Using Meta-Knowledge on ASP Modules Combination for modular reasoning in Multi-Agent System

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
Université Pierre et Marie Curie
Paris, France
bourgne@nii.ac.jp

ABSTRACT

In this work, our interest is about multi-agent systems in dynamic environment. Here, we focus on one agent regarding knowledge representation and reasoning. In such environment, an agent needs to be able to manage his knowledge in a non-monotonic way. To reach his goals in a changing context, he have to adapt his beliefs and behaviour regarding the current state of the world. Our objective is to define a method which makes easier to design agent knowledge and reasoning in a dynamic context. For this purpose, we use the expressibility of answer set programming to propose a method based on ASP modules combinations. By using meta-knowledge on these combinations we represent dynamic behaviours and meta-reasoning. We also propose a framework and a algorithm 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, ASP modules

1. INTRODUCTION

In recent years, many works focus on specific aspect of multi-agents systems such as communications, reasoning, learning or planning. Regarding communications, in [3, 4] authors are interesting by propagation of information in a group of agent. There is also interesting works like [9], 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 [8] 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.

Our interest is about representing agent reasoning in dynamic environment. In Multi-agents systems, reasoning in dynamic environment implies to deal with some interesting problems. In such environment, an agent has to adapt to the evolution of the world to achieve his goals. His knowledge have to be updated regarding world changes by adding new observations or revising old ones. The behaviour of an agent needs to be suitable to the current situation and able to evolve with it. Designing such kind of reasoning is an interesting challenge which is the purpose of this work.

In this work We divide the knowledge of an agent in two part:

- K a set of not revisable knowledge composed of:
  - C the common theory
  - O the current observations
- B a set of revisable knowledge composed of:
  - M the memory of past observations

T is the common background 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 does not contain all the system. Observations are informations retrieved from the environment, it represents the current state of the world. If we consider that sensors are perfect 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 beliefs, it represent informations supposed as true by the agent and which are revisable. In a dynamic environment, if current observations are certain, it is not the case of past ones. If an agent assume that past observations are correct until new ones prove the contrary his memory M 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.

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

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.

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 has 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 not be eaten. On a first hand, a rabbit has to eat flowers and on an other hand it has to escape from wolf. Like a wolf, if no prey is in sight a rabbit needs to explore its environment, but he can find preys and predators. The exploration behaviour of a rabbit is more complex than the one of a wolf. When a rabbit moves it could choose the position with the best sight range to spotted its predators. If a wolf is spotted, surviving is more important than feeding then a rabbit has to hide or run away. To resume, a rabbit have four behaviours: exploration, feed, hide and run away. In this paper, examples and experiments focus on rabbit agent knowledge representation and reasoning.

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

The methods introduced by this work is based on modular knowledge representation. Agent knowledge is divided into different modules which are combined for reasoning. This representation also allows agent to perform meta-reasoning by using knowledge on modules combinations. One interest is that an agent can adapt his reasoning and so on his behaviour to environment changes.

2. ANSWER SET PROGRAMMING

Answer set programming (ASP) which is a specification and an implementation language is very suitable to represent agent knowledge and reasoning. ASP is a form of declarative programming based on the stable model semantics of logic programming. In this section, we briefly introduce ASP paradigm and semantics.

DEFINITION 1 (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.

An ASP program declare a problem, not how to solve it. Such program is composed of facts, rules and constraints. A rule with an empty body: $H.$ is a fact. A rule without head: $\leftarrow a_1, \ldots, a_n, \neg b_1 \ldots, \neg b_m.$ is a constraint. When conditions of constraints are evaluated as true its imply $\bot$. In this paradigm the search of solution consist 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 deduced from facts and rules of the program. To compute it, we run an ASP solver on the 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, the set of facts {*rain*, *stay*} is an answer set of the ASP program, but {*rain*, *go_out*, *wet*} is not, because it contains *wet* which is not consistent with the constraint $\leftarrow$ *wet*.

Many research work use ASP to represent knowledge and reasoning such as [2] or [8]. In the first one, the authors use it to represent agent knowledge about others agents knowledge. 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 [6] 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 [5], the author discuss about integration of ASP modules into agent reasoning. There is also works like [1] or [7] where they discuss about possible extensions of ASP paradigm.

3. 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 represents a specific knowledge. We can have a module which contains observations about surroundings, an other one to define what is a prey and a module dedicated to compute path. By combining this three modules an agent can compute all paths to surroundings preys. To design agent knowledge we respect a module typology to represent background knowledge, observations and meta-knowledge.

3.1 Typology

Following previous distinction of agent knowledge we define a typology to represent common theory C and observations memory M, respectively by *theory modules* and *observations modules*. To represent knowledge about module combinations we define *meta-knowledge modules*. Their is

two kind of such modules, ones who produce or use knowledge to make decision and ones who represent a unique module combination. In following figures we represent *observation modules* by dot rectangle and *theory* one by plain rectangle. To distinguish *meta-knowledge modules* which make decision from ones who represent a combination, we represent it respectively by plain and dot circles. Module represented by dot form only contains facts and plain one can contain rules and constraint.

Definition 2 (Theory module). *A theory module is a set of rules which represent knowledge about a specific domain. The set of all theory modules of an agent represent his background knowledge. The purpose of these modules is to organise knowledge representation and produce new knowledge by combining it with others modules.*

Example 2. *A theory module of a rabbit about movement and one about actions where A is an agent, P a position and S a number of time step:*

Move

$reachable(A,P,\frac{D}{N}) \leftarrow distance(A,P,D),\ nb\_action(A,N).$
$reachable(A,P) \leftarrow reachable(A,P,0).$

Action

$nb\_action(me,3).$
$nb\_action(wolf,4).$
$0\{move(me,P)\}1 \leftarrow reachable(me,P).$
$\leftarrow move(A,P1),\ move(A,P2),\ P1 != P2.$
$\leftarrow move(me,P),\ reachable(wolf,P).$

Example 2 shows two theory modules, the first one represent knowledge about movement possibilities and the second one represent knowledge about actions. To simplify the first module of this example, *distance/3* is supposed given and his third argument is the distance between an agent A and a position P. Predicate *nb_action/2* specify the number of actions that an agent can perform in one time step. The first module can be use by a rabbit to compute his movement possibility and the ones of his predators. The second module define the movement action, it specify that only one move can be perform and it cannot be in the movement range of a wolf. By combining this two modules with observations about wolf, a rabbit will produce all possible movement he can perform and which are not in the direct movement range of any wolf.

Definition 3 (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. The set of all observations modules of an agent represent his memory: current and past observations. The purpose of these modules is to organise observations to facilitate their use and update.*

Example 3. *An observations module of a rabbit about wolfs positions and one about himself :*

Wolf

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

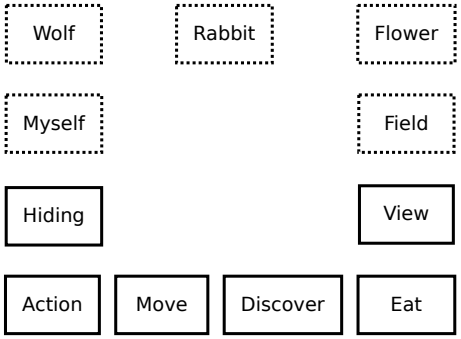

Figure 2: Knowledge base of a rabbit divided into observations modules (dot ones) and theory modules (plain ones).

Myself

$me(\text{"Bugs Bunny"}).$
$position(me,coord(0,0)).$
$type(me,rabbit).$

By this two kind of modules, we can represent agent common theory and observations. The idea now, is that we can combine these modules to produce different kind of knowledge. The figure 2 represent the knowledge of a rabbit by a set of observations modules and theory modules. In this example, by combining modules *Myself*, *Field* and *Move* a rabbit will produce all his possibility of movement. The combination {*Rabbit*, *Field*, *Move*} will produce all possible movements that others rabbits can perform. By combining {*Myself*, *Field*, *Flower*, *Move*, *Eat*, *Action*} a rabbit will produce all possible actions he can perform now and their influence on the number of step needed to eat each flower. 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 *Move* and *Eat*, an agent can use it to reason about its possibilities and the ones of others agents.

We have presented different way to use agent knowledge via module combination. But these combinations are also a kind of knowledge and it can be known by the agent. For an agent, it is meta-knowledge on his knowledge, more precisely in this case it is knowledge about the use of his knowledge. This meta-knowledge is a part of the common theory of an agent and could be represented by theory modules. But to clarify the design of agent knowledge we dedicated a new kind of modules to represent it: meta-knowledge module. We define two kind of these modules: combination module and decision module.

Definition 4 (Combination module). *A combination module is a meta-knowledge module which represent an ASP modules combination. The purpose of these modules is to represent simple behaviours and factorise meta-knowledge.*

A module combination can represent agent behaviour and by using meta-knowledge modules we can represent it in agent knowledge. Most simple behaviour can be represented by a simple list of modules to combine, like in examples 4, 5 and figures 3, 4. The module combination {*Wolf*, *Myself*, *hiding*, *Action*, *Move*, *Eat*, *View*, *Field*} and {*Myself*, *Action*, *Move*, *Eat*, *Field*, *Flower*, *Rabbit*} can respectively represent *hide* and *feed* behaviours of a rabbit.

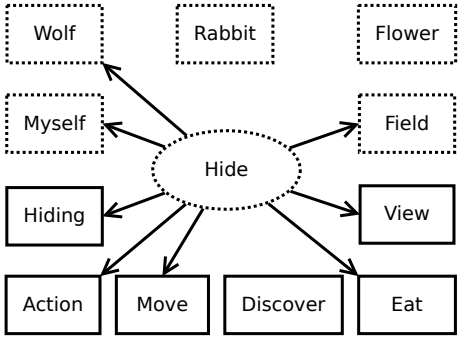

Figure 3: A meta-knowledge module (circle one) about *run away* behaviour linked to modules it combines.

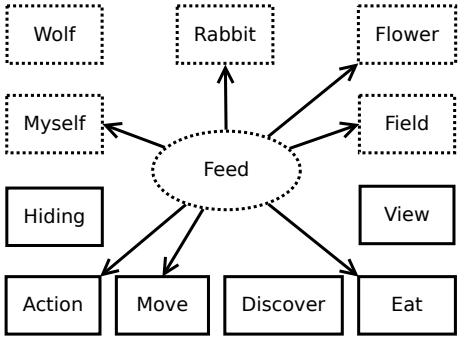

Figure 4: A meta-knowledge module about *feed* behaviour linked to modules it combines.

Example 4. *A meta-knowledge module which represent rabbit* hide *behaviour:*

Run away

***use_module**("Wolf").*
***use_module**("Myself").*
***use_module**("Hiding").*
***use_module**("Action").*
***use_module**("Move").*
***use_module**("Eat").*
***use_module**("View").*
***use_module**("Field").*

Example 5. *A meta-knowledge module which represent rabbit feed behaviour:*

Feed

***use_module**("Myself").*
***use_module**("Action").*
***use_module**("Move").*
***use_module**("Eat").*
***use_module**("Field").*
***use_module**("Flower").*
***use_module**("Rabbit").*

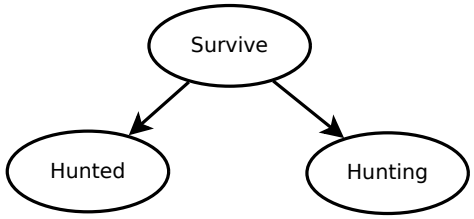

Figure 5: A meta-knowledge module with knowledge about meta-knowledge module.

Definition 5 (Decision module). *A decision module is meta-knowledge module which define the conditions to use ASP modules combinations. The purpose of these modules is to represent dynamic behaviours by performing decision on how to reason and use knowledge.*

To represent more complex behaviours, meta-knowledge modules need knowledge on other meta-knowledge modules. Decision modules contain such knowledge, it can be used to represent dynamic behaviours and perform meta-reasoning. Survive behaviour of a rabbit can be represent by such module, like in example 6 and figure 5. If a rabbit spotted a wolf it will use the *Hunted* module otherwise it will use the *Hunting* module. Survive behaviour is dynamic: it depends of the current state of the world.

Example 6. *A meta-knowledge module which describe the survive behaviour of a rabbit:*

Survive

predator ← position(wolf,Position).
***use_module**("Hunted") ← predator.*
***use_module**("Hunting") ← not predator.*

Multiple meta-knowledge modules can represent agent global behaviour. Figure 6 represent rabbit behaviour by a meta-knowledge modules tree. The behaviour of a rabbit is survive, regarding the situation a rabbit is hunted by a wolf or not. When hunted, a rabbit has to run away if wolfs are close or to search for hiding place if not. If there is no predator, he will explore his environment to find flowers and when one is spotted, find a way to eat it.

3.2 Framework

The previous section shows how ASP modules can represent agent common theory, observations and behaviours. To use this representation we define a reasoning architecture and a framework to use it in agent applications. As show in figure 7, we divide agent reasoning into three phase : acquisition, deduction, action.

Reasoning cycle start when agent sensors retrieve new observations, it's the beginning of the acquisition phase. During acquisition phase, agent memory is updated to make it consistent with new observations. New observations are stored in corresponding observations modules and old ones are modified. If a wolf see a rabbit at position P at time step T, he will store the information *position(rabbit,P)* in module *Rabbit*. Now at T+1 the wolf see position P but no rabbit are at P, then the wolf have to remove the fact *position(rabbit,P)* from module *Rabbit*.

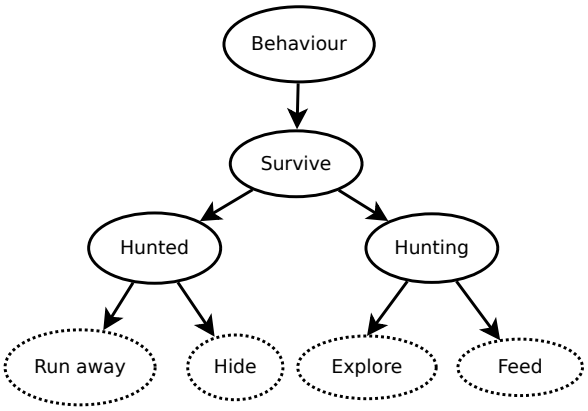

Figure 6: Representation of rabbit behaviour by a tree of meta-knowledge module. Decision module are represent by plain circle and combination modules by dot ones.

After acquired observations, an agent will reason on what he can do regarding the current state of the world: it is the deduction phase. The purpose of this reasoning phase is to deduce what actions are possible to perform. Here we use an agent reasoning architecture based on ASP modules where meta-knowledge modules are used for reasoning. Decision modules use agent observations to define his behaviour and generate actions step by step. Knowledge produced by module combination is keep in next reasoning steps, it means that answer set of the last reasoning step contains all deduction of previous ones. These last answer set contains actions and their consequences on the environment and the agent.

The figure 8 represent rabbit knowledge by an ASP modules graph. A node represent an ASP module and an edge represent the use of a module. Edges with plain arrows represent reasoning decisions and open ones represent module combination. In this representation, decision modules represent reasoning step. Here, meta-knowledge modules represent agent behaviour like in figure 6. In the example of figure 8, the deduction phase will start by using *Behaviour* module which specify to combine *Survive* module with wolfs observations. By combining these modules the rabbit will decide if he is hunted or have to hunt. When he is hunted, he will use *Hunted* module to decide if he has to run away or hide. Let's suppose that a wolf is too close to hide, then he will choose to run away, otherwise he will reason about where he can hide. In the case of there is no wolf, the rabbit will use *Hunting* module with flowers observations. If there is no flower he will use *Explore* module and when flowers are spotted he will use *Feed* module.

Module combination specify by *Run away* module generate all possible movement and the number of step needed by each wolfs to eat the rabbit after his move. The one of *Hide* module add the theory module *Hiding* and *View* to this combination, to reason about where to hide regarding wolfs sight range. *View* is also used with *Discover* in *Explore* module combination. Here, it is the knowledge about rabbit range of view which is use to reason about environment exploration. Finally, the module combination of *Feed* module generate eating possibilities. It include observations

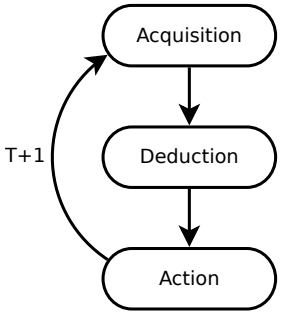

Figure 7: Agent reasoning cycle.

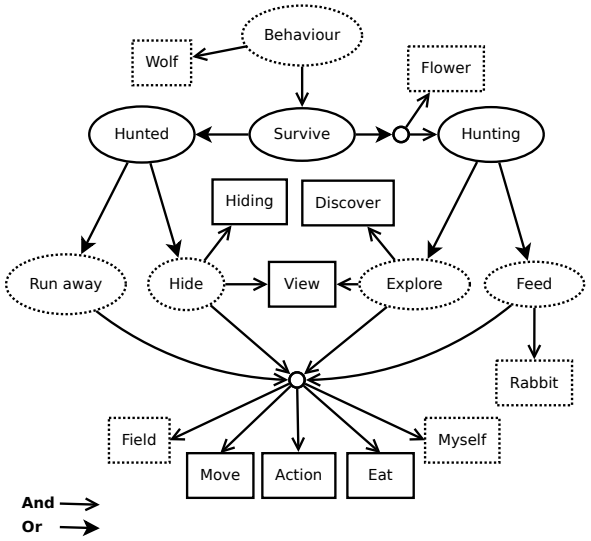

Figure 8: Representation of rabbit knowledge by an ASP modules graph.

about other rabbits to compute if a flowers can be eaten by another rabbit before the agent.

Now the agent have to choose which actions to perform, it means choose an answer set which allows him to approach or reach his goals. To make this choice we can use constraint and use some modules to compute score to rank answer set. Here, module *Eat* compute the number of action a predator have to perform to eat a prey. This module can be use by a rabbit to rank movement action to go far away from a wolf or to approach and eat a flower. After choosing the best answer set, corresponding actions are performed and the cycle continue by the acquisition of the real effect of theses actions on the environment.

3.3 Implementation

To allow an agent to use our method for reasoning in dynamic environment we implemented this framework. Algorithm 1 represent a simplified version of this tool.

Input is a set of modules and output is a set of answer set which contains actions that an agent can perform on its environment. It start by computing the combination of input modules and retrieve its answer sets by using the solver *clingo*. Answer set are exploit in a deep first exploration by extracting keyword which define the combination of modules

to use in the next reasoning step. When the answer set is totally parsed, we convert the current answer set into an ASP module and add it to the next combination of modules. Then, this combination is use for the next reasoning step and cycle continue. Finally, when there is no more module to combine it return a set of answer set.

Let's take again the example of the figure 8 and let's suppose that the rabbit 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 he is hunted or have to hunt. Let's suppose that observations contains a wolf position, then the next combination of module to use is {*Hunted*}. The wolf is too near to try to hide, so the next combination of module to use is {*Run away*} which just specify to use {*Movement*}. Finally this module will specify that the last combination of module to use is {*Myself*, *Field*, *Action*, *Move*, *Eat*}. Then the algorithm will return all answer set of this combination, each answer will contain an movement action and its consequences, for example the number of movement the wolf have to perform to catch the rabbit.

In this example we can see that only a part of the knowledge is use for reasoning: meta-knowledge modules *Hide*, *Hunting*, *Explore* and *Feed* are not used. Observations about flowers and knowledge about view 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 8. To represent behaviours it is quite intuitive to make a tree of meta-knowledge modules, where the root is the most abstract behaviour and deeper one are more specific.

Algorithm 1 Combine

```
1: INPUT : <M> M a set of ASP modules
2: OUTPUT : AS a set of answer set

3: AS a set of answer set

4: AS ← clingo(M)

5: for each answer set S of AS do
6:    M ← ∅

7:    // Extract keywords
8:    for every literal L of S do
9:       if L = "use_module(Module)" then
10:         M ← M ∪ Module
11:         S ← S / L
12:      end if
13:   end for

14:   if M ≠ ∅ then
15:      // Add S to M as a module
16:      M ← M ∪ module(S)
17:      combine(M)
18:   end if
19: end for

20: return AS
```

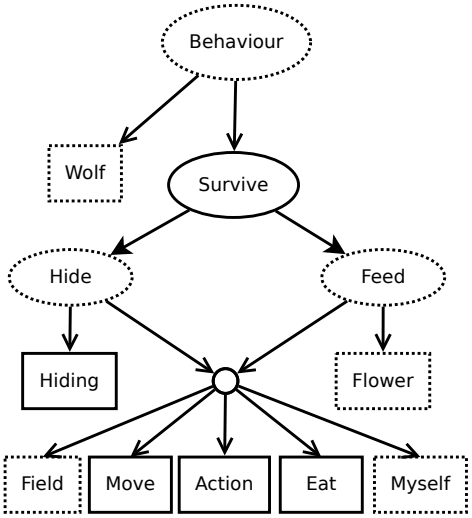

Figure 9: Rabbit knowledge in experiments, simplified to behaviours *hide* and *feed*.

4. EXPERIMENTS

To evaluate our work we implemented the algorithm 1 and use it in a toy application based on the survival game example. In this application, the environment is a grid where agent are located on a square. Agents act turn by turn and have limited number of actions per turn. Theses actions can be: move to square or eat an agent. To eat his prey, a predator have to be on the same square. These experimental results focus on the reasoning time of a rabbit regarding a specific scenario. We compare modular and monolithic reasoning time: the first one needs multiple step to reach the module combination which correspond to the situation and the second one use directly the entire knowledge base (the background knowledge and all observations). In these experiments, the module combination {*Field*, *Move*, *Action*, *Eat*, *Myself*} is always use at the end of reasoning. Difference between modular and monolithic representation is that the first one can avoid the use of flowers observation or hiding possibility.

4.1 Context

In these experiments, the rabbit use the algorithm 1 for reasoning. His observations concern the entire map, he knows what is on each square, it include the position of wolfs, flowers and hiding place. The figure 9 represent the modular knowledge of the rabbit in these experiments. It is a simplified version where rabbit have two behaviours: *run away* and *feed*. Monolithic representation always use the entire knowledge base which correspond to the module combination {*Myself*, *Wolf*, *Flower*, *Field*, *Move*, *Action Eat*, *Hiding*}.

For these experiment we use four different environment represented by figure 10, 11 and 12. The fourth one is equivalent to third one but ten time bigger in number of squares, flowers and hiding place. Our evaluation is based on height scenarios: for each environments, we evaluate rabbit reasoning time regarding if a a wolf is present or not. In scenarios where there is a wolf, reasoning about flower is useless because hiding have a bigger priority. When there is no

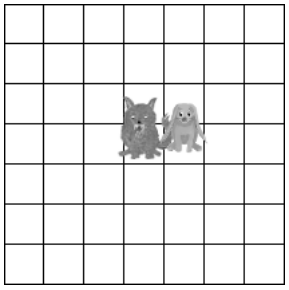

Figure 10: Environment 1, a rabbit and a wolf without flowers and no place to hide.

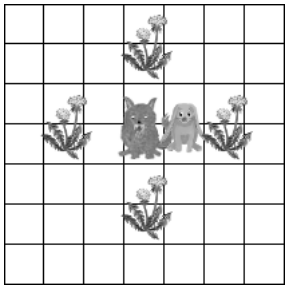

Figure 11: Environment 2, a rabbit, a wolf, 4 flowers and 4 hiding places (grass).

wolf, reasoning about hiding is a loose of time because eating flower is more important. Here, we show that modular reasoning can be useful to optimize reasoning by avoiding a part of agent knowledge which is not necessary in the current situation. In all scenario, monolithic reasoning consider movement to hiding and eating possibilities regardless the presence of a wolf. Modular reasoning consider movement to hide when there is a wolf and consider eating when there is no predator.

The figure 10 represent the first scenario where there is no flower and no hiding place. In this scenario the rabbit just consider the movement he can perform without being catch by his predator at the next turn. This scenario is used to evaluate the time we loose when modularity reasoning does not reduce search space. The figure 11 represent the second scenario where there is 4 flowers and 4 hiding places. In this scenario, when there is no wolf the time loose by multiple step of reasoning is just compensate by the reduction of search space. The figure 12 represent the third scenario, here our agents are in a field 25 flowers and 25 hiding place which are on the same square as flowers. Fourth scenario is a ten times bigger extension of the third one where the rabbit is in a field of 250 flowers and hiding place. In these two last scenario, the number of flowers and hiding place is sufficient for modular representation to reduce reasoning time.

4.2 Results

These experiments focus on rabbit reasoning time regarding modular and monolithic representation on 8 scenarios. This time correspond of the run of algorithm 1, which include clingo solving and answer set parsing. The time we use is user one, to have a relevant value we compute the average running time on 1000 runs for each methods on each

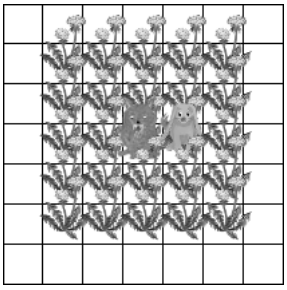

Figure 12: Environment 3, a rabbit and a wolf in a field of 25 flowers and 25 hiding places (same position as flower).

scenarios. These 16 sequences of 1000 runs test have been perform consequently in the same running conditions on a Intel Core 2 Duo P8400 2.26GHz CPU. Table **??** resume these experimental results.

In scenario 1, figure 10, when there is a wolf, monolithic reasoning run is about 0.135 and modular one is about 0.139. When there is no wolf monolithic reasoning run is about 0.107 and modular one is about 0.119. Here use of multiple reasoning step by modular representation cause more time for reasoning because it does not reduce search space. In scenario 2, figure 11, in the case of the presence of a wolf, run time of monolithic reasoning up to 0.181 because it now consider 4 flowers and 4 hiding places, and modular reasoning time just up to 0.169 because it does not consider flowers. If there is no wolf, monolithic and modular reasoning time is almost the same. The time loose by reasoning about hiding in monolithic representation is the same as the time loose by multiple step reasoning. In these environment modular reasoning is a little bit more efficient than monolithic one. For scenario 3, figure 12, modular reasoning is more efficient in both case. The proportion of flowers and hiding place is sufficient to make search space reduction interesting. When reasoning about hiding is the priority, modular reasoning take 30% less time than monolithic one and 18% less time when reasoning about eating is the most important. As the number observation increase, modular representation become more interesting, in scenario 4 when there is a wolf it take 49% less time to compute hiding actions and 32% less time for reasoning about rating.

These experiment show that modular knowledge representation and knowledge about module combination can be useful to reduce reasoning search space. Here we show that this representation can be always more efficient than monolithic one when the entire knowledge base is not necessary to solve every situation. Designing such reasoning pattern imply to find the balance between the time loose by multiple reasoning step and search space reduction.

5. CONCLUSIONS AND OUTLOOK

We provide a method based on ASP modules to design agent knowledge and reasoning. This representation allows to intuitively implements dynamic behaviours via meta-reasoning. We also provide a framework which allows agent reasoning by module combination. Regarding an equivalent monolithic representation, 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 pur-

Environment	Flower	Hiding place	Wolf	Representation	Time average	T-test	Runs
1	0	0	yes	monolithic	0.135 s	0	1000
				modular	0.149 s		
			no	monolithic	0.107 s	0	
				modular	0.119 s		
2	4	4	yes	monolithic	0.181 s	0	
				modular	**0.169 s**		
			no	monolithic	0.151 s	$2.115E-5$	
				modular	0.152 s		
3	25	25	yes	monolithic	0.416 s	0	
				modular	**0.277 s**		
			no	monolithic	0.383 s	0	
				modular	**0.314 s**		
4	250	250	yes	monolithic	3.838 s	0	
				modular	**1.963 s**		
			no	monolithic	3.819 s	0	
				modular	**2.739 s**		

Table 1: Experimental results of rabbit reasoning time on 8 scenarios of 4 different environments. For each method it shows reasoning time average of 1000 runs and T-test result.

poses.

In this paper, agent reasoning is very directed by meta-knowledge modules, an interesting outlook will be to give the possibility to the agent to really choose which module combination he want to use. Make agent able to construct themselves a reasoning architecture like the one of figure 8 is also an interesting research topic. Learning can also concern module content, an agent could choose to create new observations modules for storing specific information. Modular representation give new perspective for meta-reasoning and learning.

6. 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, 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.

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

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

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

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

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

[8] 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.

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