Towards BDI sapient agents: learning intentionally

Alejandro Guerra-Hernández and Gustavo Ortiz-Hernández

Departamento de Inteligencia Artificial
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México, 91000
aguerra@uv.mx

1 Introduction

Sapient agents have been characterized as systems that learns their cognitive state and capabilities through experience (Otterlo, Wiering, Dastani and Meyer 2003). Other features of such agents include considering social environments, interaction with other agents or humans, and the ability to deal with emotions. One of the best known models of cognitive agency is the belief-desire-intention (BDI) approach. The relevance of this model can be explained in terms of its philosophical grounds on intentionality (Dennett 1987) and practical reasoning (Bratman 1987), as well as its elegant abstract logical semantics (Rao 1996; Rao and Georgeff 1998; Singh, Rao and Georgeff 1999; Wooldridge 2000). However, if sapience is the issue , two well known limitations of this model must be considered (Georgeff, Pell, Pollak, Tambe and Wooldridge 1999): Its lack of learning and social competences. This paper discusses some extensions to the BDI model (Guerra-Hernández, El-Fallah-Seghrouchni and Soldano 2001, 2004a,b, 2005), enabling intentional and social learning. By intentional learning, we mean that these agents can learn, and then update, their practical reasons (plan's contexts) to adopt an intention, particularly to correct experienced failures. By social learning, we mean that these agents can interact through collaborative goal adoption (Castelfranchi 1998) to learn intentionally and keep consistency in their Multi-Agent System (MAS). The resulting BDI learning agents seem closer to the intended characterization of sapient agents.

The organization of the this contribution is as follows: Since the BDI model is well known, section 2 offers a very brief introduction to BDI agency, pointing out some relevant concepts and references. Section 3 discusses BDI learning agents design issues. Section 4 focuses on a scale of learning agents, induced by their degree of awareness. Section 5 present some examples of BDI learning agents at the first two levels of the proposed scale: individual and collaborative learning. Finally section 6 concludes and presents future work, including a new protocol based on incremental learning.

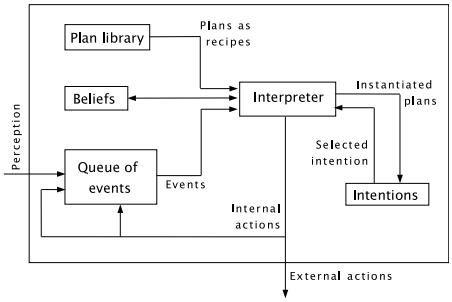

Fig. 1. A BDI architecture following dMARS specification.

2 BDI agency and learning

Agents are usually characterized as systems that exhibit flexible autonomous behavior (Wooldridge 2000). BDI models of agency approximate this kind of behavior through two related theories about the philosophical concept of intentionality: Intentional Systems (Dennett 1987) defined as entities which appear to be subject of beliefs, desires and other propositional attitudes; and the Practical Reasoning theory (Bratman 1987) proposed as a common sense psychological framework to understand ourselves and others, based on beliefs, desires and intentions conceived as partial, hierarchical plans. These related notions of intentionality provide us with the tools to describe agents at the right level of abstraction, i.e., adopting the intentional stance; and to design agents in a compatible way with such intentional description, i.e., as practical reasoning systems.

Different aspects of intentionality and practical reasoning have been formally studied, resulting in the so called BDI logics (Rao et al. 1998). For a road map of the evolution of these formalisms, see (Singh et al. 1999; Wooldridge 2000). Implementations made use of refinement techniques, e.g., using specifications in Z language (Lightfoot 1991). A different approach is provided by AgentSpeak(L) (Rao 1996), a formal BDI model closer to the agent oriented programming paradigm. Jason (Bordini and Hübner 2006) is an extended interpreter for AgentSpeak(L) implemented in Java. Originally our agents run on a BDI architecture (Fig. 1) following dMARS specification in Z (Inverno, Kinny, Luck and Wooldridge 1997). This architecture was implemented *ex nihilo* (Guerra-Hernández et al. 2001, 2004b) in Lisp. Recently, we have ported the approach to AgentSpeak(L), implementing it in Jason (Bordini et al. 2006).

When learning is considered (Fig. 2), independently of the implementation, we assume that the BDI interpreter corresponds to the performance element in the abstract learning architecture (Russell and Norvig 1995). The performance element is responsible for deciding what to do. The learning element is intended to provide modifications to the performance element to improve the behavior of the agent. The critic provides feedback about the performance of the agent, while the problem generator suggests cases or situations which constitute informative experiences for the learning element.

What do we mean by learning intentionally? Assume that the robots shown in the fig. 3 are BDI agents. Suppose that `r2` is thinking about what object should it take.

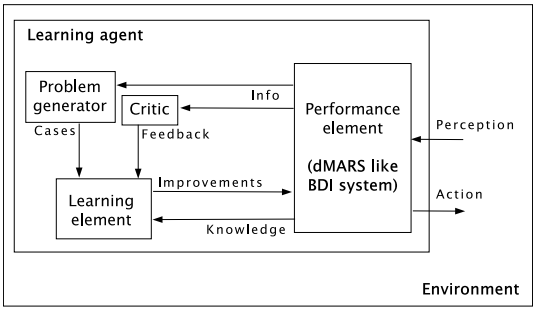

Fig. 2. Abstract learning architecture.

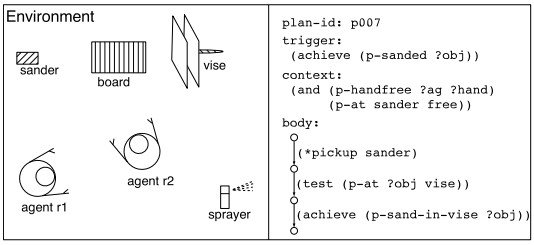

Fig. 3. Simple scenario of BDI agents and a plan.

Since it is deciding what to do, `r2` is performing practical reasoning. Suppose `r1` is figuring out what object will `r2` take. Since it is modifying its beliefs, `r1` is performing epistemic reasoning. Practical and epistemic reasoning are present explicitly in the BDI agents. They are activated by the achieve (!) and test (?) goals respectively. Learning intentionally is related with practical reasoning.

Practical reasoning lead to the adoption of intentions. For example, if the event `!p-sanded(board)` is perceived by an agent, it will look for relevant plans, i.e., the plans which trigger unifies the event. This is the case of the plan in the fig. 3:

```
@p007
!p-sanded(Obj) : p-handfree(Ag, Hand) & p-at(sander, free)
   <- .pickup(sander);
      ?p-at(Obj, vise);
      !p-sanded-vise(Obj)}.
```

In what follows, AgentSpeak(L) syntax is adopted. Plans have the form `@Id trigger : context <- body.` Then, in order to form an intention, the agent will look for an applicable plan, i.e., a relevant plan which context is a logical consequence of its current belief.

Although the agent seems to be performing a kind of epistemic reasoning to adopt their intentions, in fact it is only verifying if his practical reasons to adopt the plan as an intention are supported by its beliefs. From the role of beliefs in the theory of practical reasoning (Bratman 1987), e.g., the standard and filter of admissibility, it is clear that even when the beliefs justify the behavior of the agent, they do it as a part of a background frame that, together with prior intentions, constrain the adoption of

new intentions. In doing so, beliefs are playing a different role that the one they play in epistemic reasoning. Particularly, practical reasons to act sometimes differ from epistemic reasons. This is the case for reasonableness of arbitrary choices in Buridan cases (situations equally desirable), e.g., it is practical reasonable for agent `r2` to choose any plan in the set of relevant applicable plans to form an intention, even if there is no epistemic reason, no reason purely based on the beliefs of the agent, behind this choice. Sometimes this is called wishful thinking, and it is not epistemic reasonable for agent `r1` to belief that agent `r2` will take the `sander`, in this way.

What is relevant here, is that the contexts express practical reasons to act in some way and not in another, face to desires. The context, together with the background frame of beliefs and prior intentions, support the rational behavior of intentional agents. Contexts are the target concept, if BDI agents are going to learn about their practical reasons to satisfy desires.

3 Design issues

Once the context of the plans has been identified as the target concept, all the other issues of the abstract learning architecture can be approached.

3.1 Representation

Representations in the BDI model are based on first-order formulae. If at is an atomic formula, then the well-formed formulae (wff) of AgentSpeak(L) is given by the following grammar $\varphi ::= at|\neg at|\varphi \wedge \varphi'$. Beliefs are grounded atomic formulae, like Prolog facts. The extended interpreter Jason, considers also disjunctions of the form $\varphi \vee \varphi'$. The context of the plans is expressed as a wff which may include variables, e.g., `p-freehand(Ag, Hand) & p-at(sander, free)`. In dMARS, wff of the form $at|\neg at$ are known as belief formuae, while those including disjunctions and conjunctions are known as situation formulae. This representation has two immediate consequences for the candidate learning methods: Given the first-order nature of the representation, propositional learning methods are discarded; and the fact that the context of plans is represented as situation formulae, demands disjunctive hypothesis, e.g., decision trees.

3.2 Feedback

Getting feedback from the BDI interpreter is almost direct, since it usually detects and processes success and failure of the execution of intentions. This is done by executing a set of internal actions, e.g., adding or deleting beliefs, and posting events, accordingly to the result obtained in executions. Jason and any BDI interpreter can be extended with a special internal action, that generates a log file of training examples for the learning task. Items to built these examples include: the beliefs characterizing the situation when the plan was selected to form an intention, the label of success or failure after the execution of the intention, and the plan and the agent identifications.

3.3 Background knowledge

Although most of the time, agents do not use the knowledge they have while learning, this is not a good idea provided that BDI agents have very rich prior information. There are two possible sources of background knowledge for them. First, the plan library may be seen as their background knowledge, since plans state expected effects which, from the perspective of the agent, must hold in the environment, i.e., the event e will be satisfied with the plan p execution, and this is the case if the context of p is a logical consequence of the beliefs of the agent. Second, we keep track of predicates and functions, as well as their signatures, used to define the agents. This is used to specify the language for the target concept.

3.4 Top-down Induction of Logical Decision Trees

Because of the disjunctive nature of the context of plans, decision trees are adopted as the target representation. Top-down induction of decision trees is a widely used and efficient machine learning technique. As defined in the ID3 algorithm (Quinlan 1986) it approximates discrete target-value functions. Targets are represented as trees, corresponding to a disjunction of conjunctions of constrains on the attribute values of the instances. Each path from the decision tree root to a leaf, corresponds to a conjunction of attribute tests, and the tree itself is the disjunction of these conjunctions. However, as can be guessed, the ID3-like approaches has a propositional nature, where training examples are represented as a fixed set of attribute-value pairs. It is assumed that all the information available to learn is in the examples, i.e., the background knowledge is not considered.

Logical decision trees upgrade the attribute-value representation to a first-order representation, using the Inductive Logic Programming (ILP) setting known as learning from interpretations (Blockeel and Raedt 1998). In this setting, each training example e is represented by a set of facts. Background knowledge can be given in the form of a Prolog program B. The interpretation that represents the example is the set of all ground facts that are entailed by $e \wedge B$, i.e., its minimal Herbrand model. Observe that instead of using a fixed-length vector to represent e, as the case of attribute-value pairs representation, a set of facts is used. This makes the representation much more flexible. Learning from interpretations can be defined as follows. Given: i) A target variable Y; ii) A set of labeled examples E, each consisting of a a set of definite clauses e labeled with a value y in the domain of Y; iii) A language L; iv) A background theory B. Find a hypothesis $H \in L$ such that for all examples labeled with y: i) $H \wedge e \wedge B \models label(y)$; and ii) $\forall y' \neq y : H \wedge e \wedge B \not\models label(y')$.

The learning from interpretations setting, as the propositional case, exploits the local assumption, i.e., all the information that is relevant for a single example is localized in two ways: Information contained in the examples is separated from the information in background knowledge; and Information in one example is separated from information in other examples (Muggleton and Raedt 1994). The local assumption is relevant if we want the agents configuring their learning settings themselves.

Tilde (Blockeel, Dehaspe, Demoen, Gerda, Ramon and Vandecasteele 2000) is a learning from interpretations system to induce logical decision trees, i.e., decision trees

where every internal node is a first-order conjunction of literals. It uses the same heuristics that ID3 algorithms (gain-ratio and post-pruning heuristics), but computations of the tests are based on the classical refinement operator under Θ-subsumption, which requires the specification of a language L stating which kind of tests are allowed in the decision tree.

4 Social awareness

The awareness of other agents in the system seems to be indicative of a MAS hierarchy of increasing complexity. In a certain way, this hierarchy corresponds to the scale of intentionality discovered by D. Dennett (Dennett 1987). Levels in the awareness hierarchy are as follows:

1. At the first level, agents act and learn from direct interaction with the environment. They are not explicitly aware of other agents in the MAS. However, the changes produced by other agents in the environment may be perceived by the learning agent. For example, in the scenario of figure 3, `r1` can be specialized in painting objects, while `r2` sands them. It is possible to program the painter robot, without awareness of the sander, i.e., all `r1` has to know is that once an object is sanded, it can be painted. The true isolated learning case with one agent, may be seen as a special case of this level.
2. At the second level, agents act and learn from direct interaction with other agents using message exchange. For the example above, the sander robot can inform the painter robot, that an object is already sanded. Also, the painter agent can ask the sander robot for this information. Exchange of training examples in learning processes is also considered.
3. At the third level, agents act and learn from the observation of the actions performed by other agents in the system. It involves a different kind of awareness from that of level 2. Agents are not only aware of the presence of other agents, but are also aware of their competences, hence the painter robot is able to perceive that the sander robot is going to sand the table. Observe that this seems to involve epistemic reasoning, e.g., adopting the belief that that `r2` will sand the object.

The purpose of our BDI agents while learning, is to update context of a plan after its execution failed. If an agent can update this context after its own experience, no communication is needed and learning is performed at level 1 of the hierarchy. Otherwise, the agent will try to learn from the experience of other agents, starting a kind of collaborative goal adoption (Castelfranchi 1998) process, where the agents in the MAS sharing the same plan, cooperate with the goal of the learner agent, because they are also interested in the results of its learning process, e.g., inducing the updated context of the shared plan. The group of agents interested in a learning goal, may be seen as an emerging social structure, where the roles of learner and supervisors are dynamically assigned. An incremental version of this protocol (Bourgne, El-Fallah-Seghrouchni and Soldano 2007) is discussed in future work.

5 BDI learning agents

In order to test the AgentSpeak(L) implementation of our intentional learning approach, we decided to use the well known MAS examples, e.g., those in Jason's distribution. The idea is to modify some contexts in the plan library of the agents and observe if the they are able to learn a new context, and how does this new context compare with the original one. The example used in this section is the well known blocks world. Table 1 (analogous to figure 3) shows a situation in this environment represented by a set of beliefs, and a plan for stocking a block. The context of the plan `stack` constrains the plan the be applicable only if the destination is clear (`clear(Y)`) and the agent is holding the block to be stacked (`holding(X)`). What would happen if our definition for this plan did not consider the atom `holding(X)`? The plan would fail (the agent can not forget (-) what it did not believe) and the agent should reconsider its practical reasons to adopt this plan as an intention.

Beliefs	**Plan**
`on(b,a).`	`@stack`
`clear(b).`	`+!stack(X,Y) : clear(Y) & holding(X) <-`
`clear(c).`	`-clear(Y); -holding(X);`
`onTable(a).`	`+armEmpty; +on(X,Y);`
`onTable(c).`	`.print("Stacking ",X," on ", Y).`
`armEmpty.`	

Table 1. Some beliefs and a plan in the Blocks World.

5.1 Centralized learning (level 1)

Suppose the plan `stack` has failed because of the reasons explained above. We want the agent trying to learn why did the plan fail, if there were practical reasons to adopt it as an intention. In order to execute the learning process, the agent post the event `!pLearn(stack)` when detecting the failure of the plan. There is a relevant plan for this kind of events:

```
@learningPlan
+!pLearn(Plan) : flag(newExample)
   <- ia.makeSet(Plan);
      ia.execTilde(Plan);
      ia.updatePContex(Plan,C,V);
      ia.evalLearnedContext(Plan,C,V).
```

When the success or failure of an intention is detected, the agent keeps track of these executions as training examples. If at least one new example has been collected by the agent, it beliefs `flag(newExample)`, and the `learningPlan` becomes executable. While in dMARS failed plans are not intended again, this is not the case for Jason, where the agent can intend the plan again and collect more training examples. The internal action `ia.makeSet` configures the learning task of the agent, generating the files required by Tilde to learn:a knowledge base (`.kb`), the background knowledge

(`.bg`), and the settings (`.s`) including the language bias L. The identifier `Plan` names these files, that are generated automatically by the agent as follows.

Each learning example is coded as a model of the learning from interpretations paradigm. A model starts with a label that indicates the `success` or `failure` of the plan execution. Then a predicate `plan/2` is added to establish that the model is an instance of the execution of a particular plan by a particular agent. The model also contains the beliefs that the agent had when the plan was selected to form an intention. The label is added in the execution phase of the BDI interpreter. Table 2 shows some models generated for the plan `stack` by the agent `c1`.

```
begin(model(1)). begin(model(2)). begin(model(3)). begin(model(4)).
failure.         success.         success.         success.
plan(c1,stack).  plan(c1,stack).  plan(c1,stack).  plan(c1,stack).
clear(b).        holding(c).      holding(b).      holding(c).
clear(c).        clear(b).        clear(a).        clear(a).
onTable(a).      onTable(a).      onTable(a).      clear(b).
onTable(c).      on(b,a).         onTable(c).      onTable(a).
on(b,a).         end(model(2)).   clear(c).        onTable(b).
armEmpty.                         end(model(3)).   end(model(4)).
end(model(1)).

begin(model(5)). begin(model(6)). begin(model(7)).
success.         success.         failure.
plan(c1,stack).  plan(c1,stack).  plan(c1,stack).
holding(b).      holding(b).      clear(b).
clear(c).        clear(c).        on(c,a).
clear(a).        on(c,a).         on(b,c).
onTable(a).      onTable(a).      armEmpty.
onTable(c).      end(model(6)).   onTable(a).
end(model(5)).                    end(model(7)).
```

Table 2. Training examples for the plan `stack`, collected by agent `c1`..

Then the configuration file is generated. Following the example, it is named `stack.s`. The first part of this file is common to all configurations. It specifies the information printed while learning (talking); the minimal number of cases in a leaf; the format of the output (C4.5-like and as a logic program); and the classes for the target concept, i.e., either success or failure:

```
talking(0).
load(models).
minimal_cases(1).
output_options([c45,lp]).
classes([success, failure]).
```

The second part of the configuration file specifies the predicates to be considered while inducing the tree. The `rmode` command is used to define the language bias $\mathcal{L}$ as follows: The '`#`' sign may be seen as a variable place holder, that takes its constant values from the examples in the knowledge base. The '`+`' prefix means the variable must be instantiated after the examples in the knowledge base. The way our agent

generates this file relies on the agent definition. Table 3 shows the language bias for this example, generated automatically by the agent `c1` based on its own definition.

```
rmode(plan(Agent,Plan)).      rmode(clear(#)).
rmode(clear(X)).              rmode(clear(#)).
rmode(onTable(X)).            rmode(onTable(#)).
rmode(on(X,Y)).               rmode(on(#,+Y)).
rmode(armEmpty).              rmode(on(+X,#)).
rmode(holding(X)).            rmode(on(#,#)).
                              rmode(holding(#)).
```

Table 3. Language bias in the configuration file.

Then agent executes a modifies non-interactive version of ACE (Blockeel et al. 2000), and automatically extracts the learned hypothesis from the `stack.out` file, to accordingly modify the definition of the plan. It is also possible to ask the user to modify the plans, showing him the obtained results:

```
holding(X) ?
+--yes: [true] 5.0 [[true:5.0,false:0.0]]
+--no:  [false] 2.0 [[true:0.0,false:2.0]]
```

This example used seven models (Table 2) and the time of induction was 0.032 seconds, running on a Linux SuSe Intel Centrino 1.73 GHz. The result suggests a new plan context: `clear(Y) & holding(X)` (the original plan context!).

5.2 Decentralized learning (level 2)

Why should a BDI agent learning at level 1, try to communicate to learn? (Guerra-Hernández et al. 2004a). There are two situations under which an agent should consider communication while learning: when the agent is not able to start learning, i.e., it does not have enough examples; or when the agent is not able to find out an hypothesis to explain the failure of the plan in question. In both cases the learning agent may ask for more examples to other agents in the MAS.

Sharing examples in this way, resembles other cases of distributed learning. Vertical fragmentation uses to be a delicate issue in such settings, but not here since the examples are represented as labeled sets of definite clauses. Consider the training examples in the table 2. From the Learning from Interpretations perspective, these training examples are just models of a target concept, even if they are not homogeneous, e.g., model 1 has information about `armEmpty`, while model 2 does not. In propositional representations this implies that the attribute belongs to a different data source (vertical fragmentation). Avoiding this is very important: since our BDI learning agents face only horizontal fragmentation, the exchange of training examples is enough to achieve learning at level 2.

A group of agents is said to have the same *competence* for a given event, if they share the same plan to deal with it. Competence in our approach is defined in terms of

plans, because the event they deal with, is already included in the plan definition. Competence induce two ways for asking help: a message including the label of the plan being learned is broadcasted, then the agents sharing the same plan accept and process the message; or the agent sends the message only to the agents with the same competence. We allow both options: The internal action `ia.setTolearningMode/3` takes as arguments: The plans for which learning is enabled (it is possible to specify `all`); the agents sharing the same competence (`all` means broadcasting); and some debug options, e.g., the action `ia.setTolearningMode([stack], [c2,c3], [verbose(true)])` means that the agent will learn about plan `stack`, communicating with `c2` and `c3`, while giving some output details about the learning process.

```
@gatherExamples_f
+!gatherExamples([],Plan) : true
   <- !pLearn(Plan).

@gatherExamples_i
+!gatherExamples([Ag|T],Plan) : true
   <- .send(Ag, askAll, value([Plan,X,L,Y],example(Plan,X,L,Y)),List);
      !adoptList(List);
      !gatherExamples(T,Plan).
```

Table 4. Plans for gathering examples distributed in the MAS.

In the Jason implementation, a learning agent at level 1 always try to learn when a new example is acquired (`flag(newExample)`). So, the plan `pLearn` only can fail if the learning process does not produce an hypothesis. If this is the case, the agent post the event `!gatherExamples`. There are two applicable plans for the triggering event `+!gatherExamples` (Table 4). First, the plan `gatherExamples_i` will iterate the whole list of agents asking to each one for examples about `Plan`, while adopting answers from other agents in `List` as part of its own believes. After that, the agent tries to learn again (`gatherExamples_f`). Jason adopts the Knowledge Query Meta Language (KQML) (Moreira and Bordini 2003) for communication.

The result obtained at level 2 is the same that the obtained at level 1, the agent updates the context of plan with the original definition. But observe that following the dMARS approach to failed plans, the agent `c1` can achieve this result only at level 2. There is no way for it to collect alone the examples required to successfully learn the context. But even under the Jason approach enabling to intend again a failed plan, the examples collected by a single agent seems to contribute with few information. Also, observe that since BDI agents collect the training examples after their own experience, these examples are harder to obtain than in classic supervised Machine Learning, where they are previously collected by a supervisor; or in reinforcement learning, where agents explore in a natural way the space of hypothesis while acting. The use of ILP methods helps the agent since fewer examples may be needed to learn, provided that the background theory is relevant. Also, situating the learning agent in a MAS at level 2, may help it to collect training examples faster, taking benefit of the experience of other agents. Table 5 shows the log of a descentralized learning process.

```
[c1] saying: PLAN stack FAILED
[c1] saying: EXECUTE LEARNING FOR stack
[report] CURRENT EXAMPLES
[report] c1: example(stack,[clear(b),clear(c),onTable(a),
                    onTable(c),on(b,a),do,counter(1),
                    armEmpty],failure)
[c1] saying: CAN'T LEARN. CONTEXT IS clear(Y)
[c1] saying: NO NEW EXAMPLES, ASKING FOR HELP
...
[c1] saying: EXECUTE LEARNING FOR stack
[report] CURRENT EXAMPLES
[report] c1: example(stack,[clear(b),clear(c),onTable(a),onTable(c),
                    on(b,a),do,counter(1),armEmpty],failure)
[report] c1: example(stack,[holding(c),clear(b),clear(c),onTable(a),
                    on(b,a),do,counter(1)],success)[source(self)]
...
[report] c1: example(stack,[clear(b),clear(c),onTable(a),onTable(c),
                    on(b,a),do,counter(4),armEmpty],failure)
...
[showLearningRes] THE INDUCED TREE FOR stack IS:
[showLearningRes] holding(-A) ?
[showLearningRes] +--yes: [true] 5.0 [[true:5.0,false:0.0]]
[showLearningRes] +--no:  [false] 4.0 [[true:0.0,false:4.0]]
[c1] saying: THE NEW CONTEXT FOR stack IS clear(Y) & ((holding(X)))
```

Table 5. Decentralized learning in the blocks world.

The result of a learning process is shared by the agents in the MAS. If the user or the agent modify the plan definition accordingly to the decision tree found, this change affects automatically all agents having this plan in its library. This justifies the cooperative goal adoption strategy: sharing examples is easy, while learning is not, but learning sometimes is not possible without cooperation, so cooperation became advantageous.

6 Results and future work

Our proposal for intentional learning in the context of BDI agency constitutes a relevant step towards sapience. The agents adopting this approach are able to perform a limited form of introspection: evaluate the validity of their practical reasons and to modify such reasons trying to avoid failures. All of this in an autonomous way, as it is required for sapient agents. The agents adopting the second level of social awareness exploits the interaction with other agents to learn, as it is expected for a sapient agent. All of this exploiting social communication based on Speech Acts. The BDI agents, extended in this way, seems closer to the intended characterization of sapient agents.

Of course learning about the practical reasons to adopt a plan as an intention is only one of the possible issues a sapient agent should learn about. What is relevant is that the kind of learning proposed here is performed according to the principles of Intentionality and Practical Reasoning. Learning about other issues must take into account the same considerations.

Future work is related with Smile ;-) protocol (Bourgne et al. 2007), an acronym for Sound Multi-agent Incremental LEarning. Each agent in the MAS is assumed to be

able to learn from the information it perceives. An agent i is defined as $a_i = \langle B_i, K_i \rangle$, where B_i is the belief or knowledge state of agent i; and K_i is the information it perceives. It is assumed that all agents in the MAS shared some common knowledge, denoted by B_C so that $\forall i \ B_C \subseteq B_i$. Common knowledge introduces dependences among the agents: If an agent a_i updates its state B_i obtaining B'_i, but also modifying B_C into B'_C, then every agent a_j must update its state B_j to obtain B'_j so that $B'_C \subseteq B'_j$. An agent a_i updates it state B_i because of new information K_i is perceived. The agents are supposed to be able of verifying consistence between states and information, e.g., logical consequence. An agent a_i is *a-consistent* if and only if $Cons(B_i, K_i)$ is true, i.e., the information it perceives is a logical consequence of its beliefs. An agent a_i is *mas-consistent* if and only if $Cons(S_i, K_i \cup K)$ is true, where $\forall j \neq i \ K = \bigcup K_j$. A MAS is *consistent* if every agent in the system is *mas-consistent*.

The protocol depends on a consistency revision mechanism M with the following properties: it is *locally efficient* in the sense that an agent i can always update B_i; it is *additive* in the sense that $Cons(B_i, K_1) \wedge Cons(B_i, K_2) \implies Cons(B_i, K_1 \cup K_2)$; it is *coherent* in the sense that $\forall i, j, k \ Cons(B_i, K_i) \wedge Cons(B_j, K_j) \implies (Cons(B_i, K_i \cup k) \iff Cons(B_j, K_j \cup k))$. M is said *a-consistent* and *mas-consistent*, if it preserves such properties of the agent applying M. The consistency revision mechanism M is said *strong mas-consistent* if its application by an agent a_i preserves the *mas-consistency* of $\forall j \neq i \ a_j$.

Bourgne et al. (Bourgne et al. 2007) proved that *mas-consistency* can be ensured for the propositional case by the reiterative application of a mechanism M by the learner agent, followed by some interactions with other agents restoring the *a-consistence*, until no inconsistent information is produced in such interactions. The mechanism is triggered by an agent a_i receiving some information k inconsistent: $Cons(B_i, k) = False$. An interaction between a_i and some critic a_j is denoted by $I(a_i, a_j)$ and performed as follows: a_i sends a_j the updated common beliefs B'_C produced by the local application of M, i.e., a_i is *a-consistent*; a_j checks B'_j induced by the updated B'_C. if these modifications preserve its *a-consistency*, a_j adopts them, sending a_i its acceptance; otherwise it sends its refusal with some information k' s.t. $Cons(B'_j, k') = False$. The protocol finishes when no k' is produced by any agent. It is also suggested that if the response of the agents is minimal and serial (k' is one inconsistent example and the a_i sends B'_C sequentially to other agents), then the protocol minimizes the amount of information transmitted by the agents.

Our current approach uses a consistency revision mechanism M that is not locally efficient (Tilde). It forces the learning agent to collect all the training examples available to start learning. We are implementing an incremental version of the revision mechanism (TildeLDS) which will enable us to fully adopt Smile (B_C is the set of plans shared by the agents). The learning agent will be able to recover consistency locally and communicate the learned context to other agents to compute mas-consistency.

7 Acknowledgements

The second author is supported by Conacyt scholarship 197819. The intentional learning approach was conceived with Amal El-Fallah Seghrouchni and Henry Soldano at Paris 13. Gauvain Bourgne has shared with us his work on Smile.

References

Bratman, M. Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA., USA, and London, England, 1987.

Blockeel, H. and De Raedt, L. (1998) Top-down induction of first-order logical decision trees. Artificial Intelligence, 101(1–2):285–297.

Blockeel, H., Dehaspe, L., Demoen, B., Gerda, J., Ramon, J. and Vandecasteele, H. (2000) Executing query packs in ILP. In: Cussens, J. and Frish, A., editors. Inductive Logic Programming, 10th International Conference, ILP2000, London, U.K., LNAI 1866:60–77, Springer-Verlag, Berlin Heidelberg.

Bordini, R.H. and Hübner, J.F (2006) BDI agent programming in AgentSpeak using Jason. In Toni, F. and Torroni, P., editors. Proceedings of the Sixth International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI), London, UK, 27-29 June, 2005, Revised Selected and Invited Papers, LNCS 3900:143–164, Springer-Verlag, Berlin Heidelberg.

Bourgne, G., El-Fallah-Seghrouchni, A. and Soldano, H. (2007) SMILE: Sound Multiagent Incremental LEarning ;-) AAMAS'07, May 14-17 2007, Honolulu, Hawai'i, USA. ACM.

Castelfranchi, C. (1998) Modelling social action for AI agents. Artificial Intelligence, 103(1998):157–182, 1998.

Dennett, D.C. The Intentional Stance. MIT Press, Cambridge MA., USA, 1987.

Georgeff, M., Pell, B., Pollak, M. Tambe, M. and Wooldridge, M. (1999) The Belief-Desire-Intention Model of Agency. In: Müller, J., Singh, M. and Rao, A., editors. Proceedings of the 5th International Workshop on Intelligent Agents V: Agent, Theories, Architectures, and Languages (ATAL-98). LNAI 1555:1–10, Springer Verlag, Berlin Heidelberg.

Guerra-Hernández, A., El-Fallah-Seghrouchni, A. and Soldano, H. (2001) BDI Multiagent learning based on First-Order induction of Logical Decision Trees. In: Intelligent Agents Technology: Research and Development. Proceedings of the 2nd Asia-Pacific Conference on IAT, Maebashi, Japan, November, 2001, World Scientific, Singapour, 160–169.

Guerra-Hernández, A., El-Fallah-Seghrouchni, A. and Soldano, H. (2004) Distributed learning in Intentional BDI Multiagent Systems. In Baeza-Yates, R., Marroquín, J.L., and Chávez, E., editors. Proceedings of the Fifth Mexican International Conference on Computer Science (ENC'04), pages 225–232, USA, 2004. IEEE Computer Society.

Guerra-Hernández, A., El-Fallah-Seghrouchni, A. and Soldano, H. (2004) Learning in BDI multiagent systems. In: Dix, J. and Leite, J., editors. CLIMA IV. Computational

Logic in Multi-Agent Systems: 4th International Workshop, Fort Lauderdale, FL, USA, January 6–7, 2004. Selected and Invited Papers. LNAI 325:218–233, Springer Verlag, Berlin Heidelberg.

Guerra-Hernández, A., El-Fallah-Seghrouchni, A. and Soldano, H. (2005) On Learning Intentionally. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 9(25):9–18, Sept., 2005.

d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. (1997) A formal specification of Dmars. In Singh, M., Rao, A. and Wooldridge, M., editors. Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages. LNAI 1365:155–176, Springer Verlag, Berlin Heidelberg.

Lightfoot, D. Formal Specification Using Z. Macmillan Computer Science Series. The Macmillan Press LTD, London, UK, 1991.

Moreira, Á. F. and Bordini, R. (2003) An operational semantics for a bdi agent-oriented programming language. In Omicini, A., Sterling, L. and Torroni, P., editors. Declarative Agent Languages and Technologies, First International Workshop, DALT 2003, Melbourne, Australia, July 15, 2003, Revised Selected and Invited Papers. LNCS 2990:135–154, Springer Verlag, Berlin Heidelberg.

Muggleton, S. and de Raedt, L. (1994) Inductive logic programming: Theory and methods. Journal of Logic Programming, 19:629–679.

van Otterlo, M., Wiering, M., Dastani, M. and Meyer, J.-J. (2003) A characterization of Sapient Agents. In: IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems, KIMAS 2003. Boston MA., USA.

Quinlan, J. (1986) Induction of decision trees. Machine Learning, 1(1):81–106.

Rao, A. (1996) AgentSpeak(L): BDI agents speak out in a logical computable language. In: van Hoe, R., editor. Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands.

Rao, A. and Georgeff, M. (1998) Decision procedures of BDI logics. Journal of Logic and Computation, 8(3):293–344.

Russell, S. and Norvig, P. Artificial Intelligence, a modern approach. Prentice-Hall, New Jersey NJ, USA, 1995.

Singh, M., Rao, A., Georgeff, M. (1999) Formal Methods in DAI: Logic-based representations and reasoning. In: Weiss, G., editor. Multiagent Systems, a modern approach to Distributed Artificial Intelligence. MIT Press, Cambridge MA., USA.

Wooldridge, M. Reasoning about Rational Agents. MIT Press, Cambridge MA., USA, 2000.