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

2. STATE OF THE ART

3. 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.

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.

Figure 1: A MAS in a dynamic environment where an agent is a wolf, a rabbit or a flower: 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.

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

4.1 Background theory

Definition 1 (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 1. *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) :- edge(X,Y).
path(X,Y) :- path(X,Z), edge(Z,Y).

4.2 Observations

Definition 2 (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 2. *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).

4.3 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 kind of knowledge. The figure 2 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.

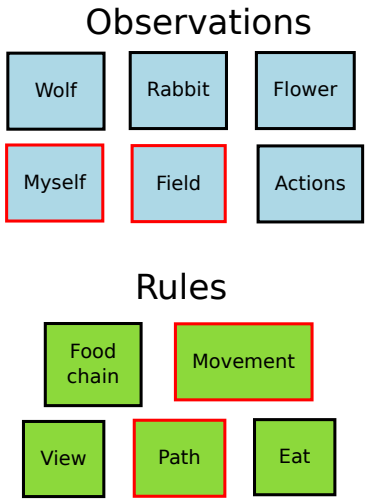

Figure 2: Knowledge base of a wolf, observations modules are blue ones and rules modules are green. The module combination {*Myself*, *Field*, *Movement*, *Path*} will produce all possible movement of the agent.

4.4 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 3 (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 by a simple list of modules to combine, like in example 4 and figure 4. 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 3. *A meta-knowledge module which describe the approach behaviour of wolf:*

use_module("Myself").
use_module("Wolf").
use_module("Rabbit").

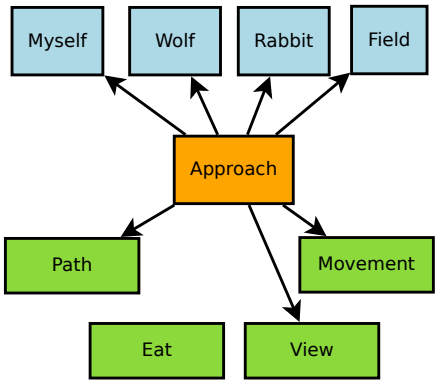

Figure 3: A meta-knowledge module about approach behaviour, with arrows on modules it knowns. Here, *Approach* define the module combination to use to approach a rabbit without being spotted.

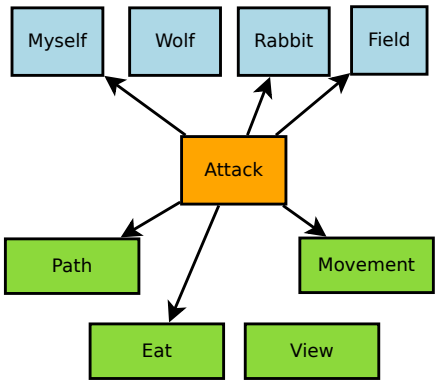

Figure 4: A meta-knowledge module about attack behaviour. Contrary to figure 4 module *Eat* is use but not *View* and *Wolf*. Here, *Attack* define the module combination to use to quickly catch a rabbit.

use_module("Field").
use_module("Food chain").
use_module("Path").
use_module("Movement").
use_module("View").

EXAMPLE 4. *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 5 and figure 5. If a wolf is spotted it will use the *Attack* module

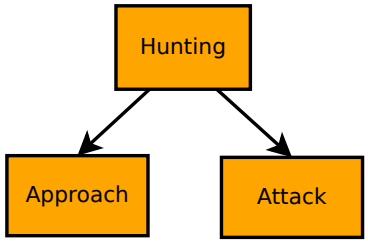

Figure 5: A meta-knowledge module with knowledge about meta-knowledge module. Here, regarding the situation the hunting behaviour correspond to a sneaky approach or an attack rush.

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.

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

use_module("Attack") :- spotted.
use_module("Approach") :- not spotted.

4.5 Organisation

With theses three kind of ASP modules we can now represent agent knowledge and reasoning. The figure 6 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.

5. 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 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 6 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

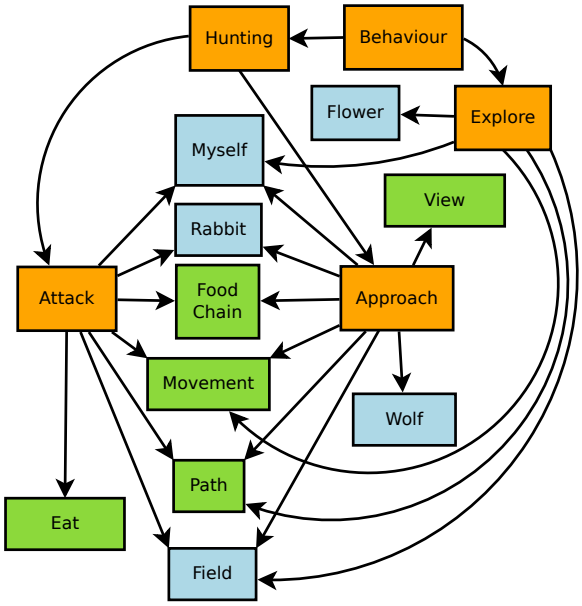

Figure 6: Representation of the knowledge of a wolf by a graph of ASP modules.

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.

6. EXPERIMENTS

7. CONCLUSIONS

We provide a method based on ASP modules to design agent knowledge. 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.

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
```