Berthouze, L., Prince, C. G., Littman, M., Kozima, H., and Balkenius, C. (2007).
Proceedings of the Seventh International Conference on Epigenetic Robotics: Modeling
Cognitive Development in Robotic Systems. *Lund University Cognitive Studies, 135*.

Constructivist Anticipatory Learning Mechanism (CALM) – dealing with partially deterministic and partially observable environments

Filipo Studzinski Perotto
Instituto de Informática
UFRGS
Porto Alegre, Brasil
fsperotto@inf.ufrgs.br

Jean-Christophe Buisson
IRIT
ENSEEIHT-INPT
Toulouse, France
buisson@enseeiht.fr

Luis Otávio Alvares
Instituto de Informática
UFRGS
Porto Alegre, Brasil
alvares@inf.ufrgs.br

Abstract

This paper presents CALM (Constructivist Anticipatory Learning Mechanism), an agent learning mechanism based on a constructivist approach. It is designed to deal dynamically and interactively with environments which are at the same time partially deterministic and partially observable. We describe in detail the mechanism, explaining how it represents knowledge, and how the learning methods operate. We analyze the kinds of environmental regularities that CALM can discover, trying to show that our proposition follows the way towards the construction of more abstract or high-level representational concepts.

1. Introduction

The real world is a very complex environment, and the transition from ***sensorimotor intelligence*** to ***symbolic intelligence*** is an important aspect to explaining how human beings successfully deal with it (Piaget 1957). The problem is the same for a situated artificial agent (like a robot), who needs to incrementally learn the observed regularities by interacting with the world.

In ***complex environments*** (Goldstein 1999), special 'macroscopic' properties emerge from the functional interactions of 'microscopic' elements, and generally these emergent characteristics are not present in any of the sub-parts that generate it. The salient phenomena in this kind of environment tend to be related to ***high-level*** objects and processes (Thornton 2003), and in this case it is plainly inadequate to represent the world only in terms of primitive sensorimotor terms (Drescher 1991).

An intelligent agent (human or artificial) who lives in these conditions needs to have the possibility to overpass the limits of direct sensorial perceptions, organizing the universe in terms of **more abstract concepts**. The agent needs to be able to detect high-level regularities in the environment dynamics, but it is not possible if it is closed into a rigid 'representational vocabulary'.

The purpose of this paper is to present an agent learning architecture, inspired in a constructivist conception of intelligence (Piaget 1957), capable of creating a model to describe its universe, and using abstract elements to represent unobservable properties.

The paper is organized as follows: Section 2 and 3 describe both the agent and the environment conceptions. Section 4 and 5 show the basic CALM mechanism, respectively detailing how to represent the knowledge, and how to learn it. Section 6 presents the way to deal with hidden properties, showing how these properties can be discovered and predicted through synthetic elements. Section 7 presents example problems and solutions following the proposed method. Section 8 compares related works, and section 9 finalizes the paper, arguing that this is an important step towards a more abstract representation of the world, and pointing some next steps.

2. Agent and Environment

The concepts of *agent* and *environment* are mutually dependent, and they need to be defined one in relation to the other. In this work, we adopt the notions of situated agent and properties based environment.

A ***situated agent*** is an entity embedded in and part of an environment, which is only partially observable through its sensorial perception, and only partially liable to be transformed by its actions (Suchman 1987). Due to the fact that sensors will be limited in some manner, a situated agent can find itself unable to distinguish between differing states of the world. A situation could be perceived in different forms, and different situations could seem the same. This ambiguity in the perception of states, also referred to as ***perceptual aliasing***, has serious effects on the ability of most learning algorithms to construct consistent knowledge and stable policies (Crook and Hayes 2003).

An agent is supposed to have motivations, which in some way represent its goals. Classically, the machine learning problem means enabling an agent to autonomously construct polices to maximize its goal reaching performance. The ***model based strategy*** separates the problem in two parts: (a) construct a world model and, based on it, (b) construct a policy of actions.

CALM (Constructivist Anticipatory Learning Mechanism) responds to the task of constructing a ***world model***. It tries to organize the sensorial

information in a way to represent the regularities in the interaction of the agent with the environment.

There are two common ways to describe an environment: either based on *states*, or based on *properties*. A ***states based environment*** can be expressed by a generalized states machine, frequently defined as POMDP (Singh et al. 2003) or as a FSA (Rivest and Shapire, 1994). We define it as $\epsilon = \{Q, A, O, \delta, \gamma\}$ where Q is a finite not-empty set of underlying states, A is a set of agent actions, O is a set of agent observations, $\delta : Q \times A \rightarrow Q$ is a transition function, which describes how states change according to the agent actions, and $\gamma : Q \rightarrow O$ is an observation function, which gives some perceptive information related to the current underlying environment state.

A ***properties based environment*** can be expressed by $\xi = \{F, A, \tau\}$ where F is a finite not-empty set of properties, composed by $F_{(p)}$, the subset of perceptible or observable properties, and $F_{(h)}$, the subset of hidden or unobservable properties, A is a set of agent actions, and $\tau_{(i)}$: $F_1 \times F_2 \times ... \times F_k \times A \rightarrow F_i$ is a set of transformation functions, one for each property F_i in F, describing the changes in property values according to the agent actions.

The environment description based on properties (ξ) has some advantages over the description based in states (ϵ) in several cases. Firstly because it promotes a **clearer relation** between the environment and the agent perception. In general we assume that there is one sensor to each observable property. Secondly, because the *state identity* can be distributed in the properties. In this way, it is possible to represent 'generalized states' and, consequently, **compact descriptions** of the transformation functions, generalizing some elements in the function domain, when they are not significant to describe the transformation. A discussion that corroborates our assumptions can be read in (Triviño-Rodriguez and Morales-Bueno 2000).

The most compact expression of the transition function represents the environment ***regularities***, in the form $\lambda^*_{(i)}$: $F_1|\varepsilon \times F_2|\varepsilon \times ... \times F_k|\varepsilon \times A|\varepsilon \rightarrow F_i$. This notation is similar to that used in grammars, and it means that at each property j, we can consider it for the domain or not ($F_j|\varepsilon$).

3. Types of Environments

We adopt the properties based description (ξ), defining 3 axis to characterize different **types of environments**. The axis ∂ represents the environment determinism in the transformations, the axis ω indicates the perceptive accessibility that the agent has to the environment, and the axis φ represents the information gain related to the properties.

The ***determinism axis level*** (∂) is equivalent to the proportion of deterministic transformations in τ in relation to the total number of transformations. So, in the **completely non-deterministic case** ($\partial = 0$), the transformation function (τ) to any property i needs to be represented as $F_1 \times F_2 \times ... \times F_k \times A \rightarrow \Pi(F_i)$, where $\Pi(F_i)$ is a probabilistic distribution. On the other hand, in the **completely deterministic case** ($\partial = 1$), every transformation can be represented directly by $F_1 \times F_2 \times ... \times F_k \times A \rightarrow F_i$. An environment is said ***partially deterministic*** if it is situated between these two axis extremities ($0 < \partial < 1$). When $\partial = 0.5$, for example, half of the transformations in τ are deterministic, and the other half is stochastic.

It is important to note that a single transition in the function δ of an environment represented by states (ϵ) is equivalent to k transformations in the function τ of the same environment represented by properties (ξ). So, if only one transformation that integrates the transition is non-deterministic, all the transition will be non-deterministic. Conversely, a non-deterministic transition can present some deterministic component transformations. This is another advantage of using the properties representation, when we combine it with a learning method based on the discovery of deterministic regularities.

The ***accessibility axis level*** (ω) represents the degree of perceptive access to the environment. It is equivalent to the proportion of observable properties in F in relation to the total number of properties.

If $\omega = 1$ then the environment is said **completely observable**, which means that the agent has sensors to observe directly all the environment properties. In this case there is no perceptual confusion, and the agent always knows what is its current situation. If $\omega < 1$, then the environment is said ***partially observable***. The lesser ω, the higher the proportion of hidden properties. When ω is close to 0, the agent is no longer able to identify the current situation only in terms of its perception.

In other words, partially observable environments present some determinant properties to a good world model, which cannot be directly perceived by the agent. Such environments can appear to be arbitrarily complex and non-deterministic on the surface, but they can actually be deterministic and predictable with respect to unobservable underlying elements (Holmes and Isbell 2006).

There is a dependence relation between these two axis – accessibility and determinism. The more an agent has sensors to perceive complex elements and phenomena, the more the environment will appear deterministic to it.

Finally, the ***informational axis level*** (φ) is equivalent to the inverse of the average number of generalizable properties to represent the environment regularities (λ^*), divided by the total number of properties in F. The greater is φ (rising to 1), the more compactly the transformation function (τ) can be expressed in terms of regularities (λ^*).

In other words, higher levels of φ mean that the information about the environment dynamics is concentrated in the properties (i.e. there is just a small sub-set of highly relevant properties for each prediction), and lower levels of φ indicate that the information about the dynamics is fuzzily distributed over all the set of properties, and in this case the agent needs to describe the transformation in function of almost all properties.

Learning methods based on the discovery of regularities can be very efficient in environments where the properties are highly informative.

4. The Basic Idea

In this section we present the basic CALM mechanism, which is developed to model its interaction with a completely observable but partially deterministic environment (COPDE), where ω=1, but ∂<1 and φ<1.

CALM tries to construct a set of schemas to represent perceived regularities occurring in the environment through its interactions. Each ***schema*** represents some regularity checked by the agent during its interaction with the world. It is composed by three vectors: Ξ = (context + action → expectation). The context and expectation vectors have the same length, and each of their elements are linked with one sensor. The action vector is linked with the effectors.

In a specific schema, the **context** vector represents the set of equivalent situations where the schema is applicable. The **action** vector represents a set of similar actions that the agent can carry out in the environment. The **expectation** vector represents the expected result after executing the given action in the given context. Each element vector can assume any value in a discrete interval defined by the respective sensor or effector.

Some elements in these vectors can undertake the **undefined value**. For example, an element linked with a binary sensor must have one of three values: true, false or undefined (represented, respectively, by '1', '0' and '#'). In both the context and action vectors, '#' represents something ignored, not relevant to make the anticipations. But for the expectation vector, '#' means that the element is not deterministically predictable.

The undefined value generalizes the schema because it allows to ignore some properties to represent a set of situations. There is **compatibility** between a schema and a certain situation when the schema's context vector has all defined elements equal to those of the agent's perception. Note that compatibility does not compare the undefined elements. For example, a schema which has the context vector = '100#' is able to assimilate the compatible situations '1000' or '1001'.

The use of undefined values makes possible the construction of a **schematic tree**. Each node in that tree is a schema, and relations of generalization and specialization guide its topology (quite similar to decision trees or discrimination trees). The root node represents the most generalized situation, which has the context and action vectors completely undefined. Adding one level in the tree is to specialize one generalized element, creating a branch where the undefined value is replaced by different defined values. This specialization occurs either in the context vector or in the action vector. The structure of the schemas and their organization as a tree are presented in Figure 1.

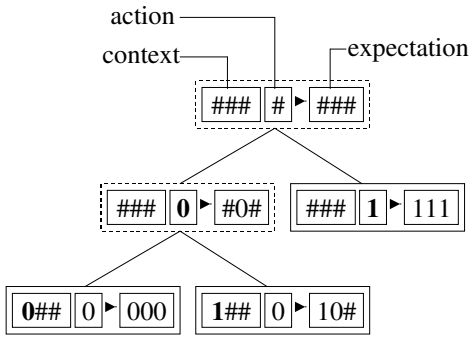

Figure 1. Schematic Tree. Each node is a schema composed of three vectors: context, action and expectation. The leaf nodes are decider schemas.

The context in which the agent is at a given moment (perceived through its sensors) is applied in the tree, **exciting** all the schemas that have a compatible context vector. This process defines a set of excited schemas, each one suggesting a different action to do in the given situation. CALM will choose one to **activate**, and then the action proposed by the activated schema will be performed through the agent's effectors. The algorithm always chooses the compatible schema that has the most specific context, called **decider schema**, which is the leaf of a differentiated branch. Each decider has a kind of episodic memory, which represents (in a generalized form) the specific and real situations experimented in the past, during its activations.

5. Learning Methods

The **learning process** happens through the refinement of the set of schemas. The agent becomes more adapted to its environment as a consequence of that. After each experienced situation, CALM checks if the result (context perceived at the instant following the action) is in conformity to the expectation of the activated schema. If the anticipation fails, the error between the result and the expectation serves as parameter to correct the tree or to adjust the schema.

CALM combines top-down and bottom-up learning strategies. In the schematic tree topology, the context and action vectors are considered together. This concatenated vector identifies the node in the tree, which grows up using the top-down strategy.

The agent has just one **initial schema**. This root schema has the context vector completely general

(without any differentiation, ex.: '#####') and expectation vector totally specific (without any generalization, ex.: '01001'), created at the first experienced situation, as a mirror of the result directly observed after the action.

The **context** vector will be gradually specialized by differentiation. In more complex environments, the number of features the agent senses is huge, and, in general, only a few of them are relevant to identify the situation class ($1 > \varphi >> 0$). In this case, a **top-down strategy** seems to be better, because there is a shorter way beginning with an empty vector and searching for these few relevant features to complete it, than beginning with a full vector and having to eliminate a lot of useless elements.

Selecting the good set of relevant features to represent some given concept is a well known problem in AI, and the solution is not easy, even to approximated approaches. As it will be seen, we adopt a kind of forward greedy selection (Blum and Langley 1997).

The expectation vector can be seen as a label in each decider schema, and it represents the predicted **anticipation** when the decider is activated. The evolution of **expectations** uses a **bottom-up strategy**. Initially all different expectations are considered as different classes, and they are gradually generalized and integrated with others. The agent has two alternatives when the expectation fails. In a way to make the knowledge compatible with the experience, the first alternative is to try to divide the scope of the schema, creating new schemas, with more specialized contexts. Sometimes it is not possible and the only way is to reduce the schema expectation.

Three basic methods compose the CALM learning function, namely: *differentiation*, *adjustment* and *integration*. **Differentiation** is a necessary mechanism because a schema responsible for a context too general can hardly make precise anticipations. If a general schema does not work well, the mechanism divides it into new schemas, differentiating them by some element of the context or action vector.

In fact, the differentiation method takes an unstable decider schema and changes it into a two level sub-tree. The parent schema in this sub-tree preserves the context of the original schema. The children, which are the new decider schemas, have their context vectors a little bit more specialized than their parent. They attribute a value to some undefined element, dividing the scope of the original schema. Each one of these new deciders engages itself in a part of the domain. In this way, the previous correct knowledge remains preserved, distributed in the new schemas, and the discordant situation is isolated and treated only in its specific context. Differentiation is the method responsible to make the schematic tree grows up. Each level of the tree represents the introduction of some constraint into the context vector. Figure 2 illustrates the differentiation process.

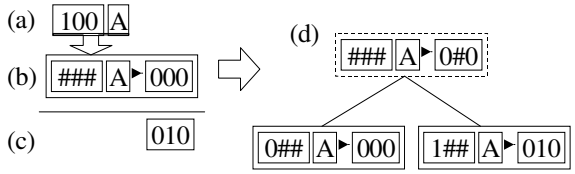

Figure 2. Differentiation method; (a) experimented situation and action; (b) activated schema; (c) real observed result; (d) sub-tree generated by differentiation.

The algorithm needs to choose what will be the differentiator element, and it could be from either the context vector or the action vector. This differentiator needs to separate the situation responsible for the disequilibrium from the others, and the algorithm chooses it by calculating the information gain.

When some schema fails and it is not possible to differentiate it, then CALM executes the **adjustment** method. This method reduces the expectations of an unstable decider schema in order to make it reliable again. The algorithm simply compares the activated schema's expectation and the real result perceived by the agent after the application of the schema, setting the incompatible expectation elements to undefined value ('#').

As CALM always creates schemas with expectations totally determined (as a mirror of the result of its first application), the walk performed by the schema is a reduction of expectations, up to the point it reaches a state where remains only those elements that really represent the **regular results** of the action carried out in that context. Figure 3 illustrates it with an example.

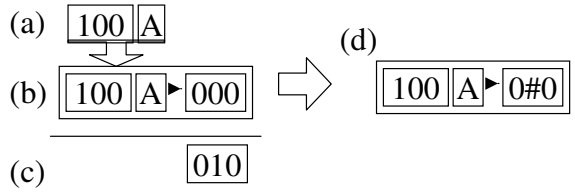

Figure 3. Adjust method; (a) experimented situation and action; (b) activated schema; (c) real observed result; (d) schema expectation reduction after adjustment.

The adjustment method changes the schema expectation (and consequently the anticipation predicted by the schema). Successive adjustments can reveal some unnecessary differentiations. After an adjustment, CALM needs to verify the possibility to regroup some related schemas. It is the **integration** method that searches two schemas with equivalent expectations to approach different contexts in a same sub-tree, and join these schemas into a single one, eliminating the differentiation. The method is illustrated in figure 4.

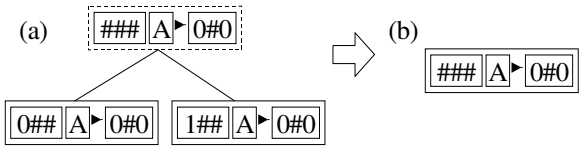

Figure 4. Integration method; (a) sub-tree after some adjustment; (b) an integrated schema substitutes the sub-tree.

To test this basic CALM method, we have made some **experiments** in simple scenarios showing that the agent converges to the expected behavior, constructing correct knowledge to represent the environment deterministic regularities, as well as the regularities of its body sensations, and also the regular influence of its actions over both. We may consider that these results have corroborated the mechanism ability to discover regularities and use this knowledge to adapt the agent behavior. The agent has learned about the consequences of its actions in different situations, avoiding emotionally negative situations, and pursuing those emotionally positive. A detailed description of these experiments can be viewed in (Perotto and Alvares 2006).

6. Dealing with the Unobservable

In this section we will present the **extended mechanism**, developed to deal with partially observable and partially deterministic environments (**CALM-POPDE**), where $\partial<1$, $\varphi<1$, and also $\omega<1$.

In the **basic mechanism (CALM-COPDE)**, presented in previous sections, when some schema fails, the first alternative is to differentiate it based on direct sensorimotor (context and action) elements. If it is not possible to do that, then the mechanism reduces the schema expectation, generalizing the incoherent anticipated elements. When CALM reduces the expectation of a given schema, it supposes that there is no deterministic regularity following the represented situation in relation to these incoherent elements, and the related transformation is unpredictable.

However, sometimes the error could be explained by considering the existence of some **abstract or hidden property** in the environment, which could be able to differentiate the situation, but which is not directly perceived by the agent sensors. In the extended mechanism, we introduce a new method which enables CALM to suppose the existence of a non-sensorial property in the environment, which it will represent as a **synthetic element**.

When a new synthetic element is created, it is included as a new term in the context and expectation vectors of the schemas. Synthetic elements suppose the existence of something beyond the sensorial perception, which can be useful to explain non-equilibrated situations. They have the function of amplifying the differentiation possibilities.

In this way, when dealing with partially observable environments, CALM has two additional challenges: a) infer the existence of unobservable properties, which it will represent by synthetic elements, and b) include these new elements into its predictive model. A good strategy to do this task is looking at the historical information. Holmes and Isbell (2006) have proved that it is always possible to find sufficient little pieces of history to distinguish and identify all the underlying states in D-POMDPs.

The first CALM-POPDE additional method is called **abstract differentiation**. When a schema fails in its prediction, and when it is not possible to differentiate it by the current set of considered properties, then a new boolean synthetic element is created, enlarging the context and expectation vectors. Immediately, this element is used to differentiate the incoherent situation from the others. The method attributes arbitrary values to this element in each differentiated schema. These values represent the presence or absence of some unobservable condition, necessary to determine the correct prediction in the given situation. The method is illustrated in figure 5, where the new elements are represented by card suites.

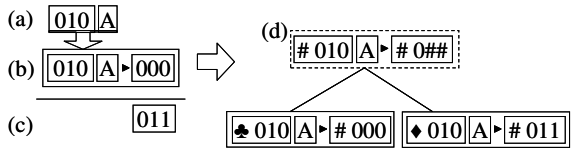

Figure 5. Synthetic element creation method; (d) incremented context and expectation vectors, and differentiation using synthetic element.

Once a synthetic element is created, it can be used in next differentiations. A new synthetic element will be created only if the existing ones are already saturated. To avoid the problem of creating infinite new synthetic elements, CALM can do it only until a determined limit, after which it considers that the problematic anticipation is simply unpredictable, undefining the expectation in the related schemas by adjustment.

The synthetic element is not associated to any sensorial perception. Consequently, its value cannot be observed. This fact can place the agent in ***ambiguous situations***, where it does not know whether some relevant but not observable condition (represented by this element) is present or absent.

Initially, the value of a synthetic element is verified *a posteriori* (i.e. after the execution of the action in an ambiguous situation). Once the action is executed and the following result is verified, then the agent can rewind and deduce what was the situation really faced in the past instant (disambiguated). Discovering the value of a synthetic element after the circumstance where this information was needed can seem useless, but in fact, this delayed deduction gives information to

the second CALM-POPDE additional method, called **abstract anticipation**.

If the unobservable property represented by this synthetic element has a regular behavior, then the mechanism can "backpropagate" the deduced value for the activated schema in the previous instant. The deduced synthetic element value will be included as a new anticipation in the previous activated schema.

For **example**, in time t_1 CALM activates the schema $\Xi_1 = (\#0 + x \rightarrow \#1)$, where the context and expectation are composed by two elements (the first one synthetic and the second one perceptive), and one action. Suppose that the next situation '*#1*' is ambiguous, because it excites both schemas $\Xi_2 = (\clubsuit 1 + x \rightarrow \#0)$ and $\Xi_3 = (\blacklozenge 1 + x \rightarrow \#1)$. At this time, the mechanism cannot know the synthetic element value, crucial to determine what is the real situation. Suppose that, anyway, the mechanism decides to execute the action '*x*' in time t_2, and it is followed by the sensorial perception '*0*' in t_3. Now, in t_3, the agent can deduce that the situation really dealt with in t_2 was '*♣1*', and it can include this information into the schema activated in t_1, in the form $\Xi_1 = (\#0 + x \rightarrow \clubsuit 1)$.

7. Example Problem and Solution

To exemplify the functioning of the proposed method we will use the **flip problem**, which is also used by (Singh et al. 2003) and (Holmes and Isbell 2006). They suppose an agent who lives in a two states universe. It has 3 actions (*l*, *r*, *u*) and 2 perceptions (*0*, *1*). The agent do not have any direct perception to know what is the underlying current state. It has the perception *1* when the state changes, and the perception *0* otherwise. Action *u* keeps the state the same, action *l* causes the deterministic transition to the left state, and action *r* causes the deterministic transition to the right state. The flip problem is showed as a Mealy machine in figure 6.

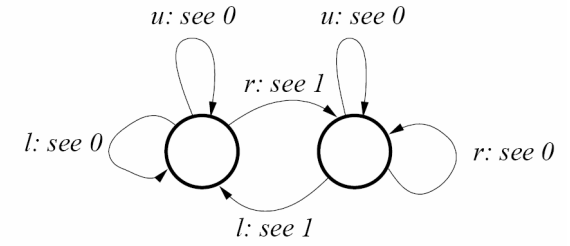

Figure 6. The flip problem.

CALM-POPDE is able to solve this problem. Firstly it will try to predict the next observation in function of its action and current observation. However, CALM quickly discovers that the perceptive observation is not useful to the model, and that there is not sufficient information to make correct anticipations. So, it creates a new synthetic element which will be able to represent the underlying left (♣) and right (♦) states.

Figure 7 shows the first steps in the schematic tree construction for the flip problem. We suppose that the first movements do not betray the existence of a hidden property. These movements are: "*r*1, *u*0, *l*1, *r*1, *l*1, *u*0, *r*1". Figure 8 shows the first abstract differentiation, after the sequence "*r*0", and also the abstract anticipation, that refers to the immediately previous sequence ("*r*1"). Figure 9 shows the abstract anticipation coming from the repetition of "*r*0". Figure 10 shows a new abstract differentiation and its anticipations by following "*l*1, *l*0, *l*0". Finally, figure 11 shows the final solution, with the last differentiation resulting from the execution of "*u*0, *l*0, *r*1, *u*0, *r*0".

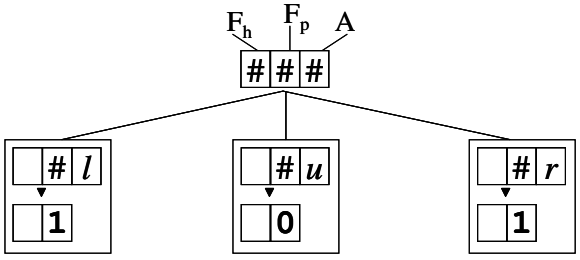

Figure 7. Initial schematic tree to the flip problem. The vector represents synthetic elements (Fh), perceptible elements (Fp) and actions (A). The decider schemas show the expectations.

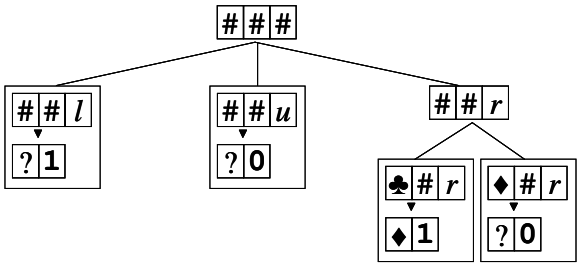

Figure 8. First abstract differentiation.

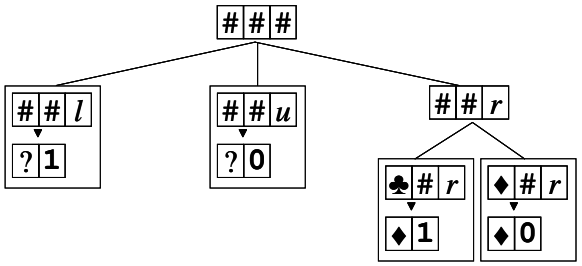

Figure 9. Abstract anticipation.

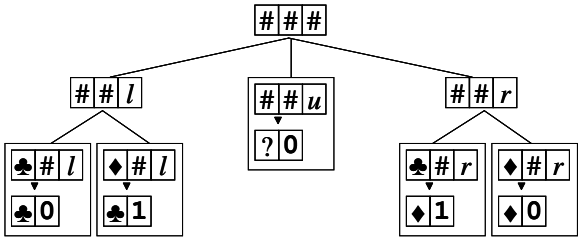

Figure 10. New abstract differentiations and anticipations.

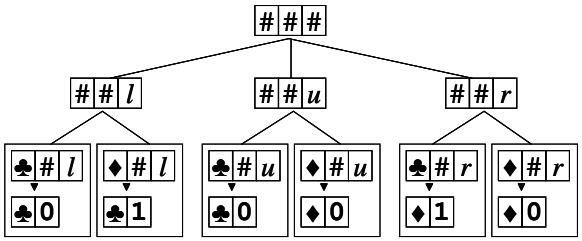

Figure 11. Final schematic tree to solve the flip problem.

In a second problem, we consider a robot which have the mission of buying some cans in a **drink machine**. It has 3 alternative actions: "insert a coin" (*i*), "press the button" (*p*), or "go to another machine" (*g*); it can see the state of an indicator light in the machine: "off" (○) or "on" (ⓘ); and it perceives whether a can is returned (☺) or not (☹). There are 2 hidden properties: "no coin inserted" (❼) or "coin inserted" (❶); and "machine ok" (☑) or "machine out of service" (☒). The light turns on (ⓘ) just during one time cycle, and only if the agent presses the button without having inserted a coin before, otherwise the light indicator is always off. The goal in this problem is to take a determined number of drinks without losing coins in bad machines.

This example poses two challenges to the agent: First, the machine does not present any direct perceptible change when the coin is inserted. Since the agent does not have any explicit memory, apparently it faces the same situation both before and after having inserted the coin. However, this action changes the value of an internal property in the drink machine.

Precisely, the disequilibrium occurs when the agent presses the button. In an instantaneous point of view, sometimes the drink arrives, and the goal is attained (○☹ + *p* → ○☺), but sometimes only the led light turns on (○☹ + *p* → ⓘ☹). To reach its goal, the agent needs to coordinate a chain of actions (insert the coin and press the button), and it can do that by using a synthetic element which represents this machine internal condition (❼ or ❶).

Second, the agent does not have direct perceptive indications to know if the machine is working, or if it is out of service. The agent needs to interact with the machine to discover its operational condition (☑ or ☒). This second problem is a little bit different from the first, but it can be solved by the same way. The agent creates a test action, which enables it to discover this hidden property before inserting the coin. It can do that by pressing the button.

Table 1 presents the set of decider schemas that CALM learns to the drink machine problem. We remarks that Ξ_4 presents the unpredictable transformation that follows the action *g* (go to another machine) due to the uncertainty of the operational state of the next machine. The test is represented in Ξ_1 and Ξ_2, that can be simultaneously activated because the ambiguity of the result that follows the activation of Ξ_4 but anticipates the operational state of the machine.

Table 1. Schemas to the drink machine problem.

Ξ_1 = (	☒ #	# #	+	*p*	→	☒❼ ○☹)
Ξ_2 = (	☑❼	# #	+	*p*	→	☑❼ ⓘ☹)
Ξ_3 = (	☑❶	# #	+	*p*	→	☑❼ ○☺)
Ξ_4 = (	# #	# #	+	*g*	→	#❼ ○☹)
Ξ_5 = (	☒ #	# #	+	*i*	→	☒❼ ○☹)
Ξ_6 = (	☑ #	# #	+	*i*	→	☑❶ ○☹)

8. Related Works

CALM-POPDE is an original mechanism that enables an agent to incrementally create a world model during the course of its interaction. This work is the continuity of our previous work (Perotto and Alvares 2006), extended to deal with partially observable environments.

The pioneer work on Constructivist AI has been presented by **Drescher** (1991). He proposed the first constructivist agent architecture (called ***schema mechanism***), that learns a world model by an exhaustive statistical analysis of the correlation between all the context elements observed before each action, combined with all resulting transformations. Drescher has also suggested the necessity to discover hidden properties by creating 'synthetic items'.

The schema mechanism represents a strongly coherent model, however, there are no theoretical guarantees of convergence. Another restriction is the computational cost of the kind of operations used in the algorithm. The need of space and time resources increases exponentially with the problem size. Nevertheless, many other researchers have presented alternative models inspired by Drescher, like as (Yavuz and Davenport 1997), (Birk and Paul 2000), (Morrison et al. 2001), (Chaput 2004) and (Holmes and Isbell 2005).

Our mechanism (CALM) differs from these previous works because we limit the problem to the discovery of deterministic regularities (even in partially deterministic environments), and in this way, we can implement direct induction methods in the agent learning mechanism. This approach presents a low computational cost, and it allows the agent to learn incrementally and find high-level regularities.

We are also inspired by (Rivest and Shapire 1994) and (Ron 1995), who had suggested the notion of ***state signature*** as an historical identifier to the DFA states, strongly reinforced recently by (Holmes and Isbell 2006), who have developed the idea of learning anticipations trough the analysis of relevant pieces of history.

9. Discussion and Next Steps

The CALM mechanism can provide autonomous adaptive capability to an agent, because it is able to incrementally construct knowledge to represent the deterministic regularities observed during its interaction with the environment, even in partially deterministic universes.

We have also presented an extension to the basic CALM mechanism in a way which enables it to deal with partially observable environments, detecting high-level regularities. The strategy is the induction and prediction of unobservable properties, represented by synthetic elements.

Synthetic elements enable the agent to overpass the limit of instantaneous and sensorimotor regularities. In the agent mind, synthetic elements can represent 3 kinds of "unobservable things". (a) Hidden properties in partially observed worlds, or sub-environment identifiers in discrete non-stationary worlds. (b) Markers to necessary steps in a sequence of actions, or to different possible agent points of view. And (c), abstract properties, which do not exist properly, but which are powerful and useful tools to the agent, enabling it to organize the universe in higher levels.

With these new capabilities, CALM becomes able to overpass the sensorial perception, constructing more abstract terms to represent the universe, and to understand its own reality in more complex levels.

CALM can be very efficient to construct models in partially but highly deterministic ($1 > \partial >> 0$), partially but highly observable ($1 > \omega >> 0$), and its properties are partially but highly informative ($1 > \varphi >> 0$). Several problems found in the real world present these characteristics.

Currently, we are improving CALM to enable it to form action sequences by chaining schemas. It will allow the creation of composed actions and plans. We are also including methods to search good policies of actions using the world model constructed by the learning functions.

The next research steps comprehends: to formally demonstrate the mechanism efficiency and correctness; to make comparisons between CALM and related solutions proposed by other researchers; and to analyze the mechanism performance in more complex problems.

Future works can include the extension of CALM to deal with non-deterministic regularities, noisy environments, and continuous domains.

References

Birk, A. and Paul, W. (2000). *Schemas and Genetic Programming*. In: Ritter, Cruse, Dean (Eds.), Prerational Intelligence: Adaptive Behavior and Intelligent Systems without Symbols and Logic, Volume II, Studies in Cognitive Systems 36, Kluwer.

Blum, A.L. and Langley, P. (1997). *Selection of relevant features and examples in machine learning*. Artificial Intelligence. Essex: Elsevier, v.97, p.245-271.

Chaput, H. (2004). *The Constructivist Learning Architecture*. PhD Thesis. University of Texas.

Crook P. and Hayes G. (2003). *Learning in a State of Confusion: Perceptual Aliasing in Grid World Navigation*. In: Proc. Towards Intelligent Mobile Robots, Bristol: UWE.

Drescher, G. (1991). *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press.

Goldstein, J. (1999), *Emergence as a Construct: History and Issues*. In: Emergence: Complexity and Organization 1: 49-72.

Holmes M. and Isbell C. (2005). *Schema Learning: Experience-based Construction of Predictive Action Models*. Advances in Neural Information Processing Systems, volume 17.

Holmes, M. and Isbell, C. (2006). *Looping Suffix Tree-Based Inference of Partially Observable Hidden State*. Proc. 23th ICML.

Morisson, C., Oates, T., and King, G. (2001). *Grounding the Unobservable in the Observable: The Role and Representation of Hidden State in Concept Formation and Refinement*. In: Working Notes of AAAI Spring Symposium.

Perotto, F.S., Alvares, L.O. (2006). *Incremental Inductive Learning in a Constructivist Agent*. In: Research and Development in Intelligent Systems XXIII, Proc. SGAI-2006. London: Springer-Verlag, p.129 – 144.

Piaget, J. (1957). *Construction of Reality in the Child*. London: Routledge & Kegan Paul.

Singh S., Littman M., Jong N., Pardoe D. e Stone P. (2003). *Learning Predictive State Representations*. Proc. 20th ICML.

Suchman, L.A. (1987). *Plans and Situated Actions*. Cambridge: Cambridge University Press.

Thornton, C. (2003). *Indirect sensing through abstractive learning*. Intelligent Data Analysis, v.7, n.3, p.1-16.

Trivino-Rodriguez J.L. and Morales-Bueno R. (2000). *Multiattribute Prediction Suffix Graphs*. Malaga.

Rivest R. and Shapire R. (1994). *Diversity-based inference of finite automata*. ACM Journal, v.43, n.3, p.555-589.

Yavuz, A. and Davenport D. (1997). *PAL: A Constructivist Model of Cognitive Activity*, Proc. Int. Conf. on New Trends in Cognitive Science: Does Representation Need Reality? Vienna, Austria.