Technical Report

IRI-TR-08-04

Learning Rules from Cause-Effects Explanations

Alejandro Agostini
Enric Celaya
Carme Torras
Florentin Wörgötter

May 2008

Institut de Robòtica i Informàtica Industrial

Learning Rules from Cause-Effects Explanations

A. Agostini, E. Celaya, C. Torras, F. Wörgötter

Abstract. In this work we propose a learning system to learn on-line an action policy coded in rules using natural human instructions about cause-effect relations in currently observed situations. The instructions only on currently observed situations avoid complicated descriptions of long-run action sequences and complete world dynamics. Human interaction is only required if the system fails to obtain the expected results when applying a rule, or fails to resolve the task with the knowledge acquired so far.

Index

Introduction 2
Policy and Sequences 2
Aliasing and Completeness 3
Motivations 3
Learning Action and Task-Preconditions 4
Advantages of using cec 4
Problem Formulation 4
Example Application. Pushing Boxes. 5
Perception Function 5
Cause-effect Description 6
Expected Outcome 7
Agent-Teacher Interaction 8
Instructing cec's 9
Policy Representation 9
Rule Generation 11
Learning Method Delineation 14
Conclusions 17
Complete vs Incomplete Instruction 17
Cognition and Learning: The OAC Concept 18
References 18

Introduction

In this work we are facing the problem of making a robot learns to perform tasks in the real world, a problem that have been pursued for many years with different degrees of success. A robot capable of acting in the world should perform a good action decision in accordance to its goals. This decision could be performed using planning techniques [1] that find optimal sequence to the goal using the model of the world, or by learning a function that directly maps situation to actions, usually known as a *policy* function [2], avoiding the necessity of a permanent sequence finding. Our work proposes a hybrid approach that learns an action policy through instructed sequences to the goal using a special coding of the model of the world based on cause-effect relations.

There are two learning paradigms involved in learning a policy: a *behavioural learning* paradigm used by the agent to learn which action to perform in each experienced situation, and a *representational learning* paradigm that generates the structure where the policy is coded. A couple of well known behavioural learning paradigms are the *supervised learning* paradigm, where a teacher explicitly instructs the robot about the action to take, and the *Reinforcement Learning* paradigm (RL) [2] where the robots is implicitly guided by a reward function associated to the actions preformed. On the other hand, there are many representational learning paradigm applied in policy representation like induction trees [3], Feature Based [4], Rule Based [3],[5],[6], Neural Networks [3].

In this contribution we combine supervised learning with a policy representation based on rules. A rule based representation permits an explicit declaration of the abstract meaning of the perceptions of the agent making easier the interaction teacher-agent needed in supervised learning, and a better understanding of the results. In particular, from the possible rule based representations, the one used in this work is an ordered set of rules of the type of Decision Lists [7] that permits a more compact knowledge representation and a better generalization [8].

One of the motivations of this work was the development of a method for fast policy learning using supervised learning. We consider that there are many tasks where an explicit instruction leads to faster learning convergence and lesser memory requirements than in RL. One of the reasons is that the convergence of RL is only assured by repeatedly experiencing the same (representative) situations and by an exhaustive exploration of the state space. In the method develop in this work the policy is learned rapidly by guiding the agent through the space of the searched policy, avoiding the tedious exploration needed in RL. The policy is derived by merging situations that require the same sequences of cause-effects.

Policy and Sequences

In order to show how the policy is coded for supervised learning we revise briefly the background of an agent acting in the environment. The agent follows a sequence of situations guided by the coded policy so far. The transition from one situation to the other depends on the action taken and on the probability of transition from the current situations to all the other situations. From the transitions perspective, the policy could be considered as a function that permits to follow a sequence of transitions to the goal. In this sequence the transitions that take place could be summarized as a succession of changes that transforms the current situation into a goal situation. Therefore, from a representational point of view, a policy could be also considered as a mapping from situations to sequences of changes. It should permit to select the proper sequence among all the possible sequences, or equally, to select the proper change among all the feasible changes that could take place in a situation.

For each situation, the policy coding should involve the conditions that permit the action selection that leads to the needed change in the sequence. But the desired changes will only take place if the conditions that afford those changes are involved in the situation. Therefore, the policy should contemplate both conditions. The former preconditions will be denoted as *task-preconditions* and serve to select the change from all the possible changes, and the later ones as *action-preconditions*, that permit the selected change to occur. If one of those preconditions is wrongly considered then the coded policy leads to a bad sequence.

Aliasing and Completeness

If the conditions coded in the policy are not correct, some situations merged together in a rule, hence requiring the same sequence, would lead to different changes producing what is known as *aliasing*. In the case of aliasing we will denote that the agent has an *incomplete representation* with respect to the desired outcome. In order to detect the aliasing the agent needs to evaluate every obtained change and compares it with the desire change that should take place in the sequence. We denote the desire change as the expected outcome (*eo*) and will be estimated during the learning with the help of the instructor. The agent improves the *eo* estimation while learning and every observed outcome is evaluated by checking the consistency with respect to it.

In this work we assume that all the perceptions of the agent fulfil the Markov property [2] with respect to the changes. Therefore, there will be no aliasing if all the perceptions are considered for the task and action-preconditions. Nevertheless, we could extend our previous statements to non Markov perceptions by considering also past perceptions as part of the current situation (see [9] for an illustration).

Motivations

One of the arguments that sustain the method developed in this work relies on the fact that a particular change could take place in many different sequences. Usually, the learning approaches attach to the policy coding the expected outcome to evaluate the completeness of the representation with respect to the desired outcomes. This implies that the rate of convergence of the expected outcome estimators depends on the frequency experiencing of situations with the same task-preconditions. Nevertheless, many situations that require different task-preconditions may involve the same action-preconditions as the needed change is the same despite the sequence is different. Separating the coding of action-preconditions from the policy allows a rapidly convergence of the *eo* because the experiencing rate of action-preconditions is much higher than the experiencing rate of the task-preconditions.

Another motivation for the development of the presented method arises from the analysis of the learning behaviour of the children in their early stages. The children explore the world by experimenting with objects and their bodies performing different actions like introducing objects into containers or narrow places, throwing objects, touching, pushing, hiding, etc. Doing this they learn how to perform individual actions and how the world reacts to them, i.e. the cause-effect behaviour of the world. Once the cause-effects are learned it is much easier to teach them how to perform a task by indicating the corresponding sequence of cause-effects. This is because it is not necessary to teach them how to perform each individual action and produce each individual changes of the sequence from scratch. These observations are sustained by some enunciations of the theory of Piaget's cognitive development [10] who claims that the children gradually acquire knowledge of the cause-effects relations trough *circular reactions* where actions and processes are repeatedly executed and sequenced to reach goals.

Learning Action and Task-Preconditions

As mentioned before, in order to code the policy, both action and task-preconditions should be correctly considered to produce the desired changes in the sequence. There are different ways of finding the proper action and task-preconditions. The method proposed in this work lets the agent learns action-preconditions by experiencing the world using the help of a teacher and the task preconditions are derived from sequences of action-preconditions in accordance to a task. The agent codes the action-preconditions in a structure called cause-effect couples (*cec*) that involves the action-preconditions and the expected outcome needed to evaluate completeness. The sequence of *cec*'s is used to produce the coding of the policy that consists in a set of rules that merge situations that require the same sequence. The sequence is uniquely coded in rules by *linking* the *cec*'s using the action-preconditions to generate the task-preconditions. Then, the task-preconditions "keep track" of the detectors that will be needed for future changes affordance.

To assure continuity in the agent behaviour the experiencing and learning of *cec* and rules are spread along a task dependent performance. The selection of *cec* is dictated by the coded rules so far and supervised by the teacher. The teacher instructions serve to update the existing *cec*'s and to generate those *cec*'s needed in the sequence and not known by the agent.

Advantages of using cec

Coding and evaluating the policy using the *cec*'s has many advantages. As mentioned, the speed of convergence of the *eo* is high and the amount of information needed for the policy learning is reduced compared to the conventional policy learning methods. On the other hand, as we are using supervised learning, action-preconditions are much easier to instruct than task-preconditions that require a precise knowledge about which preconditions should change in the whole run of the task execution. Another important advantage is that the *cec*'s are independent of the task performed depending only on individual actions. Therefore, we can use them to build different set of rules for different tasks by simply specifying different sequences. Therefore, the experience acquired during the learning of a task could be used to rapidly learn a policy for another task making this approach very suitable for multitask learning.

Problem Formulation

We now introduce the theoretical formulations of the concepts described before and how they are implemented in the learned method.

We assume that the agent has a set of N sensors that measure some characteristics of the environment. The value of the senor i is called an observation o_i. Each of these sensors is internally represented by the agent as a detector d_i that could take different values d_{ij} called *conditions* depending on the observation o_i. The function that maps observations to conditions is called the *perception function*.

A *world state so* is formed by the set of observations o_i , $i = 1..N$. We call a *prior world state* so^{prior} to the *so* composed by sensor observations before the action selection and a *posterior world state* so^{post} to the *so* composed by sensor observations after the action execution.

An internal agent state *s*, from now on a *state*, is constituted by a set of conditions d_{ij}, i=1..*N*, s={d_{1j}, d_{2k},…,d_{Nl}}.

Finally, we assume that at each moment the agent is capable of performing a well defined action a_i corresponding to the set of all possible actions *A*.

Example Application. Pushing Boxes.

In order to illustrate the method developed we select a simple application where a wheeled robot is embedded in a simple world with white background that contains black squared boxes, all of the same size. The goal of the agent is to clear the path in front of its trajectory. It is important to note that this application was proposed only with illustration purposes but is not complex enough to elucidate all of the difficulties that the learning method could face in a real world application.

The application involves six visual sensors configured as a grid taken form an image of the immediate front of the robot as shown in Figure 1 *so*={o_1,o_2,o_3,o_4,o_5,o_6}. The observation of each sensor consists in the mean grey scale of the pixels inside the corresponding cell in the grid. The boxes lie on the path of the robot in different positions.

The perception function transforms each world state *so* into a state *s*={d_1,d_2,d_3,d_4,d_5}. Each detector d_i could take value 0 when the observation is considered to be a white (or an empty space) or 1 when the observation is a black (or an occupied space). Figure 1 also shows an example of these states for a particular situation.

The goal states are represented as S_g={?, 0, ?, ?, ?}, where the sign "?" specifies that the detector could take any of its possible values.

The actions available for the robot are enumerated.

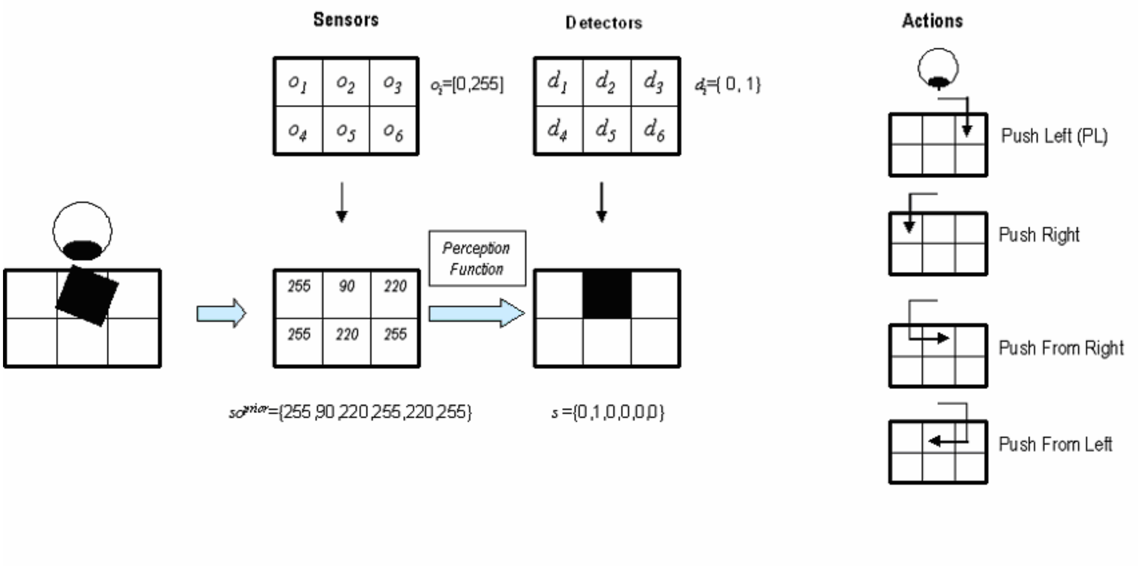


Figure 1. Example Application

Perception Function

The robot should be able to map a given value sensed from the environment to a predefined condition. This mapping from sensed values o_i^{prior} to conditions d_{ij} could be hand coded as previous knowledge, or, as it is done in this work, could be learned by the agent while interacting with the environment.

The idea is that the teacher is more capable of giving a meaning for a sensed value (the "name" of the condition) than the specific range of the sensed values where this meaning should take place. For instance, suppose we have a cup with certain amount of sugar inside. The precise range of fullness and emptiness is difficult to be specified by the teacher but the meanings of fullness and emptiness are easier to provide.

Besides, usually there could be different interpretations for one sensed value as considered in fuzzy logic. In our case we consider probabilities for all the conditions given an observed value and the one with highest probability is considered for the meaning of the observation (see figure 2).

It is assumed that each condition has a normal probability distribution over the observed values where the mean and variance are estimated using the sampled mean and variance fed with the interpretations given by the instructor.

Giving a formal background, for every *condition* d_{ij} of detector d_i we associate three statistics values: An estimation m_{ij} of the mean value of observations $o_i^{\,prior}$ when condition d_{ij} is interpreted; an estimation var_{ij} of the variance of observations $o_i^{\,prior}$ when d_{ij} is interpreted; and n_{ij} the number of times condition d_{ij} is the meaning of the observations $o_i^{\,prior}$. The statistic values related to detector d_{ij} are updated using a weighted average method in the following way,

$$n_{ij} = n_{ij} + 1$$

$$\beta = 1/n_{ij}$$

$$m_{ij} = (1-\beta)\, m_{ij} + \beta\, o_i^{prior}$$

$$var_{ij} = (1-\beta)\, var_{ij} + \beta\, (m_{ij} - o_i^{prior})^2$$

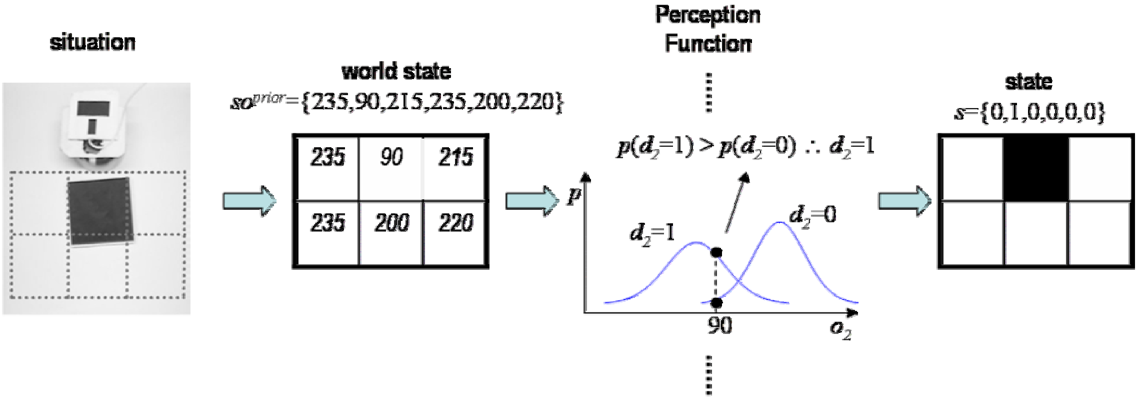


Figure 2. Example of a situation experienced by the robot and how the perception function maps the observation o_2 into a value for the detector d_2. In this case the probability for the condition d_2=1 is higher than the condition d_2=0 given the observed value o_2=90.

Cause-effect Description

We represent a *cec* using a tuple that consists in a subset P_a of state conditions that will describe the *action-preconditions*, an action a_i from the set of actions A, and an estimation of the expected outcome *eo* produced by action a_i in those states where P_a are present.

$$cec \rightarrow \langle P_a=\{d_{ij},\ldots,d_{ml}\},\, a_i\,,\, eo\rangle$$

Expected Outcome

The expected outcome *eo* is coded using a probabilistic approach commonly used in learning approaches for stochastic environments.

As mentioned above, the changes are sufficient to evaluate the followed sequence; hence, evaluating only the changed condition is enough for the outcome evaluation.

Nevertheless, there are some points that should be considered to code the *eo*. One of them is related to which of the changes are really causally produced and not due to other contingency factors. In the approach this is overcome by making use of the fact that the conditions that will (causally) change are included in the conditions that afford that change. As the conditions in P_a are completely instructed then those conditions are also instructed. In the approach develop so far we consider all the observed outcomes of the sensors related to the conditions in P_a, despite some of them may not change but afford other conditions changes.

Another point to tackle is to establish how large should be a change to be considered as a change. Usually the outcomes thresholds are predefined by the designer. But, as we are considering complete instructions to produce desired changes, the observed outcomes after the instructions are indeed consistent with the desired outcomes and the thresholds could be derived from the expected outcomes estimations. A problem could arise when the estimators are updated when no instruction is given because the observed outcome could be inconsistent with the desire one and the updating could distort the *eo*. To avoid this problem the updating of the *eo* is only produced after an instruction is given or after verifying that the observed outcome is indeed consistent with the *eo* before the updating.

Another important issue is how to code the *eo*. To perform a theoretically correct analysis of the outcomes we should consider a multivariate probability distribution that involves all the observed values for the different sensors analyzed. This multivariate probability distribution is very complicated to deal with. Instead we will consider a single variable probability distribution for each evaluated sensor. If at least one observation is not consistent with the expected outcome then the changes produced by the action are considered not to belong to the sequence. In this case we say that the agent get a *surprise*. Figure 3 illustrates a situation where the observed change is consistent with the expected one and a situation where a surprise takes place.

We now formally define the expected outcome *eo* as a set of confidence intervals I_i, one for each $o_i^{\ post}$ where i indicates the index of the condition that will change. The confidence intervals are built using a sampled mean m_i and variance var_i of the observed values $o_i^{\ post}$, the number of observations n_i that fed the statistics, and the sampled distribution t in the following way,

$$I_i = \left[m_i - t_{\alpha/2}^{n_i-1}\sqrt{\mathrm{var}_i}, m_i + t_{\alpha/2}^{n_i-1}\sqrt{\mathrm{var}_i} \right] = [I_i^{min}, I_i^{max}]$$

where $t_{\alpha/2}^{n_i-1}$ is the value of the t distribution with $p=\alpha/2$ and n-1 degrees of freedom.

The statistics for the confidence intervals are fed by the observations obtained after the *cec* occurrence (instructed or dictated by a rule),

$$n_i = n_i + 1$$

$$\beta = 1/n_i$$

$$m_i = (1-\beta)\, m_i + \beta\, o_i^{post}$$

$$var_i = (1-\beta)\ var_i + \beta\ (m_i - o_i^{post})^2$$

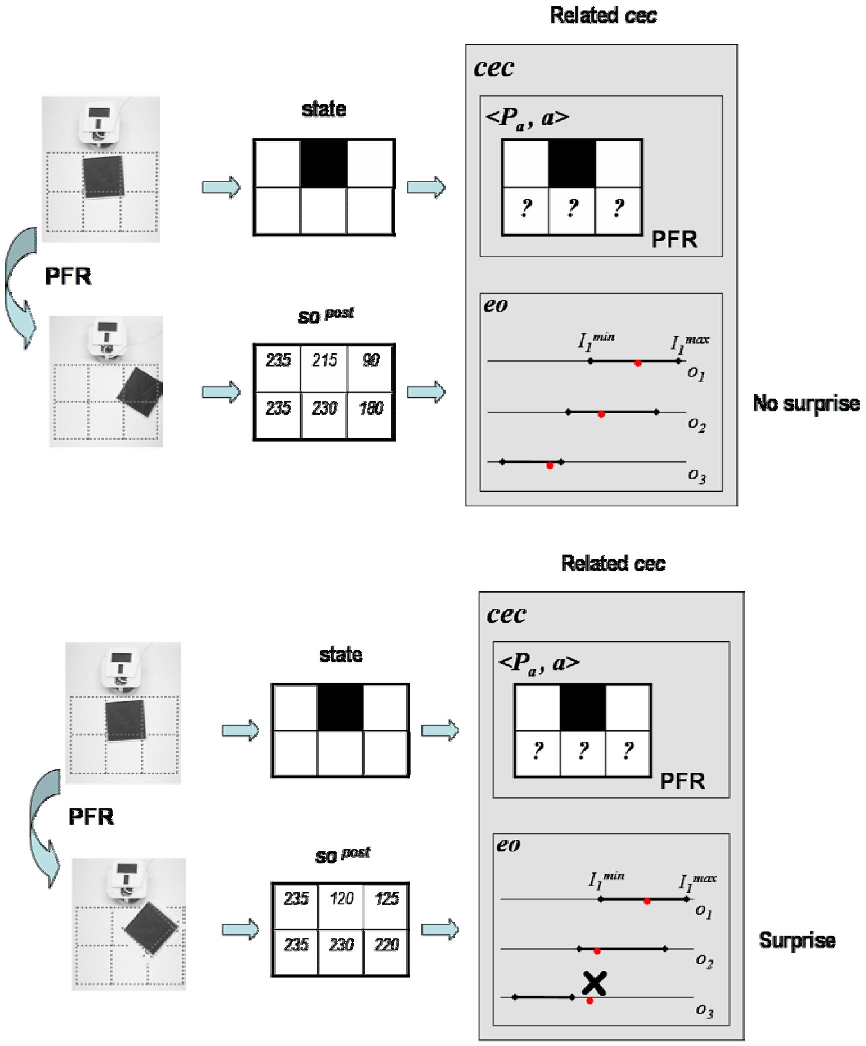


Figure 3. Surprise examples.

Agent-Teacher Interaction

The agent interacts with the environment guided by the rules coded so far and supervised by the teacher. When the agent fails to obtain the expected outcome or has little knowledge about the changes that should take place in the experienced situation, it asks for instruction. We assume that the teacher has complete knowledge of the policy to perform and the conditions that afford each desired change in the sequence.

We now analyze briefly the cases where the agent receives instruction. For some situations experienced by the agent the *cec's* that should be used to evaluate the desire change could have been experienced only a few times, so it is the followed sequence. This low confidence leads the agent to ask for instruction about which *cec* should take place to make sure that the proper sequence is followed. If the instructed *cec* corresponds to a different sequence then the agent generate the *cec* and update the policy generating new rules. Conversely, is the instructed *cec* belongs to the already coded sequence then the agent only updates the *cec* statistics.

Another circumstance that needs instruction is when the agent has enough confidence in the estimators of the *eo* but fails to obtain the expected changes. This permits a robust management of the false surprises because the teacher will instruct the same inferred *cec* and no rules will be generated due to the sequence checking (see flow diagram below).

Instructing cec's

After analyzing when an instruction of a *cec* is needed we will explain how this instruction could be done. One of the possibilities, not used in this work, is that the teacher instructs the agent only about the action to perform without specifying the action-preconditions P_a that avoids the aliasing. The agent should find those conditions that afford the changes using the conditions of the experienced states. Letting the agent to find the proper conditions that solve the aliasing is a very complicated issue and many constructivist techniques could be tried to tackle this problem [3]. Another option is that the teacher instructs the agent about the conditions that the agent should "pay attention to" in order to obtain the desired changes. If this instruction is incomplete then we fall in the same case as before, the agent should complete the action-preconditions using a constructivist technique.

In the approach developed so far we let the agent receives instruction from a teacher that indicates which action to perform and the complete conditions the agent should pay attention to afford the observed changes.

Future works will consider how to find the action-preconditions with less instruction while coding the policy.

Policy Representation

Before explaining how the rules are generated we will explain the rule representation used in this work. There are many different rule based representations, in this work we will use the one described in [8], a rule based policy representation that belongs to the type of Decision Lists [7]. Here we make a brief explanation of the representation. For a more detailed description see [8].

As in the case of many rule representations, each rule is a function that maps all the states in a region X_r of the state space to a real value r: $X_r \rightarrow \Re$. The description of the regions X_r is done using subsets of state conditions $X=\{d_{ij},...,d_{kl}\}$, $k \leq N$, where N is the number of conditions in s. Each X describes a subspace of the state space. Figure 4 shows three subspaces for a simple discrete function of two variables and three regions description marked in grey.

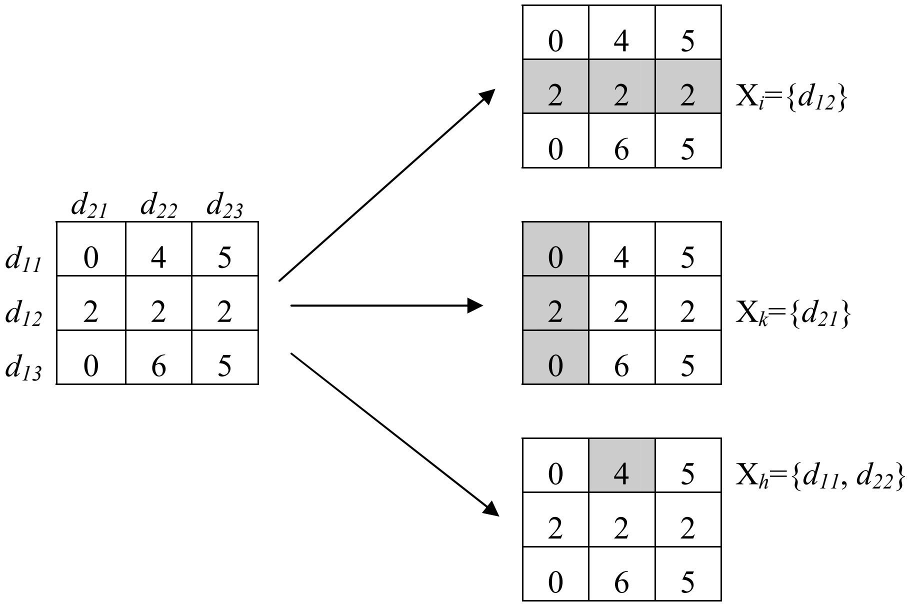


Figure 4. Subspaces description.

One of the most distinguished features of the used representation is the way the regions X_r are described allowing a more compact representation of the knowledge. Each X_r is described using an ordered list of subspaces X_i where the index i denotes the position in the list. Each X_i in the list has a rule associated r_i. For a rule r_i associated to subspace X_i, the region X_{ri} represented by the rule is given by:

$$X_{ri} = X_i - (X_{i-1} \cup \ldots \cup X_1)$$

being X_1 the first subspace in the list. For instance, figure 5 illustrates one possible X_r for the exampled discrete function in figure 1. In this case the rule maps the region X_r to the inferred value 0, $r(X_r)$=0.

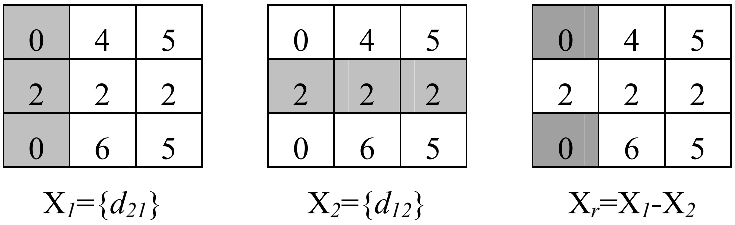
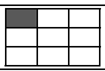
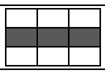
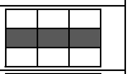
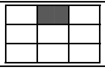
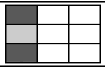
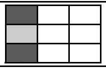
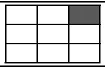
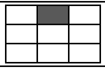
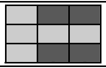
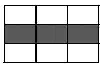
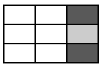
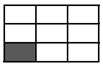
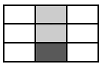
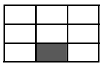
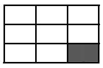


Figure 5. Rule region example.

Finally, table 1 shows three possible rule representations of the exampled function to elucidate how a decision list could lead to a more compact description. Table 1A shows a conventional rule representation where the subspace X associated to a rule is indeed the rule space X_r. Table 1B shows one possible rule representations using decision lists. Note how the amount of information needed for the representation decrease with respect to 1A. Table 1C reveals how this representation could lead to an even smaller set of ordered subspaces by decreasing the precision requirements for the performed function approximation. In this case an approximation error up to 1 is allowed.

Table 1. Examples of a rule representation for the discrete function in Figure XX.

A- Not ordered description $\delta=0$			**B- Ordered with** $\delta=0$			**C- Ordered with** $\delta=1$		
r	X_r	Points	*r*	X	Points	*r*	X	Points
$r_1(x)=0$	$\{d_{11}, d_{21}\}$		$r_1(x)=2$	$\{d_{12}\}$		$r_1(x)=2$	$\{d_{12}\}$	
$r_2(x)=4$	$\{d_{11}, d_{22}\}$		$r_2(x)=0$	$\{d_{21}\}$		$r_2(x)=0$	$\{d_{21}\}$	
$r_3(x)=5$	$\{d_{11}, d_{23}\}$		$r_3(x)=4$	$\{d_{11}, d_{21}\}$		$r_3(x)=5$	{Ø}	
$r_4(x)=2$	$\{d_{12}\}$		$r_4(x)=5$	$\{d_{23}\}$				
$r_5(x)=0$	$\{d_{13}, d_{21}\}$		$r_5(x)=6$	$\{d_{22}\}$				
$r_6(x)=6$	$\{d_{13}, d_{22}\}$							
$r_7(x)=5$	$\{d_{13}, d_{23}\}$							

The main concern of the representation learning methods is to find the least information needed to have an accurate enough approximation of the represented function. In the case of the methods that use decision lists this concern is focused in the way these rules are generated and ordered. It is clear that for applications where the number of possible conditions is large, the number of possible rules and ordering increase exponentially and finding the optimal set of rules is a very tough task. The method propose in this work generates and arrange the rules efficiently using the *cec*'s.

Having introduced the rule representation used we now define a rule *r* in the context of this work as a set of conditions P_T called *task-preconditions* that describes the rule domain X_r and an action a_i. Each rule is labelled with the index of the associated *cec*,

$$r_{icec} \rightarrow < P_T=\{d_{mj}, d_{kl},\ldots,d_{ml}\}, a_i>$$

Rule Generation

To illustrate how the rule generation takes place we split the explanation into two different processes. The first one consists in how a conventional rule representation of the policy (not a DL) could be generated from a sequence of *cec*'s, and the second one is about how the rules already generated could be arranged in a DL leading to a more compact and intuitive representation. Finally, we show how both methods are combined in one procedure.

Firstly, we assume that a complete sequence of *cec's* to the goal from a given situation is already instructed and learned. With this sequence of *cec's* the rules are generated using the P_a of the involved *cec's*. In order to show how the rules are generated first note that each rule should be applied in those situations where the whole sequence should follow. With this in mind note that the occurrence of the immediate *cec* in the sequence is assured after checking the existence of the P_a of the *cec* in the state, and the feasibility of the sequence is guaranteed by checking the existence of all the conditions P_a needed for future changes not contemplated in the P_a of the immediate *cec*. In order to generate the rules that permit the checking of the previous conditions (that in conjunction constitutes the P_T's coded in the rules) they are generated by back-propagating all the conditions departing from the last instructed *cec* and generating a rule for each visited *cec* by accumulating the P_a's. Figure 6 shows an example of how this is done for a particular sequence in the Pushing Boxes (PB) application.

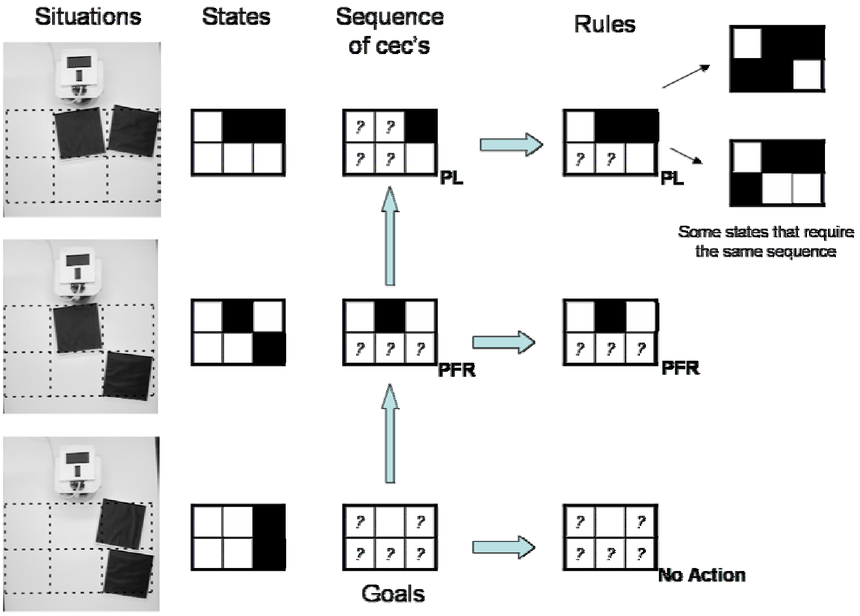


Figure 6. Rule generation of a not ordered set of rules.

The rules obtained in figure 6 could be represented in a DL in such a way that the amount of information needed is decreased while producing a more intuitive representation of the sequence since the order permits following the sequence of *cec's* as a cascade execution of rules, eliminating "restrictions" that prevent former changes in the sequence to occur. Figure 7 illustrates how this is performed in the previous example.

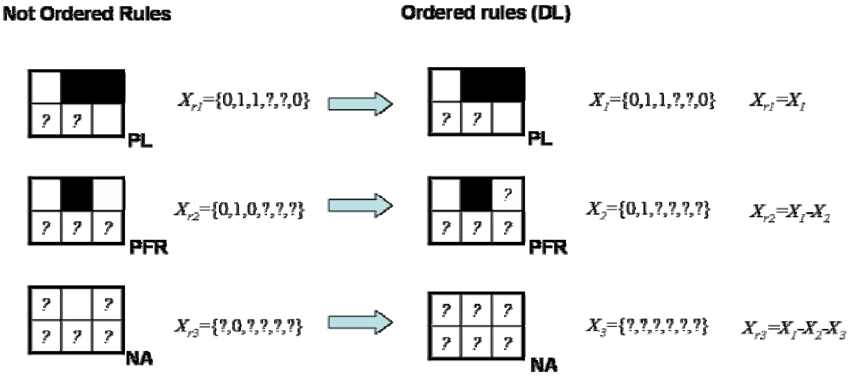


Figure 7. Conversion from not ordered to ordered set of rules.

As it is observed in figure 7 the last rule of the DL covers the entire state space. In general, if we code the policy using a DL the inference not only is done in the regions where the instructed sequences take place but also in the rest of the state space where the DL permits to generate hypothesis about the value to infer depending on the subspaces X_i considered for each rule. Nevertheless, this does not imply an inference distortion in the X_r, the inference properties in those regions remains the same.

Finally, we present a method that combines the previously explained procedures generating rules from *cec's* and representing them in a DL at the same time.

Now we will generate the subspaces X_i instead of X_r. The subspaces X_i are generated from bottom to top departing from the goal state. A subspace related to a *cec* is generated considering all the conditions needed to afford the immediate and future changes and avoiding all the conditions in the sequence that have already changed (from the current *cec* to the first in the sequence). Note that the ones that have changed before the current *cec* are the eliminated restrictions that permit the sequence to take place. A cascade of restriction elimination is produced with the DL while decreasing the amount of info needed. Figure 8 illustrates how this is performed in the example considered before.

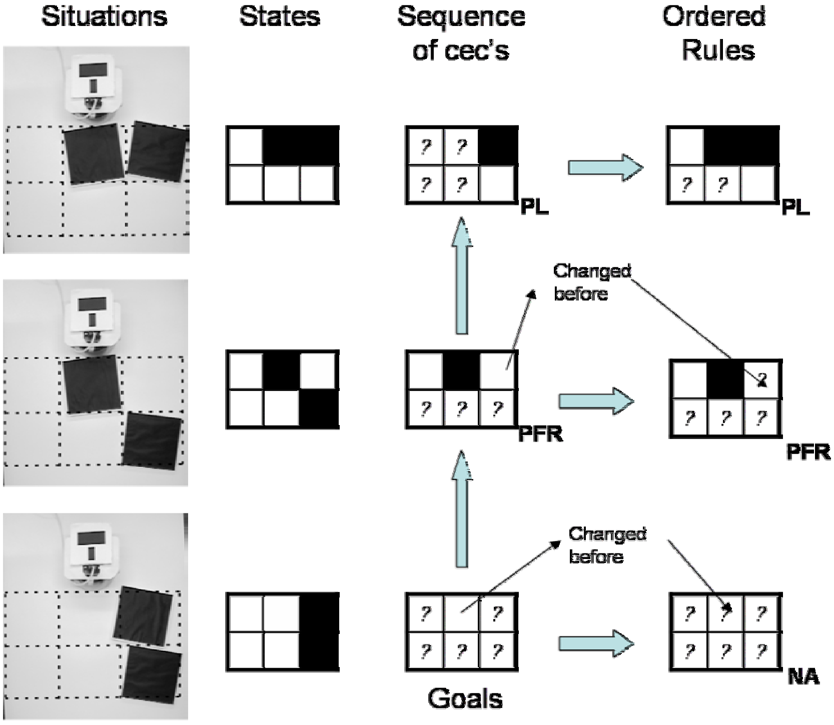


Figure 8. Rule generation ordered set of rules.

We have explained how to generate rules given the whole sequence of *cec's*. In general it is not necessary to have explicitly specified the whole sequence of *cec's* to generate rules during the policy learning. This is because many sequences have steps in common when approaching to the goal. Thus, it is sufficient to produce rules until a good sequence is reached and then continue with the already coded rules to the goal. The *cec's* are then instructed and placed in order but only when there is a "gap" in the sequence, until a good rule is applied. Then, the new rules are generated using the instructed *cec's* and linked to sequence by back-propagating the conditions of the first good rule in the following sequence and forward-propagating the conditions that have changed in the new instructed *cec's* decreasing even more the amount of information needed for the policy description. This process is illustrated in figure 9 using the same sequence presented before making the agent experiences a situation with no good sequence coding.

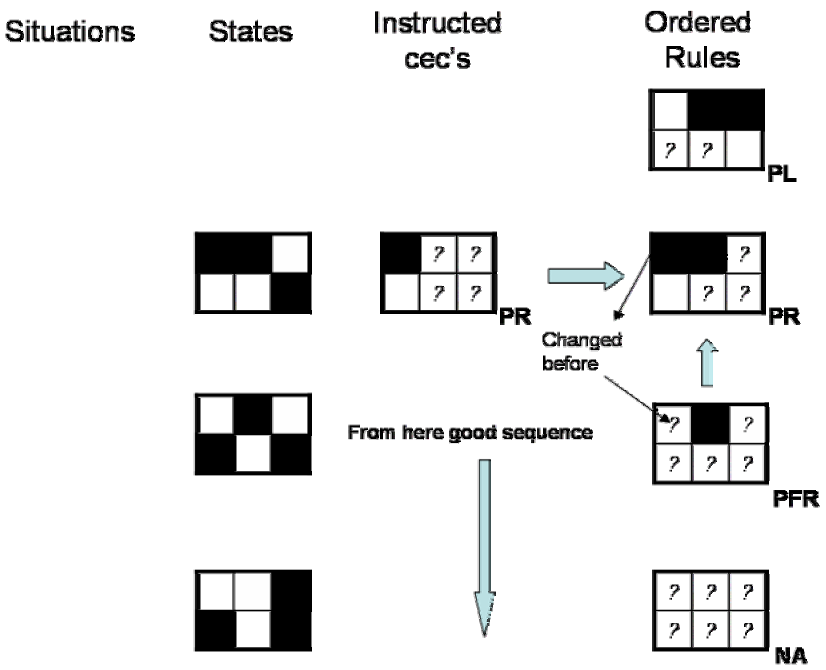


Figure 9. Rule generation using one instructed *cec* and already existing rules.

Learning Method Delineation

In this section we compile all the procedures explained before in a general learning method, which flow diagram is presented in figure 10.

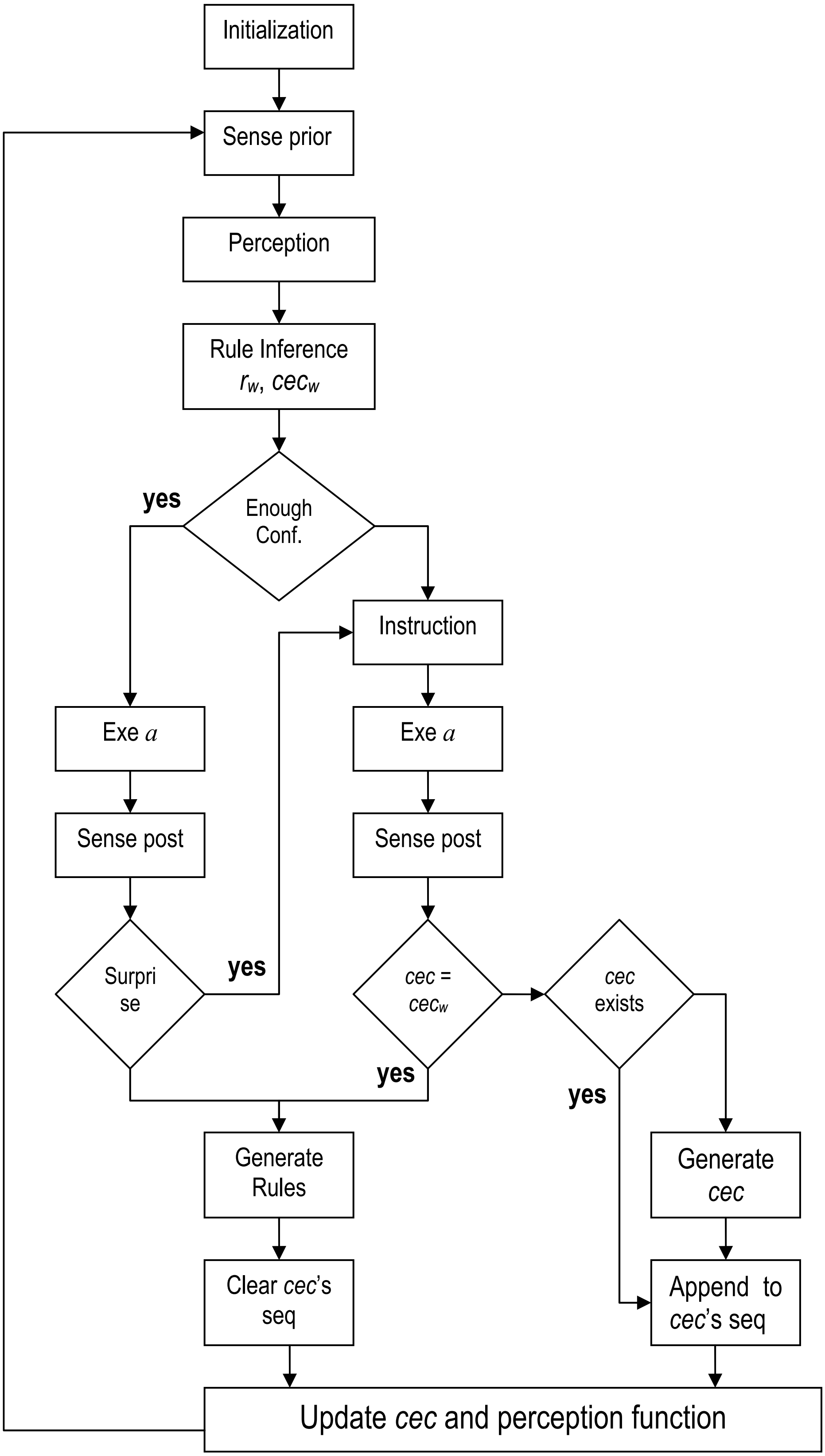


Figure 10. General flow diagram of the method.

Initialization

As the policy should indicate which action to perform in any state of the state space, the agent begins the learning process with a rule that assumes all the states as a goal state and indicating the agent to perform no action (NA). The rule is evaluated by a *cec* that codes indeed the goal states, leading to surprises in all the states that require some actions to reach the goal. Therefore, the set of rules will initially contain one rule and so it is the set of *cec's*,

$$R = \{\ r_0 \rightarrow <\{?,?,?,?,?,?\}, \mathrm{NA}, i_{c0}>\ \}$$

$$C = \{c_0 \rightarrow <\ \{?,0,?,?,?,?\}, \mathrm{NA}, eo = \{I_2\}>\}$$

For the perception function, the mean values m_{ij} associated to each condition d_{ij} are set equally spaced in the entire range of the sensor o_i. The variances var_{ij} are initialized to a high value and the n_{ij} to 1.

Sense Prior

Get the sensed value o_i for each detector d_i and form the so^{prior} .

Perception

Map each sensed value $o_i^{\ prior}$ into a condition d_{ij} using the *perception function* and form the state s=$\{d_{1j}, d_{2k},\ldots,d_{Nl}\}$, where N is the number of detectors and sensors.

Rule Inference

Select the upper subspace X_i that involves the state s and extract the rule r_w associated to this subspace, in conjunction with its cec_w.

Confidence Evaluation

IF the number of samples n_i in the statistics stored in the *eo* of the cec_w is under a critic number n_c then it is considered that there in no enough confidence in the followed sequence and in the estimators to evaluate the observed outcome and the agent needs instruction.

Instruction

The agent receives instruction about the action to perform a_i and the conditions d_{ij} that afford the desired changes.

Sense Posterior

Get the sensed value o_i and form the so^{post}

cec=cec$_w$

In order to check if the proper sequence is taking place when there is little confidence about the inferred cec_w and its estimations of the *eo*, the agent checks the instructed *cec* that indeed belong to the desire sequence, against the inferred cec_w.

cec exists

The existence of the instructed *cec* in the data base of *cec's* (the permanent one, not the temporal ordered list which is different) is verified. If it does not exist then it is generated and appended to the temporal ordered list of instructed *cec's*.

Surprise Evaluation

Evaluate the consistency between the observed outcome *so*^{post} and the *eo* of *cec*$_W$. If there is an inconsistency then the agent has a surprise and asks for instruction.

Generate cec

Using the instructions given by the teacher about P_a and a form a new *cec,*

$$cec \rightarrow < P_a, a, eo>$$

where the *eo* contains the statistics (mean, variance and number of samples) to form the confidence interval I_i for each of the sensor values o_i after performing the action *a*. This statistics are initialized in the same way as in the Initialization step (see above).

Append Generated cec

Append the new generated *cec* to the temporal ordered list of instructed *cec*'s that will be used to generate the rules (see section Rule Generation above). If there is no list, then it is generated using the instructed *cec* as the only element.

Generate Rules

The rule generation take place when a proper sequence is find, i.e., when there is no surprise after performing an action or when the instructed *cec* coincides with the inferred *cec*.

If there is no *cec's* in the temporal ordered list of instructed *cec's* then no rule is generated, otherwise the rule generation take place as explained in the section Rule Generation and then the ordered list of *cec's* is cleared up.

Update

The update process takes place in two structures:

a) Update Perception Function. For every condition d_{ij} of the *cec* update the statistics of the perception function using *so*^{prior}.
b) Update *cec eo*. Using *so*^{post} update the statistics of the *eo* estimators.

Conclusions

Complete vs Incomplete Instruction

With the assumed complete instruction of the P_a's, if we opt to generate a conventional rule representation instead of an ordered list, the surprise could only take place when there is no rule applicable, otherwise the desired changes will always occur because all the conditions to afford it are instructed (in some cases there could be a surprise indeed despite the completeness of the instruction when there is a contingency that prevent the desired changes to occur. However these cases are solved in the same way as incomplete instruction explained below). Nevertheless, if we code the policy using a decision list the inference not only is done in the regions where the instructed sequences take place but also in the rest of the state space where the DL permits to generate hypothesis about the value to infer depending on the subspaces X_i considered for each rule (see figure 7 as an example). These hypotheses could be wrong and then a surprise is possible despite of the complete instruction, actually the case considered in the presented method.

On the other hand, the case of a teacher only capable of giving incomplete instructions is more probable in complex applications where the conditions responsible for the changes are not well known by the teacher and the causal changes are more difficult to identify. In this case the *cec's* should be generated and restructured constantly using some constructive learning approach until the conditions that afford the causal changes are correctly recognized. Thus, when incomplete instruction with a DL is used, the surprise could be originated by even a bad hypothesis or a bad sequence resulted from the incomplete instruction.

Another point to analyze is that the rules generated with incomplete *cec's* are also incomplete and every *cec* updating should involve the updating of the rules related to it.

Finally, another case to take into account is when despite the hypothesis produced by a decision list is wrong the observed changes is consistent with the *eo*. This happens when the sequence is incorrect but the *cec* inferred could indeed occur. Despite that this is possible it is not likely to occur because of the sequence checking when the confidence in the estimations is low. This permits a sequence correction in almost all the rules in the list. Nevertheless, to assure a complete convergence a sequence checking should be performed after a no surprise is obtained. This could be done explicitly by the teacher or implicitly by, for instance, attaching to each rule an estimation of the steps to the goal from that point in the sequence and performing an analysis similar to the cumulative reward of reinforcement learning; if the observed steps to the sequence are not consistent with the estimated step, then the sequence is wrong and the agent should be instructed.

Cognition and Learning: The OAC Concept

Most of the learning to act approaches are based in human learning and cognition capabilities. Despite these approaches present many differences among them, they all establish a direct relation between perceptions of the agent, coded mainly as states, and actions. In contrast to the amount of approaches developed, a few attempts were done to create a common framework that permits to consistently relate the learning to act approaches with the human cognition capabilities for learning and acting. One of these attempts is the concept of object-action complexes (OACs) [11] that has been evaluated and developed by the European PACO+ consortium [12]. Briefly, the OAC concept claims that the world contains undistinguished "things" meaningless for the agent that become meaningful "objects" through actions and tasks, where the objects are described by the properties relevant for the fulfilment of the final desired outcome through the action.

We believe that from all the possible approaches that perform a state-action linkage in learning to act, the explicit coding of the world conditions and actions through rules and cause-effects presented above is one of the most suitable for a first insight in the study and refinement of the OAC concept. One of the reason is that the elements of these structures could be directly associated with the main elements of the OACs concept formulated so far. Another reason is that they permit a direct association with the human cognition capabilities through the explicit declaration of the abstract meaning of the conditions of the world, and hence a better understanding and a faster evaluation of the results.

References

[1] La Valle, S. Planning Algorithms. Cambridge University Press. 2006.

[2] Sutton, R. and Barto, A. Reinforcement Learning. An Introduction. MIT Press, 1998.

[3] Mitchell, T. Machine Learning. McGraw Hill. 1997.

[4] Liu, H. and Motoda, H. Feature Selection for Knowledge Discovery and Data Mining. Boston: Kluwer Academic, 1998.

[5] Wojtusiak, J., Michalski, R. S., Kaufman, K. and Pietrzykowski, J., The AQ21 Natural Induction Program for Pattern Discovery: Initial Version and its Novel Features. In Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, Washington D.C., November 13-15, 2006.

[6] Clark, P. and Niblett, T. The CN2 Induction Algorithm. Machine Learning Journal. 3(4):261-283. 1989.

[7] Rivest, R. Learning Decision Lists. Machine Learning, 2(3):229-246. 1987.

[8] Agostini A., Celaya E. Generalization in Reinforcement Learning with a Task-Related World Description using Rules. Technical Report IRI-DT 2006/01. Institut de Robòtica i Informàtica Industrial. UPC-CSIC. (Barcelona, Spain), June 2006.

[9] McCallum A. Reinforcement Learning with Selective Perception and Hidden State. Phd. thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1995.

[10] Piaget, J. The origins of intelligence in children. New York: International Universities Press. 1952.

[11] C. Geib, K. Mourao, R. Petrick, N. Pugeault, M. Steedman, N. Krüger and F. Wörgötter. "Object Action Complexes as an Interface for Planning and Robot Control", presented at IEEE RAS Int Conf. Humanoid Robot, Genova, Italy, 2006.

[12] http://www.paco-plus.org