Representing Agent Reasoning With Meta-Knowledge on ASP Modules Combination

Tony Ribeiro
National Institute of Informatics
Tokyo, Japan
ribeiro@nii.ac.jp

Katsumi Inoue
National Institute of Informatics
Tokyo, Japan
ki@nii.ac.jp

Gauvain Bourgne
????
Paris, France
bourgne@nii.ac.jp

ABSTRACT

In this work, we focus on multi-agent systems in dynamic environment. Our interest is about individual agent reasoning in such environment. For reasoning in dynamic environment, an agent needs to be able to manage his knowledge in a non-monotonic way. To reach his goals in a changing environment, an agent needs to adapt his behaviours regarding the current state of the world. Our objective is to define a method which makes easier to design agent knowledge and reasoning in such environment. We use the expressivity of answer set programming to represent agent knowledge. To design agent reasoning, we propose a method based on ASP modules combination and meta-knowledge. We also propose a framework to implement and use this method in multi-agent systems.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design

Keywords

Multi-Agents System, Answer Set Programming, Meta-knowledge, ASP modules

1. INTRODUCTION

In Multi-agents systems, reasoning in dynamic environment imply to deal with some interesting problem. In such environment an agent have to adapt to the evolution of the world to achieve his goals. His knowledge have to be update consequently to world changes by adding new observations or revise old ones. The behaviour of an agent need to be suitable to the current situation and able to evolve with it. Design such kind of reasoning is an interesting challenge, its why in this work focus on the simplification of this task.

Answer set programming which is a specification and an implementation language is very suitable to represent agent knowledge and reasoning. The methods we introduce in this work is base on ASP modules and their combinations. ASP modules allows to represent agent behaviours in an intuitive and elegant way. We divide agent knowledge into different modules and use different modules combinations for reasoning. Our agent do meta-reasoning: they use knowledge on modules combinations to adapt their reasoning and so on their behaviour to the situation.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

2. MULTI-AGENTS SYSTEMS

Definition 1 (Agent). *An agent is an entity which is able to reason and communicate with his fellows. The knowledge of an agent can be divide in two part:*

- *K a set of not revisable knowledge composed of :*
  - *T the common theory*
  - *O a set of observations*
- *B a set of revisable knowledge*

T is the common knowledge of all agent of a multi-agent system, it is considered as certain and not revisable. A common theory can also be limited to a group of agent which not contains all the system. Observations are informations retrieved from the environment, it represent the current state of the world. If we consider that sensors are perfects then a current observation is not revisable. At time step 0, the knowledge of an agent is his common theory. This knowledge is extended by adding observations and its consequences at each new time step.

B is agent's believes, it represent informations supposed as true by the agent and which are revisable. In a dynamic environment, if current observations are certain its not the case of past ones. If an agent assume that past observations are correct until new ones prove the contrary his memory is a part of his beliefs. For example if an agent a see an agent b at position p at time T. At $T+1$, a does not see any more the position p but he also does not see agent b. Then a can assume that b is always at position p at $T+1$ and when a will see again the position p or b he will revise his memory if needed.

Definition 2 (Multi-agents System). *A multi-agents system (MAS) is a network of agent where agent can interact. Such a system can be represented by a graph where:*

- *nodes represent agents*
- *edges represent communication links*

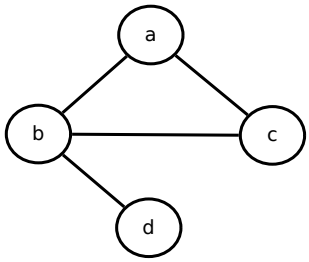

Figure 1: A multi-agents system with 4 agents and 4 communication links.

Figure 2 represent a multi-agents system with bilateral communication. This system is composed of four agents a, b, c, d and four communication links between : a and b, a and c, b and c, b and d. In this example, edge (a,b) means that a can send messages to b and vice versa.

Many works focus on specific aspect of multi-agents systems such as communications, reasoning, learning, planning, etc. Regarding communications, in [5, 6] authors are interesting by propagation of information in a group of agent. There is also interesting works like [18], where agents can lie to achieve their goals. In this work, agents are in competition and tried to persuade others agents to reach their objectives. Others work focus on group planning, in [17] they consider a hierarchy between agent of a MAS. This hierarchy allows more flexibility of planning by prefer satisfaction of hight grade agent. Its also simplify communications between agents: an agent does not have to justify his choice to his subordinate.

3. ANSWER SET PROGRAMMING

Definition 3 (Rule). *Let H, a_i and b_i literals and the rule*

$$H \leftarrow a_1, \ldots, a_n, \neg b_1 \ldots, \neg b_m.$$

If all a_i are true and all b_i are not true then H is true. H is the head of the rule and the conjunction of a_i, $\neg b_i$ is the body.

Answer set programming (ASP) is a form of declarative programming based on the stable model semantics of logic programming. An ASP program describes a problem, not how to solve it. Such program is composed of facts, rules and constraints. Rule with an empty body: $H.$ are facts. Rules without head: $\leftarrow a_1, \ldots, a_n, \neg b_1 \ldots, \neg b_m.$ are constraints. When conditions of constraints are evaluated as true its imply $\bot$. In this paradigm the search of solution is resume to compute answer sets of ASP program. An answer set is a set of facts which is consistent with all constraints of the program, its facts are deduce from facts and rules of the program. To compute it we run a solver on such program.

Example 1. *An ASP program composed of one fact, three rules and one constraint :*

rain.
stay $\leftarrow$ *not go_out.*
go_out $\leftarrow$ *not stay.*
wet $\leftarrow$ *rain, go_out, not umbrella.*
$\leftarrow$ *wet.*

In example 1 {*rain*, *stay*} is an answer set of this ASP program but {*rain*, *go_out*, *wet*} is not, because it is not consistent with the constraint.

Many research work concern ASP or use it to represent knowledge such as [3] or [17]. In the first one, the authors use it to represent the knowledge of an agent about the knowledge of others agents. In the second one, they focus on the flexibility of reasoning by introducing soft constraints: constraints which can be violate in some case. Others works like [8] design ASP methods to improve some specific property of agent reasoning. In this work the author focus on reactivity by using ASP modules where constraints are used as actions trigger. In [7], the author discuss about integration of ASP modules into agent reasoning. There is also works like [1] or [9] where they discuss about possible extensions ASP paradigm.

4. DYNAMIC ENVIRONMENT

Our interest is about representing agent reasoning in dynamic environment. To make our work more understandable we will follow an intuitive example along our propositions: a survival game which represent a MAS in a dynamic environment. In this game there are three groups of agents: wolfs, rabbits and flowers. Each kind of agent have specific goals and behaviours. To be simple, wolfs eat rabbits and rabbits eat flowers.

Wolfs have only one goal: feed themselves. To reach this goal they have to catch and eat rabbits. A wolf can be in two situations: a prey is in sight or not. If there is no rabbit in the sight range of a wolf, the predator have to explore his environment to find one. When a prey is spotted, a wolf will try to perform a sneaky approach if he is not spotted himself, otherwise the predator will rush on his target. To resume, a wolf have three behaviours: exploration, approach and attack.

Rabbits have two goals : feed and survive. On a first hand, a rabbit have to eat flowers and on an other hand it have to escape from wolf. Like a wolf, if no prey is in sight a rabbit need to explore its environment. But by doing this, a rabbit can find preys or predators. The exploration behaviour of a rabbit is more complex than the one of a wolf. For example: when a rabbit move it will choose the position with the best sight range for easily spotted its predators. When a wolf is spotted, survive is more important than feed then a rabbit have to hide or run away. To resume, a rabbit have three behaviours: exploration, hide, run away.

Flowers are passive agents: the environment have no effect on their behaviour. The only goal of a flower is to reproduce. A flower produce regularly a new one after a certain amount of time. Knowledge of a flower does not change regarding time. A flower is an independent agent, it does not care about others agents.

5. ASP MODULES

An ASP module is an ASP program which have a specific form and a specific use. The first advantage of these modules is their simplicity: a module is a little program which represent specific knowledge. We can have a module which contain observations about surroundings, an other one to define what is a prey and a module dedicated to compute path. To obtain all paths to surroundings preys an agent will combines this three modules. By combining modules

Figure 2: A MAS in a dynamic environment where an agent is a wolf, a rabbit or a flower: Wolfs eat rabbits and rabbits eat flowers.

an agent can produce knowledge, it the purpose of our ASP modules.

5.1 Background theory

Definition 4 (Rules module). *A rules module is a set of rules which represent knowledge about a specific domain. The content of such module is static: it does not change regarding time. The purpose of these modules is to organise knowledge representation and produce new knowledge by combine it with others modules.*

Example 2. *Two example of rules module.*
A rules module of a rabbit about food chain:

predator(wolf).
fellow(rabbit).
prey(flower).

A rules module about path:

$path(X,Y) \leftarrow edge(X,Y).$
$path(X,Y) \leftarrow path(X,Z),\ edge(Z,Y).$

5.2 Observations

Definition 5 (Observations module). *An observations module is a set of facts which represent related observations. The content of such module is dynamic: it change regarding time. An agent use it like a specific memory database. The purpose of these modules is to organise observations to facilitate their use and update.*

Example 3. *Two example of observations module.*
An observations module of a rabbit about wolfs positions :

position(wolf,0,3).
position(wolf,2,5).
position(wolf,2,6).

An observations module of a rabbit about himself :

me("Bugs Bunny").
position("Bugs Bunny",0,0).
type("Bugs Bunny",rabbit).

5.3 Combinations

With this two kind of modules we can represent the background theory and observations of an agent. The idea now is that we can combine our modules to produce different

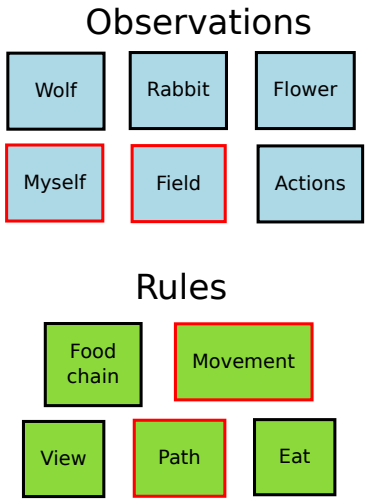

Figure 3: Knowledge base of a wolf, observations modules are blue ones and rules modules are green. The module combination {*Myself*, *Field*, *Movement*, *Path*} will produce all possible movement of the agent.

kind of knowledge. The figure 3 represent the knowledge of a wolf by a set of observations modules and rules modules. In this example, by combining modules *Myself*, *Field*, *Movement* and *Path* a wolf will produce all its possibility of movement. The combination {*Rabbit*, *Field*, *Movement*, *Path*} will produce all possible movements that rabbits in sight can perform. By combining {*Myself*, *Field*, *Rabbit*, *Movement*, *Path*, *Food Chain*, *Eat*} a wolf will produce all movements which allow to eat a rabbit.

We can see here a first advantage of knowledge modularity: a module can be used for multiple purpose. If we take the example of modules *Movement* and *Path*, an agent can use it to reason about its movement possibilities and predict other agent movements.

5.4 Behaviours

In the last section we presented different way to combine rules and observations modules. But these combinations are also a kind of knowledge and then can be known by the agent. Its a kind of meta-knowledge on our knowledge, more precisely in this case it is knowledge about the use of knowledge. This meta-knowledge is a part of the background theory of an agent and could be represented by rules modules. But the singularity of this knowledge involve to dedicated a new kind of modules to represent it.

Definition 6 (Meta-knowledge module). *A meta-knowledge module is a set of rules which define the conditions to use an ASP module. The content of such module does not change regarding time. It contains knowledge on modules combination. The purpose of these modules is to guide reasoning and represent dynamic behaviours.*

A module combination can represent agent behaviour and by using meta-knowledge modules we can represent it in a elegant way. Most simple behaviour can be represented

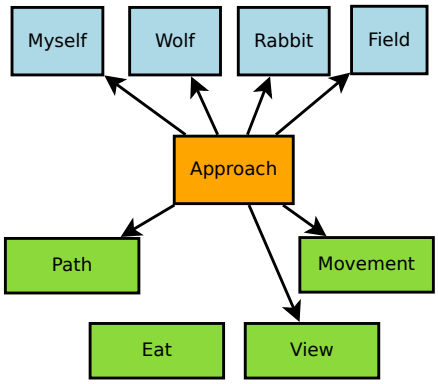

Figure 4: A meta-knowledge module about approach behaviour, with arrows on modules it knowns. Here, *Approach* define the module combination to use to approach a rabbit without being spotted.

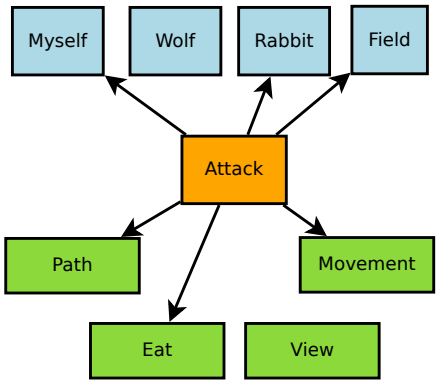

Figure 5: A meta-knowledge module about attack behaviour. Contrary to figure 5 module *Eat* is use but not *View* and *Wolf*. Here, *Attack* define the module combination to use to quickly catch a rabbit.

by a simple list of modules to combine, like in example 5 and figure 5. The module combination {*Myself*, *Field*, *Rabbit*, *Movement*, *Path*, *Food Chain*, *Eat*} and {*Myself*, *Field*, *Wolf*, *Rabbit*, *Movement*, *Path*, *Food Chain*, *View*} can respectively represent the attack and the approach behaviour of a wolf. This two combinations of modules can be known by meta-knowledge modules, in this case theses modules have knowledge on rules and observations modules.

EXAMPLE 4. *A meta-knowledge module which describe the approach behaviour of wolf:*

use_module("Myself").
use_module("Wolf").
use_module("Rabbit").
use_module("Field").
use_module("Food chain").
use_module("Path").
use_module("Movement").
use_module("View").

EXAMPLE 5. *A meta-knowledge module which describe the attack behaviour of wolf:*

use_module("Myself").
use_module("Rabbit").
use_module("Field").
use_module("Food chain").
use_module("Path").
use_module("Movement").
use_module("Eat").

To represent more complex behaviour we can have meta-knowledge modules which have knowledge on other meta-knowledge modules. For example we can represent the hunting behaviour of a wolf with such module, like in example 6 and figure 6. If a wolf is spotted it will use the *Attack* module otherwise it will use the *Approach* module. Regarding the situation the hunting behaviour will be a sneaky approach or an attack rush. Such meta-knowledge modules allows to represent dynamic behaviours which are dependent of the evolution of the environment.

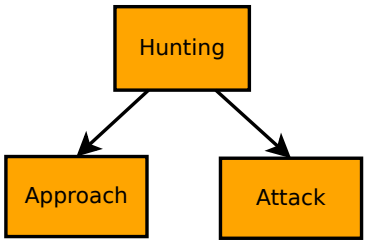

Figure 6: A meta-knowledge module with knowledge about meta-knowledge module. Here, regarding the situation the hunting behaviour correspond to a sneaky approach or an attack rush.

EXAMPLE 6. *A meta-knowledge module which describe the hunting behaviour of wolf:*

use_module("Attack") ← *spotted.*
use_module("Approach") ← *not spotted.*

5.5 Organisation

With theses three kind of ASP modules we can now represent agent knowledge and reasoning. The figure 7 represent the knowledge of a wolf by a graph of modules. A node is an ASP module, an edge represent knowledge about modules. In this figure, the behaviour of wolf is divide into exploration and hunting. The Hunting behaviour is itself divide into approach and attack behaviour. We can see here that meta-knowledge modules allows to represent agent behaviours in an elegant and intuitive way.

6. REASONING

In the last section we showed how to represent agent knowledge and behaviours with ASP modules. Our design is also dedicated to guide agent reasoning step by step. To allow an agent to use our method for reasoning in dynamic environment we implemented a framework. Algorithm 1 represent a simplified version of this tool.

Input is a set of modules and output is a set of actions which an agent can perform on its environment. It start

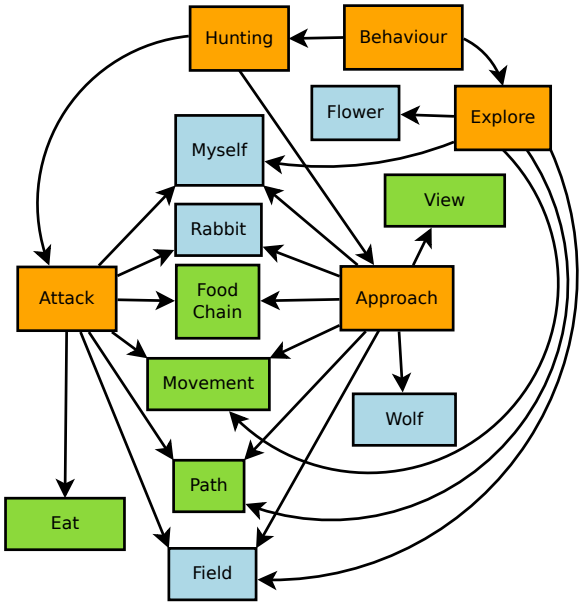

Figure 7: Representation of the knowledge of a wolf by a graph of ASP modules.

by compute the combination of input modules and retrieve only one of its answer sets by using the solver *clingo*. This answer set is selected randomly from all answer set of the module combination. From this answer set we retrieve different commands by extracting keywords. By parsing these keyword we extract the combination of modules to use in the next reasoning step and a set of action to perform. When the answer set is totally parsed, the cycle restart with the new combination of modules. Finally, when there is no more module to combine it return the set of actions to perform.

Let's take again the example of the figure 7 and let's suppose that the wolf just receive a set of new observation from its sensors. First our agent will combine theses new observations with the module *Behaviour* to decide if it have to explore the environment or hunt a prey. Let's suppose that new observations contains the position of a rabbit, then the next combination of module to use is {*Hunting*}. Nothing say that the wolf is spotted so the next combination of module to use is {*Approach*}. Finally the module *Approach* will specify that the last combination of module to use is {*Myself*, *Field*, *Wolf*, *Rabbit*, *Movement*, *Path*, *Food Chain*, *View*}. The wolf will then perform a movement which allow him to approach the rabbit without being spotted.

In this example we can see that only a part of the knowledge is use for reasoning: modules *Explore*, *Attack*, *Flower* and *Eat* are not used. The same knowledge can be represented in only one ASP program, but in this case all knowledge is use. Regarding this monolithic representation our method allows to reduce reasoning search space.

Reasoning always finish when an agent use a module graph which is acyclic like in figure 7. To represent behaviours its quite intuitive to make a tree of meta-knowledge modules. Where the root is the most abstract behaviour and more specific one are deeper.

7. EXPERIMENTS

Algorithm 1 Reasoning

```
1: INPUT : M a set of ASP modules
2: OUTPUT : A a set of actions

3: A a set of actions
4: AS an answer set

5: A ← ∅
6: AS ← ∅

7: repeat
8:    // Select one answer set of M
9:    AS ← clingo(M)
10:   M ← ∅

11:   // Extract framework commands
12:   for every literal L of AS do
13:      if L == "use_module(Module)" then
14:         M ← M ∪ Module
15:      end if
16:      if L == "action(Action)" then
17:         A ← A ∪ Action
18:      end if
19:   end for
20: until M == ∅ // No more modules to combine

21: return A
```

8. CONCLUSIONS

We provide a method based on ASP modules to design agent knowledge and reasoning. This representation allows to intuitively implements dynamic behaviours. We also provide a framework which allows agent reasoning via ASP modules. Regarding monolithic ASP program, a first improvement of our method is the reduction of reasoning search space. Its also cause reduction of code size because modules are reusable for multiple purposes.

In this paper we use ASP but this method can be easily adapted for other programming language, logic or not.

9. REFERENCES

[1] C. Baral, S. Anwar, and J. Dzifcak. Macros, macro calls and use of ensembles in modular answer set programming. In *AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 1–9, 2006.

[2] C. Baral and G. Gelfond. On representing actions in multi-agent domains. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pages 213–232, 2011.

[3] C. Baral, G. Gelfond, T. C. Son, and E. Pontelli. Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge. In *AAMAS*, pages 259–266, 2010.

[4] G. Bourgne. *Hypotheses refinement and propagation with communication constraints*. PhD thesis, University Paris IX Dauphine, 2008.

[5] G. Bourgne, K. Inoue, and N. Maudet. Abduction of distributed theories through local interactions. In *ECAI*, pages 901–906, 2010.

[6] G. Bourgne, K. Inoue, and N. Maudet. Towards efficient multi-agent abduction protocols. In *LADS*, pages 19–38, 2010.

[7] S. Costantini. Integrating answer set modules into agent programs. In *LPNMR*, pages 613–615, 2009.

[8] S. Costantini. Answer set modules for logical agents. In *Datalog*, pages 37–58, 2010.

[9] W. Faber and S. Woltran. Manifold answer-set programs and their applications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pages 44–63, 2011.

[10] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91, 2007.

[11] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[12] D. Hatano and K. Hirayama. Dynamic sat with decision change costs: Formalization and solutions. In *IJCAI*, pages 560–565, 2011.

[13] R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Ann. Math. Artif. Intell.*, 25(3-4):391–419, 1999.

[14] R. A. Kowalski and F. Sadri. Abductive logic programming agents with destructive databases. *Ann. Math. Artif. Intell.*, 62(1-2):129–158, 2011.

[15] M. Nakamura, C. Baral, and M. Bjäreland. Maintainability: A weaker stabilizability like notion for high level control. In *AAAI/IAAI*, pages 62–67, 2000.

[16] P. Nicolas and B. Duval. Representation of incomplete knowledge by induction of default theories. In *LPNMR*, pages 160–172, 2001.

[17] D. V. Nieuwenborgh, M. D. Vos, S. Heymans, and D. Vermeir. Hierarchical decision making in multi-agent systems using answer set programming. In *CLIMA*, pages 20–40, 2006.

[18] C. Sakama, T. C. Son, and E. Pontelli. A logical formulation for negotiation among dishonest agents. In *IJCAI*, pages 1069–1074, 2011.

[19] N. Tran and C. Baral. Hypothesizing about signaling networks. *J. Applied Logic*, 7(3):253–274, 2009.