as a bar-graph: positive (green bars), negative (red bars), or null.

As an example, this paper presents a system that can experience six different chunks of experience represented by *white*/*blue circles*/*squares*/*triangles*. *Square* and *circle* experiences have a null valence; *white triangles* have a positive valence (+1); and *blue triangles* have a negative valence (-1). These valences are arbitrary and remain constant during the system's existence. That is, we think of this system as if it liked experiencing *white triangle* experiences, disliked experiencing *blue triangle* experiences, and was indifferent to experiencing other experiences. For the sake of demonstration, we intentionally place the reader in the same situation as the system: initially ignorant of the meaning of experiences. The impatient reader may make a detour to Section 4 where this meaning is revealed.

At any given moment, the system has partial choice about the experience that it will experience next. In general, the system wants to experience positive experiences, but it may experience negative experiences for two reasons: the reason of *multiple motivations* and the reason of *bounded freedom*.

The reason of multiple motivations is that the system may have other motivational drives besides experiencing positive experiences. Section 3 proposes two of these other drives that may make the system choose null or negative experiences.

The reason of bounded freedom is that the system does not have entire freedom to choose the experience that it is going to experience next. At any given step, the system may intend to experience a particular chunk of experience, but this intention may fail due to reasons external to the system's decision, causing the system to actually experience a different chunk of experience. Figure 2 represents the dual structure of such a bounded-freedom system: the system has an *Intentional Subsystem* (IS), which chooses the next intended chunk of experience i_s, and a *reactive subsystem* (RS), which generates the *actually experienced* chunk of experience e_s. Note that Figure 1 only displays the stream of actual experiences.

Assume that the RS affords regularities in the way it generates the actual experience e_s. These regularities may depend both on the intended experience i_s and on previous history (the RS may not be Markov, i.e., we don't know the extension of dependency a priori). To be in control of the system's stream of experience, the IS must anticipate the actual experiences on the basis of the RS's regularities. To anticipate actual experiences, the IS must construct knowledge that represents the regularities afforded to it by the RS.

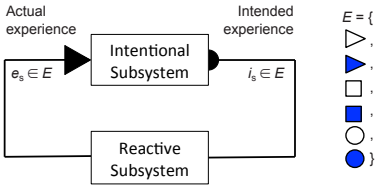

Figure 2: The dual coupling of a bounded-freedom system. At step *s*, the Intentional Subsystem (top) chooses the intended experience i_s from amongst the set E of experiences available to the system. In return, the reactive subsystem (bottom) generates the actually experienced experience e_s. If $e_s = i_s$, then the system succeeded in experiencing the intended experience, otherwise, it failed.

2 Task

We distinguish between two kinds of regularities: *immediate regularities* and *sequential regularities*. Immediate regularities determine how actual experiences correlate with intended experiences regardless of the RS's history. In our example, immediate regularities are such that an intended experience always results in an actual experience of the same shape (but not necessarily of the same color). In fact, we chose the shapes (*squares*, *circles*, *triangles*) to reflect immediate regularities. For example, we expect the IS to learn that, when it intends a *square* experience, the system will indeed actually experience a *square*, perhaps of a different color, but not a *circle* or a *triangle*.

Sequential regularities depend on previous history. A sequential regularity is in the form: in the context when a particular series of experiences has just been experienced (pre-experiences), a particular other series of experiences may probably be successfully experienced next (post-experiences). Figure 3 lists 10 first-order (two-step) regularities afforded by our example RS.

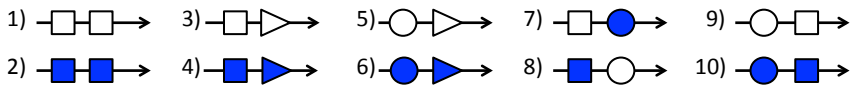

Figure 3: 10 two-step regularities afforded by the RS, consisting of a pre-experience followed by a post-experience. Regularity 1: in the context when *white square* has just been experienced (pre-experience), it is likely that *white square* can be experienced again (post-experience); i.e., if the IS intends to experience any of the *square* experiences again, then the system will more likely experience *white square* than *blue square*. Similar, Regularity 3 means that, in the *white square* context, if the IS intends a *triangle* experience then the system is likely to experience a *white triangle* experience (immediate regularities prevail over sequential regularities).

We expect the IS to discover these regularities and exploit them to obtain positive experiences and avoid negative experiences (with the valence of experiences defined in Section 1). As a result, the system should develop the patterns of experience depicted in Figure 4.

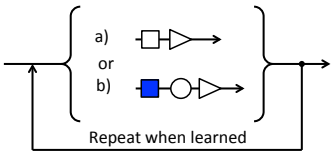

Figure 4: Patterns of experience that we expect the system to develop. The curly brackets represent a conditional scheme that results in sequence (a) or (b) depending on the current state of the RS. Once the IS has learned this scheme, it applies it indefinitely.

As illustrated in Figure 4, we expect the IS to learn to use *square* experiences to inform subsequent behavior. If an intended *square* experience yields a *white square*, the system should subsequently experience *white triangle* because it has a positive valence (Figure 4-a, exploiting Regularity 3 of Figure 3). If the intended *square* experience yields a *blue square*, the system should subsequently experience *white circle* so that it can then experience *white triangle* (Figure 4-b, exploiting Regularities 8 and then 5). After this, we expect the IS to intend a *square* experience again and to repeat the same conditional choice again (Figure 4-a or 4-b again). In a previous study, we presented a hierarchical sequence-learning algorithm that learned this kind of behaviors [4, 5].

In this paper, we introduce a new algorithm that allows the IS to explicitly interpret the patterns of behavior in Figure 4 as if this behavior was made possible by the existence of phenomena in the RS. Indeed, this behavior can be interpreted as if the IS used *square* experiences to observe the state of the RS. If this observation results in a *white square*, the IS can consider that a certain kind of phenomenon—which we call a *white phenomenon*—exists in the RS. A *white phenomenon* affords the *white triangle* experience. Since there is no sequential regularity based on *white triangle*, the IS can consider that a *white triangle* experience causes the disappearance of the *white phenomenon*, leaving the RS in a state unknown to the IS (as if the IS ate the *white phenomenon*, and enjoyed eating it, since *white triangle* has a positive valence). A *blue square* experience similarly denotes the presence of a *blue phenomenon*, which affords the *blue triangle* experience (eating the *blue phenomenon*, and disliking it, since *blue triangle* has a negative valence). *Blue phenomena* also afford the *white circle* experience, which appears to transform *blue phenomena* into delicious *white phenomena*. Overall, we expect the IS to construct the belief system represented in Figure 5.

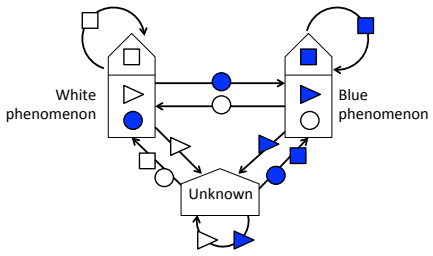

Figure 5: Petri net representing the belief system that we expect the IS to construct. Nodes represent beliefs that the IS may hold about the RS's state: *white phenomenon* (top-left), *blue phenomenon* (top-right), or *unknown* (bottom). Phenomena are represented by pointed rectangles split in two areas: upper area: the persistent experience that allows observing this phenomenon; lower area: the sporadic (not persistent) experiences afforded by this phenomenon. Arcs show the sporadic experiences that cause the IS to transition from one belief state to another.

3 Algorithm

The problem of reconstructing causality from sequences of events has been studied in process mining (PM). For example, Van der Aalst et al. [6] surveyed four PM algorithms used to construct Petri nets from event logs. Such surveys show that PM algorithms generally exploit specific properties of the flow of events and of the Petri net to be constructed. Typically, Van der Aalst's [7] foundational α-algorithm exploits the assumption that the Petri net has no internal loops. Our system, however, does not satisfy this assumption.

Our algorithm was inspired by the α-algorithm but exploits two different assumptions: (a) the system is actively generating the events, rather than learning from logs generated beforehand; and (b) there exist "self-loops" (with same starting and ending node). In essence, our algorithm begins with looking for experiences that can be repeated several times in a row (persistent experiences). When the IS finds a candidate persistent experience, it creates a self-loop based on this experience, and a node attached to this self-loop. This node thus represents a hypothetical phenomenon that can be consistently observed through this experience. Next, the IS learns arcs between nodes. Table 1 summarizes this algorithm at the highest level.

Table 1: A proof-of-concept algorithm to infer phenomena from regularities of experience

```
01   belief ← "unknown", mood ← "curious"
02   Loop
03      if mood = "curious"
04         intention ← leastTriedExperience(Belief)
05      if mood = "hedonist"
06         intention ← intentionWithMaxExpectedOutcomeValence(belief)
07      if mood = "excited"
08         intention ← experience
09      experience ← ReactiveSubsystem(intention)
10      if mood = "excited"
11         if experience ≠ intention
12            intention is sporadic
13         else if excitement > excitementThreshold
14            experience is persistent
15            createNewBeliefState(experience)
16         else
17            excitement++
18      belief ← updateAndGetBelief(experience)
19      if mood ≠ "excited"
20         mood ← "hedonist"
21      if experience is unknown
22         mood ← "excited"
23      if all the experiences have not been tried yet in the context of this belief
24         mood ← "curious"
```

The IS has three possible motivational states (moods): *curious*, *excited*, and *hedonist*. On initialization, it has only one predefined belief state: *unknown*, and it is in a *curious* mood. Lines 03 and 04: if it is in a curious mood, then it intends the experience that has been the least tried in the context of the current belief. Lines 05 and 06: if it is in a hedonist mood, then it intends the experience that has the maximum expected valence in the context of the current belief. Lines 07 and 08: if it is in an excited mood, then it intends to repeat the same experience.

Line 09: The RS computes the actual experience from the intended experience (see Section 4).

Lines 10 to 17: if the IS is excited and the intention failed, the intended experience is marked *sporadic* (not persistent). Otherwise, if the IS has reached a preset excitement threshold, the experience is marked *persistent*, and a new belief state is created based on this experience, otherwise the IS gets more excited. Line 18: learn regularities and update the current belief state. Lines 19 to 24:

if the mood is not *excited* then it is *hedonist*. If the experience has neither yet been marked *persistent* or *sporadic*, the mood becomes excited. If the IS has not yet experimented much with the current belief state, the mood is curious.

Figure 6 reports an empirical evaluation of this algorithm based upon the trace of its stream of experience.

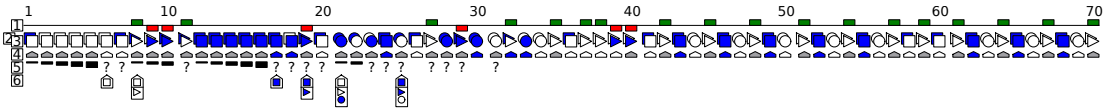

Figure 6: Our system's first 70 experiences. Line 1: valence of the actual experience as in Figure 1. Line 2: intended experience. Line 3: actual experience. Line 4: current belief: *unknown* / *white phenomenon* / *blue phenomenon* (little grey/white/blue pointed rectangles). Line 5: mood: *curious* ("?"), *excited* (black bar-graph), or *hedonist* (blank). Line 6: progressive construction of phenomena. Step 1: the IS intends a *blue square* and obtains a *white square*. Since the *white square* experience is neither yet marked *sporadic* or *persistent*, the IS gets excited (black bar in Line 5). Step 2 to 5, the *white square* keeps succeeding and the IS gets increasingly excited. Step 6: the IS reaches the excitement threshold; it marks the *white square* persistent, and creates the *white phenomenon*; Line 4: the current belief becomes *white phenomenon*; Line 6: thus far, the *white phenomenon* contains only the *white square* experience; Line 5: the IS becomes curious to play with the newly created *white phenomenon*. Step 7: the IS tries *blue square*. Step 8: the IS tries *blue triangle* and obtains *white triangle*; Line 6: the IS notes that the *white phenomenon* affords *white triangle*. Step 9: *white triangle* is learned to be *sporadic* since it failed on the second attempt. Step 11: similar, *blue triangle* is *sporadic*. Step 17: *blue square* is *persistent* and the IS creates the *blue phenomenon*. Step 21: the *white phenomenon* is complete. Step 25: the *blue phenomenon* is complete; the IS continues learning the arcs of the Petri net (function *updateAndGetBelief()*, not examined in this paper). From Step 41 on, the system exhibits the behavior described in Figure 4, demonstrating that it has learned the belief system that we expected it to learn.

4 The agent/environment coupling

The environment is implemented as a subprogram of the *ReactiveSubsystem()* function (Table 1, Line 09). The rest of the main loop implements the agent. Unbeknownst to the agent, the environment is an arbitrary string of 10 digits initialized with E_0=[1, 7, 3, 2, 9, 3, 5, 6, 7, 8], plus an integer p in the interval [0, 9], initialized with $p_0 = 0$, which represents the agent's position in this string. This environment is similar to Georgeon & Hassas's [5] *String Environment*.

Triangle experiences consist of moving the agent to the next digit (if $p < 9$ then $p \leftarrow p + 1$ else $p \leftarrow 0$). If this takes the agent to a lower digit then it yields *blue triangle*, otherwise, *white triangle*. Note that triangle experiences are uninformative of the resulting situation of the agent; the agent's input data is not necessarily a representation of the current state of the environment. *Square* experiences consist of testing whether the next digit is lower than the current one, if yes it yields *blue square*, otherwise, *white square*. *Circle* experiences consist of swapping the current digit with the next (only if $p < 9$); if the resulting current digit is lower than the next, it yields *blue circle*, otherwise, *white circle*.

The trace in Figure 6 shows that, over time, the agent learns to use *square* experiences as active perception to test whether the current digit is lower than the next, and, if yes, to use a *circle* experience to swap the digits, in order to always move to a greater or equal next digit (because the agent has an innate preference for moving towards greater or equal digits implemented through the positive valence of the *white triangle* experience).

5 Conclusion

This proof-of-concept algorithm illustrates how an agent can construct knowledge from regularities observed in its stream of experience—a question which theoreticians and philosophers of mind consider crucial to cognition. All along, the agent ignores the underlying reality in which it exists (noumenal reality: the string of digits); it only constructs its own interpretation of this reality in the form of phenomenal knowledge (*white* and *blue* phenomena).

We proposed a design paradigm that focuses on the system's stream of experience rather than separating perception from action. This design paradigm does not contradict the traditional perception/action paradigm; the relation between the two is examined in Section 4. Yet, we found that it facilitated the conception and the explanation of the algorithm. Additionally, it complies with theories of embodied cognition and enaction [e.g., 8] in that the agent is an active observer of its environment, and constructs knowledge from sensorimotor schemes of experience. We develop these arguments further in previous papers [e.g., 3, 9].

Our agent has three motivational drives: curiosity, excitement, and hedonism. These drives implement inborn preferences analogous to natural organisms' preferences that presumably evolved through natural selection (e.g., playing, liking ingesting nutritive food and disliking ingesting toxic food). Such inborn preferences relate to Jonas's [10] notion of *needful freedom*; they drive the agent's behavior without imposing predefined goal states.

This preliminary study led us to formulate crucial questions that remain to be answered if we want to design agents that can construct higher-level knowledge from their experience interacting with a more complex reality (e.g., the real world): how to revise erroneous hypothetical phenomena; how to learn to interact with an environment in which several phenomena can exist simultaneously at different locations (i.e., spatial environment); how to distinguish between phenomena that afford some experiences in common but differ by some others; how to learn complex spatio-sequential patterns of interactions afforded by phenomena; and how to handle noise in the agent's input data.

References

[1] Russell, S., & Norvig, P. (2003), Artificial Intelligence, A Modern Approach. Prentice Hall.

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

[3] Georgeon, O., Marshall, J. & Manzotti, R. (2013). ECA: An enactivist cognitive architecture based on sensorimotor modeling. Biologically Inspired Cognitive Architectures 6: 46-57.

[4] Georgeon, O. & Ritter, F. (2012). An intrinsically-motivated schema mechanism to model and simulate emergent cognition. Cognitive Systems Research 15-16: 73-92.

[5] Georgeon, O. & Hassas, S. (2013). Single Agents Can Be Constructivist too. Constructivist Foundations 9(1): 40-42.

[6] Van der Aalst, W., Van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Wei- jters, A. (2003). Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47(2), 237–267.

[7] Van der Aalst, W. & Weijters, A. & Maruster, L (2003). Workflow Mining: Discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering, vol 16.

[8] Froese, T. & Ziemke, T. (2009). Enactive artificial intelligence: Investigating the systemic organization of life and mind. Journal of Artificial Intelligence, 173(3-4), pp 466–500.

[9] Georgeon, O. & Cordier, A. (2014). Inverting the interaction cycle to model embodied agents. Procedia Computer Science, 41, pp 243-248. The Fifth international conference on Biologically Inspired Cognitive Architectures. Boston, MA.

[10] Jonas, H. (1966), The Phenomenon of Life: Toward a Philosophical Biology, Evanston, Illinois: Northwestern University Press, 2001.