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, our interest is about multi-agent systems in dynamic environment. Here, we focus on representation of agent knowledge and reasoning. For 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, an agent needs to adapt his believes 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 such environment. 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 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 this work we consider an agent as an entity which is able to reason and communicate with his fellows. We divide the knowledge of an agent in two part:

- K a set of not revisable knowledge composed of :
  - T the common theory
  - O the current 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 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 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.

In recent years, many works focus on specific aspect of multi-agents systems such as communications, reasoning, learning or planning. 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.

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.

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 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 not be eat. 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 can 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 four behaviours: exploration, feed, hide and run away. In this paper our examples will focus on rabbit agent for knowledge representation and reasoning.

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.

Answer set programming which is a specification and an implementation language is very suitable to represent agent knowledge and reasoning. The methods introduce by this work is based 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. It allows agent to perform meta-reasoning: they use knowledge on modules combinations to adapt their reasoning and so on their behaviour to the situation.

2. ANSWER SET PROGRAMMING

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.

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 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 represent a 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. 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

Definition 2 (Background module). *A background module is a set of rules which represent knowledge about a specific domain. The set of all background 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 show two background modules, the first one represent knowledge about agent possibility of movement

and the second one represent knowledge about movement as an action. To simplify the first module of this example, *distance/3* is supposed given by an other module and his third argument is the distance between an agent A and a position P. Predicate *action/2* specify the number of action 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 predator. The second module define the movement action, it specify that only one move can be perform and it should not be in the movement range of a wolf. By combining this two modules with observations about wolf a rabbit will produce all possible movement which are not in the direct movement range of any wolf.

EXAMPLE 2. *A background module of a rabbit about movement where A is an agent, P a position and S a number of step:*

reachable(A,P,S) ← distance(A,P,S).
reachable(A,P) ← reachable(A,P,S), action(A,S).
A background module about actions:

action(me,3).
action(wolf,4).
0{move(me,P)}1 ← reachable(me,P).
← move(A,P1), move(A,P2), P1 != P2.
← move(me,P), reachable(wolf,P).

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 :*

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).

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 rabbit by a set of observations modules and background 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

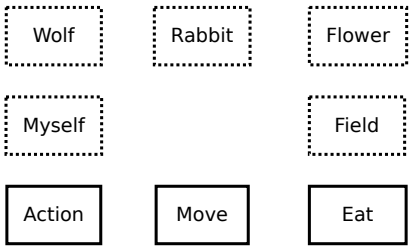

Figure 2: Knowledge base of a rabbit divided into observations modules (dot ones) and background modules (plain ones).

the example of modules *Move* and *Eat*, an agent can use it to reason about its possibilities and the ones of other agent.

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 the 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 background theory of an agent and could be represented by background modules. But to clarify the design of agent knowledge we dedicated a new kind of modules to represent it.

DEFINITION 4 (META-KNOWLEDGE MODULE). *A meta-knowledge module is a set of rules 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.*

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 examples 4, 5 and figures 3, 4. The module combination {*Myself*, *Wolf*, *Field*, *Move*, *Eat*, *Action*} and {*Myself*, *Rabbit*, *Flower*, *Field*, *Move*, *Eat*, *Action*} can respectively represent the *run away* and the *feed* behaviour of a rabbit. This two combinations of modules can be represented by meta-knowledge modules, in this case theses modules have knowledge on background and observations modules.

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

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

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

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

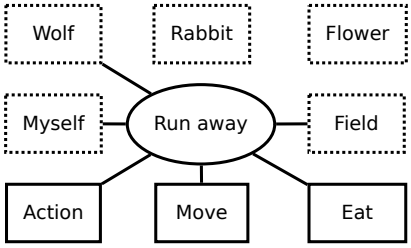

Figure 3: A meta-knowledge module (circle one) about *run away* behaviour linked to modules it combines.

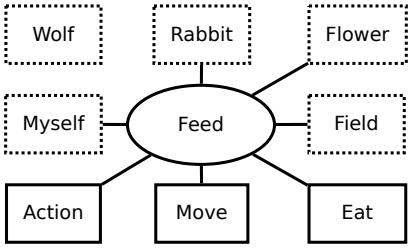

Figure 4: A meta-knowledge module about *feed* behaviour linked to modules it combines.

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 survive behaviour of a rabbit with such module, like in example 6 and figure 5. If a rabbit spotted a wolf it will use the *Run away* module otherwise it will use the *Feed* module. Regarding the situation the survive behaviour will be run way or approach flower. Meta-knowledge modules allows to represent such dynamic behaviours which are dependent of the evolution of the environment. Figure 6 represent rabbit behaviour by a meta-knowledge module tree. The behaviour of a rabbit is survive, regarding the situation a rabbit is hunted by a wolf or not. When hunted, a rabbit have to run away or try to hide by go out of view range of his predator. If there is no predator he will explore his environment to find flower and when one is spotted go to eat it.

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

predator ← *position(wolf,Position).*
use_module*("Run away")* ← *predator.*
use_module*("Feed")* ← *prey, not predator.*

3.2 Architecture

The previous section shows that ASP modules can represent background knowledge, observations and behaviours of an agent. To use this representation we define a reasoning architecture and a framework to use it in agent application. 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. This phase deals with new observations and their consequences

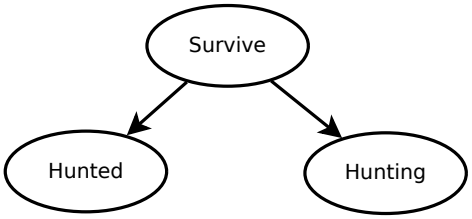

Figure 5: A meta-knowledge module with knowledge about meta-knowledge module.

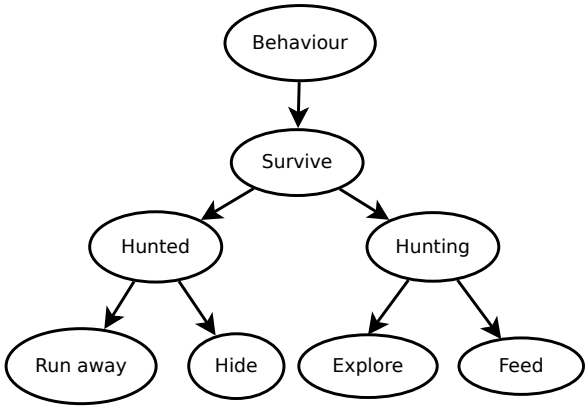

Figure 6: Representation of rabbit behaviour by a tree of meta-knowledge module.

on old ones. New observations are stored in corresponding observations modules and old ones are update. 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*. During acquisition phase, agent memory is update to make it consistent with new observations.

After acquire 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 use for meta-reasoning. Meta-knowledge modules use agent observations to define his behaviour step by step. Knowledge produce 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 action and their consequences on the environment and the agent.

The figure 8 represent rabbit knowledge by an ASP modules graph where a node is an ASP module, undirected edge represent the use of a module and directed edge represent reasoning decisions. Here meta-knowledge module *Movement* is used to represent a module combination which produce movement actions of a rabbit, other ones represent his behaviour like in figure 6. The reasoning phase will start by using *Behaviour* module which just specify to use *Survive* one. By combining *Survive* module and wolf observation the rabbit will decide if he is hunted or have to hunt. If he is hunted he will then combine *Hunted* with wolf observa-

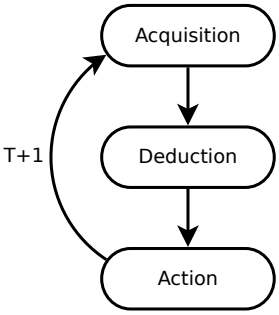

Figure 7: Agent reasoning cycle.

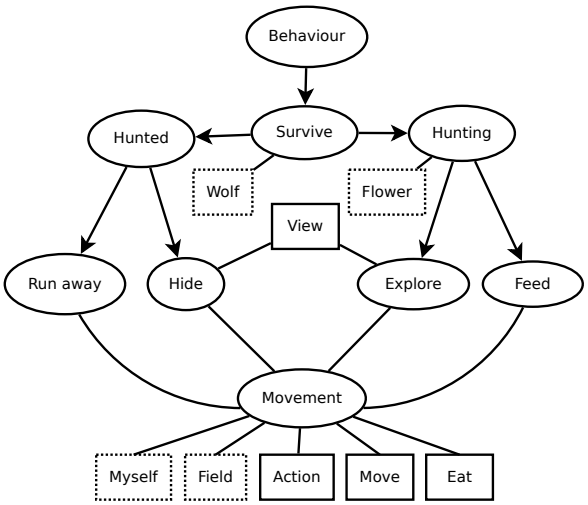

Figure 8: Representation of rabbit knowledge by an ASP modules graph.

tions to decide if run away or hide is the best solution, for example if the wolf is very near it is very risky to hide then run away will be choose. If there is no wolf the rabbit will use *Hunting* with flower observation, if there is no flower he will use *Explore* module and *View* to try to find something to eat (*View* module should contains some knowledge about view range). If there is flower in his observations, a rabbit will use *Feed* to produce actions will allows him to approach and eat flowers.

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 or use some modules to compute score to rank answer set. For example module *Eat* can compute the number of action a predator have to perform to eat a prey. Then 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.

4. 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.

5. 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. Here 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 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*. Every scenarios concern a rabbit which has spotted a wolf and which reason about how to run away. The specificity of *run away* behaviour is that it ignore observations about flowers.

Reasoning about flower is useless in these scenarios and can take a lot of time regarding the number of flower to consider. Here, flowers are used to illustrate unnecessary

Algorithm 1 Combine

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

AS a set of answer set

AS ← clingo(M)

for each answer set S of AS do
    M ← ∅

    // Extract keywords
    for every literal L of S do
        if L = "use_module(Module)" then
            M ← M ∪ Module
            S ← S / L
        end if
    end for

    if M ≠ ∅ then
        // Add S to M as a module
        M ← M ∪ module(S)
        combine(M)
    end if
end for

return AS
```

informations and how modular reasoning can avoid this kind of knowledge.

The figure 10 represent the first scenario, here there is no flower. In this scenario the rabbit just consider the movement he can perform regarding the distance between him and his predator. This scenario is used to evaluate the time we loose when modularity reasoning is not useful.

The figure 11represent the second scenario, here there is 4 flowers. Here, monolithic reasoning consider movement regarding the wolf and these flowers. For each movement it compute the number of action the rabbit will need to eat each flowers. Modular reasoning perform the same reasoning as in the first scenario because it ignore flowers observations.

The figure 12 represent the third scenario, here our agent are in a field 25 flowers. Like in the second scenario, the monolithic reasoning lose time to reason about these flowers but the modular one does not.

6. 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 purposes.

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

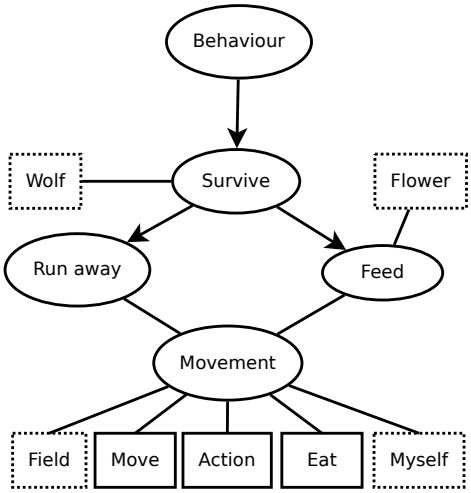

Figure 9: Representation of rabbit knowledge by an ASP modules graph.

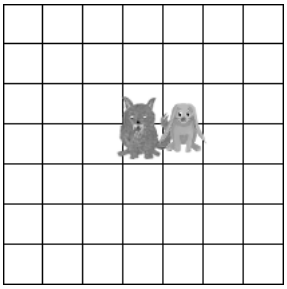

Figure 10: Scenario 1, a rabbit and a wolf.

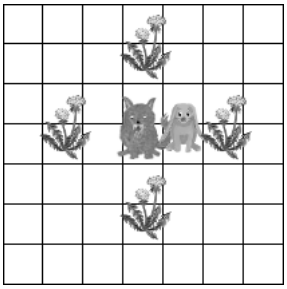

Figure 11: Scenario 2, a rabbit, a wolf and 4 flowers.

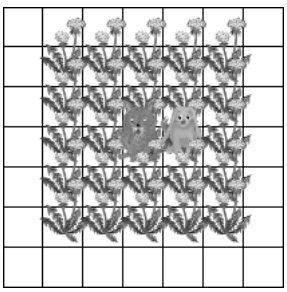

Figure 12: Scenario 3, a rabbit and a wolf in a field of 25 flowers.

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.

7. 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.