

# ANSWER SET PROGRAMMING TO MODEL PLAN AGENT SCENARIOS

Fernando Zacarias Flores<sup>1</sup>, Rosalba Cuapa Canto<sup>2</sup> and José María Ángeles López<sup>1</sup>

<sup>1</sup>Department of Computer Science, Autonomous University of Puebla, México

<sup>2</sup>Faculty of Architecture, Autonomous University of Puebla, México

## **ABSTRACT**

*One of the most challenging aspects of reasoning, planning, and acting in an agent domain is reasoning about what an agent knows about their environment to consider when planning and acting. There are various proposals that have addressed this problem using modal, epistemic and other logics. In this paper we explore how to take advantage of the properties of Answer Set Programming for this purpose. The Answer Set Programming's property of non-monotonicity allow us to express causality in an elegant fashion. We begin our discussion by showing how Answer Set Programming can be used to model the frog's problem. We then illustrate how this problem can be represented and solved using these concepts. In addition, our proposal allows us to solve the generalization of this problem, that is, for any number of frogs.*

## **KEYWORDS**

*Agent, Logic, Answer Set Programming, Planning, Reasoning*

## **1. INTRODUCTION**

When we want to design an intelligent agent capable of behaving intelligently in some environment, then we need to supply this agent with sufficient knowledge about this environment. The computational logic provides solutions, at a sufficient level of abstraction afforded by the nature of its very foundation in logic. On the other hand, logic programming has evolved, which has allowed the development of alternative proposals for the creation of logical rational agents, such as Answer Set Programming. Thus, we need a modelling language that provides a well defined, general, and rigorous framework for expressing this knowledge, together with some precise and well understood way of manipulating sets of sentences of the language which will allow us to draw inferences, answer queries, and update both the knowledge base and the desired program behaviour. Therefore, we will support our proposal on the stable model semantics currently used in logic programming and known as answer set programming. The intuitive meaning of stable sets can be described in the same way as the intuition behind stable expansions in other logics (as: auto-epistemic logic): they are possible sets of beliefs that a rational agent might hold given P (knowledge gained from its context) as its premises.

All of the above, Answer set programming (ASP) is this language. Answer Set Programming (ASP, Answer Set (Stable Model) Semantics [1] [2]), is the realization of much theoretical work on Non-monotonic Reasoning and Artificial Intelligence applications of Logic Programming. ASP is a novel paradigm of logic programming that has now great acceptance in the artificial intelligence community. One of the most important reasons for its acceptance is the existence of efficient software to compute answer sets [2]. ASP provides a simple, expressive and efficient language that can be well suited for modelling the agent rational component into computer

science and artificial intelligence. The answer sets for a logic program can be described as the satisfying interpretations for a set of propositional formulae.

The content of our paper is the following: Section 2 provides the theoretical bases on the paradigm used for knowledge representation. We also provide the formulations for the agent interpretation about logic programming. In section 3 and 4, we discuss the frog's problem from the perspective of the planning agent. Also, we describe in detail the representation of the agent's beliefs and objectives, as well as the actions and effects of these reflected in his environment, through the  $Dlv^K$  action language. Next, the discussion on generalization of this problem and future work are given in section 5. Finally, we provide conclusions.

## **2. ANSWER SET PROGRAMMING TO KNOWLEDGE REPRESENTATION**

ASP has originated notable changes regarding as designing intelligent systems based on agents. The notion of intelligent agents has begun to provide a more unified and more coherent approach to problems in the field of Artificial Intelligence. Artificial Intelligence (AI) has developed logic beyond the confines of monotonic cumulatively, typical of the precise, complete, enduring, condensed, and closed mathematical domains, in order to open it up to the nonmonotonic real world domain of the imprecise, incomplete, contradictory, arguable, revisable, distributed, and evolving knowledge.

AI has added dynamics to former static knowledge representation. In AI, alike any other science, common logic has an important role in it. AI has helped researchers to explore new reasoning issues and methods, and to combine disparate reasoning modalities into a uniform unified framework to deal with incomplete, imprecise, contradictory, and changing information.

### **2.1. Different Proposals based on Answer Set Programming**

One line of research used by many researchers is the use of logic as a language of knowledge representation combined with automated deduction and constructive logic. This combination is currently reflected in a new paradigm called "answer set programming". ASP is the result of having a declarative semantics for logic programs with negation as failure. Furthermore, the proposal of this paradigm brought with it the formalization of non-monotonous reasoning and the use of negation. These languages have a well-defined semantics, independent of a particular inference mechanism. The mathematical formalisms related to ASP can be found in [8], [9]. An in-depth coverage of many aspects of knowledge representation and reasoning with ASP can be found in [7]. Finally, several logic-based works in artificial intelligence are collected in [10].

ASP is a logic programming language based on answer sets / stable model semantics [1]. ASP provides a simple, expressive and efficient language that can be well suited for modelling the agent rational component into computer science and artificial intelligence. The answer sets for a logic program can be described as the satisfying interpretations for a set of propositional formulae.

A different and interesting proposal for the modelling of agents, reasoning and knowledge representation is called "Ans-prolog" [13]. This proposal is based on answer set programming. This proposal uses SAT solver for to find models. Another proposal for the representation of knowledge, reasoning and modelling of agents is smodels [14]. However, the use of smodels may be less clear and simple than  $Dlv^K$ . There are some other proposals for the representation of knowledge and reasoning based on temporal logic. However, these proposals do not have a clear syntax, and the search for plans is not as efficient as in  $Dlv^K$ . Finally, clasp [15] is a proposal similar to  $Dlv^K$  in relation to its efficiency, that is, the calculation of models is as efficient as in

Div<sup>K</sup>. Both have even won in several international competitions. Potassco is a bundles tools for ASP developed at the University of Potsdam. With Potassco tools you can concentrate on an actual problem, rather than a smart way of implementing [11].

## 2.2. Propositional Logic

We use the language of propositional logic in order to describe rules within logic programs. Formally we consider a language built from an alphabet consisting of atoms:  $p_0, p_1, \dots$ ; connectives:  $\wedge, \vee, \leftarrow, \perp$ ; and auxiliary symbols:  $(, ), ', \cdot$ , where  $\wedge, \vee, \leftarrow$  are 2-place connectives and  $\perp$  is a 0-place connective. Formulas are defined as usual. The formula  $\top$  is introduced as an abbreviation of  $\perp \leftarrow \perp$ , not  $F$  as an abbreviation of  $\perp \leftarrow F$ , and  $F \leftrightarrow G$  as an abbreviation of  $(G \leftarrow F) \wedge (F \leftarrow G)$ . The formula  $F \rightarrow G$  is another way of writing the formula  $G \leftarrow F$ , we use the second form because of tradition in the context of logic programming. We will represent the default negation with  $\neg$ . A signature  $L$  is a finite set of atoms. If  $F$  is a formula then the signature of  $F$ , denoted as  $LF$ , is the set of atoms that occur in  $F$ . A literal is either an atom  $a$  (a positive literal) or a negated atom  $\neg a$  (a negative literal). A logic program is a finite set of formulas. The syntax of formulas within logic programs has been usually restricted to clauses with a very simple structure. A clause is, in general, a formula of the form  $H \leftarrow B$  where  $H$  and  $B$  are known as the head and body of the clause respectively. Two particular cases of clauses are facts, of the form  $H \leftarrow \top$ , and constraints,  $\perp \leftarrow B$ . Facts and constraints are sometimes written as  $H$  and  $\leftarrow B$  respectively.

## 3. THE FROGS PROBLEM USING PLANNING AGENT

In the last years, agent's paradigm using logic programming has claimed a major role in defining the trends of modern research, influencing a broad spectrum of disciplines such as Computational Logic, Philosophy, Logic Programming, Answer Set Programming, among others. Agent paradigm has invaded every subfield of Computer Science [3], [1], [4]. The agent concept has recently increased its influence in the research and development of computational logic-based systems. Also, Logic Programming (LP) allows an adequate representation for knowledge. Currently, we have noticed significant improvements in the efficiency of logic programming implementations for nonmonotonic reasoning [5], [6], [7]. These implementations allow us a unified declarative and procedural semantic, eliminating the traditional wide gap between theory and practice. Besides, some extensions have been given to Logic Programming in the field of non-monotony.

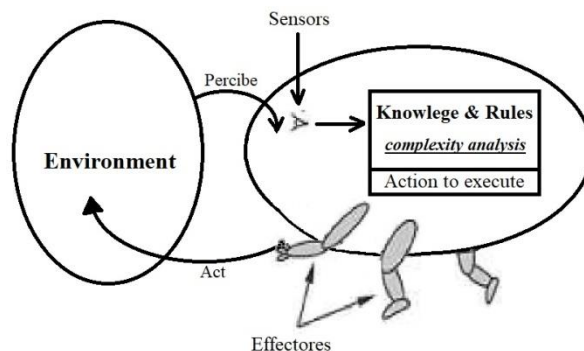


Figure 1. General agent architecture

In figure 1, we can see the general architecture of our agent. The agent has a knowledge base conformed by the following rules: First, the left set of counters can only move right, the right set of counters can only move left. Second, counters can move forward one space, or move two

spaces by jumping over another counter. Third, the puzzle is solved when the two sets of counters have switched positions. Furthermore, the agent performs an analysis on the various plans found to determine the best of them.

### 3.1. The Frogs Problem

The frog problem assumes a configuration like the following: the scenario consists of seven stones in a line, in these there are six frogs placed one on each of the stones, leaving an empty stone in the middle. The three frogs on the left will be the blue frogs' team, and the three frogs on the right will be the green frogs' team (figure 2).

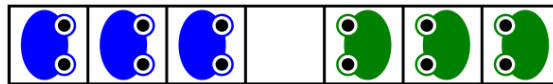


Figure 2. The Frogs problem

### 3.2. The Frogs Problem Rules

The blue team can only move to the right, and the green team can only move to the left. On any move, a frog can either slide into the empty stone, or a jump over one frog from the other team into the empty stone. The goal is to have the two teams swap positions, where the green team ends up on the left and the blue team ends up on the right.

At this point, it is important to ask ourselves the following questions: Can the blue frogs end up in the green frogs' original positions? Can the green frogs end up in the blue frogs' original positions? If so, what is the smallest number of moves that it takes? If not, why not? Generalize, extend and justify everything.

## 4. THE FROGS PROBLEM USING DLV<sup>K</sup>

Three frogs in each side is just enough to make the problem non-trivial. To make it really interesting, you might jump straight into eight frogs in each side but let's hold that for a moment. It doesn't take so long for the rules to be absorbed and to learn that the best way to solve the problem is avoiding to get two frogs of the same color next to each other, unless you're at the final stages. So, it doesn't take long time before the following moves are made, and the frogs swapped (see figure 2).

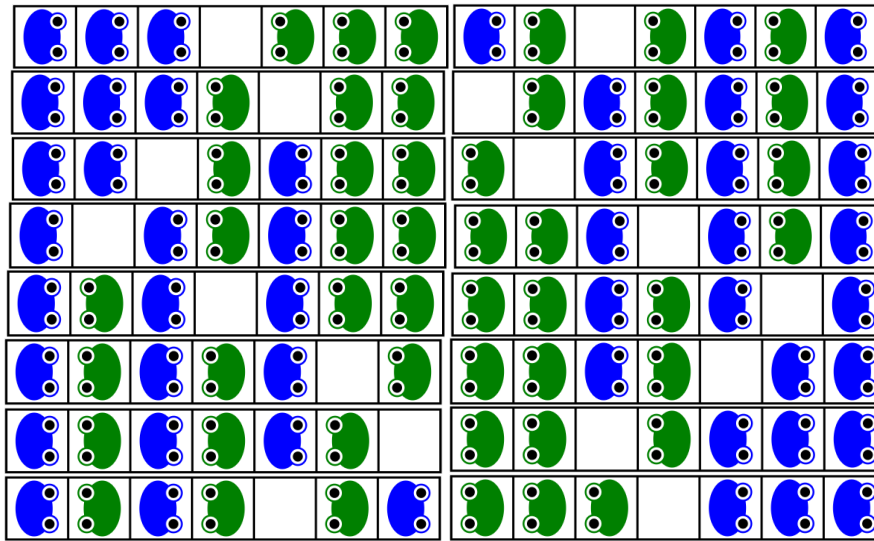


Figure 3. You did it! Well done! (moves 15)

The figure 3 answers both parts of question. On the other hand, a question arises: Is possible to solve this same problem with 4, 5, 6 or  $n$  frogs of each color? If it's not possible, why it can't support more frogs? This question will be addressed later.

#### 4.1. The Frogs Problem Rules

In this section, we describe how planning problems can be described as “planning programs” – PP in  $Dlv^K$  system.  $Dlv^K$  programs are built using statements of the language  $K$ , plus further optional control statements. The representation of this problem can be done using scheme presented in [12] or [16]. A planning problem is a pair  $P = \langle PD, q \rangle$  of a planning domain  $PD$  in a query  $q$ , which specifies the goal. A planning problem is represented as a combination of a background knowledge, which is represented by a planning agent, and a program of the following general form: **fluents: FD / actions: / AD always: CR / initially: IR / goal: q**

Where fluent represent basic properties of the world, which can change over time. They are similar to propositional assertions. States are collections of fluents, each of them is associated with a true-value. We distinguish between so called *world states* and *knowledge states*. The current state of the world with respect to a set of fluent is:  $F = \{f_1, \dots, f_n\}$ , can be defined as a function  $s: F \rightarrow \{\text{true}, \text{false}\}$ , that is, a set of literals which contains either  $f$  nor  $\neg f$  for any  $f \in F$ .  $s$  is a state of knowledge. From an agent's point of view, states can also be seen as partial functions  $s'$ , that is, consistent sets of fluent literals, where for a particular fluent  $f \in F$  neither  $f$  nor  $\neg f$  may hold.

**fluents:** on\_r(B,L) requires frog\_r(B), location(L).  
 on\_l(B,L) requires frog\_l(B), location(L).  
 occupied(B) requires location(B).

Fluents on\_r and on\_l are used to characterize that frogs are on a stone. In particular, on\_r describes that a frog from blue team is on a stone and that it will move to the right and, on\_l describes that a frog from green team is on a stone and that it will move to left.

On the other hand, the actions defined for this problem are:

**actions:** jump\_r(B,L) requires frog\_r(B),location(L),on\_r(B,T),difference(L,T,R),  $R < 3$ ,  $L > T$ .  
 jump\_l(B,L) requires frog\_l(B), location(L),on\_l(B,T),difference(T,L,R),  $R < 3$ ,  $L < T$ .

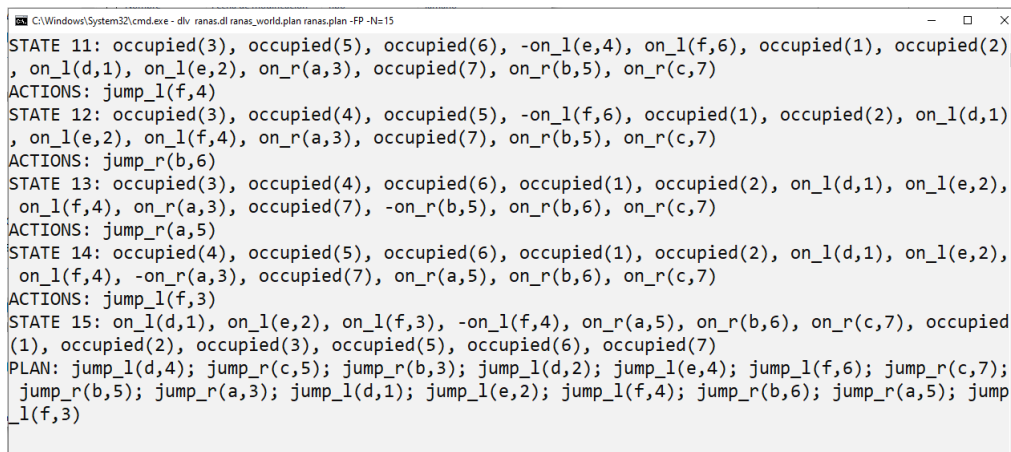
Actions jump\_r represents the jumping action that frogs can perform. In particular, jump\_r describes that a frog from blue team jumps to right and jump\_l describes that a frog from green team jumps to left.

#### 4.2. The Plan to Solve the Frogs Problem

Dlv<sup>K</sup> is a knowledge-based planning system supported by the declarative language *K*, which is similar in spirit to the logic-based language *C* [12]. Unlike the *C* language, *K* includes some logic programming features such as: default negation, strong negation, optimistic and secure planning, i.e., construction of a “credulous” plan or a “sceptical” plan, which works in all cases. Thus, we must define both configurations, the initial configuration and the goal. These settings are defined as follows:

initially: on\_r(a,1). on\_r(b,2). on\_r(c,3). on\_l(d,5). on\_l(e,6). on\_l(f,7).

This initial configuration describes that frogs a, b, and c move to the right and are on stones 1, 2 and 3 respectively. Also, frogs d, e and f move to the left and are on stones 5, 6 and 7 respectively.



```

C:\Windows\System32\cmd.exe - dlv ranas.dl ranas_world.plan ranas.plan -FP -N=15
STATE 11: occupied(3), occupied(5), occupied(6), -on_l(e,4), on_l(f,6), occupied(1), occupied(2) ^
, on_l(d,1), on_l(e,2), on_r(a,3), occupied(7), on_r(b,5), on_r(c,7)
ACTIONS: jump_l(f,4)
STATE 12: occupied(3), occupied(4), occupied(5), -on_l(f,6), occupied(1), occupied(2), on_l(d,1)
, on_l(e,2), on_l(f,4), on_r(a,3), occupied(7), on_r(b,5), on_r(c,7)
ACTIONS: jump_r(b,6)
STATE 13: occupied(3), occupied(4), occupied(6), occupied(1), occupied(2), on_l(d,1), on_l(e,2),
on_l(f,4), on_r(a,3), occupied(7), -on_r(b,5), on_r(b,6), on_r(c,7)
ACTIONS: jump_r(a,5)
STATE 14: occupied(4), occupied(5), occupied(6), occupied(1), occupied(2), on_l(d,1), on_l(e,2),
on_l(f,4), -on_r(a,3), occupied(7), on_r(a,5), on_r(b,6), on_r(c,7)
ACTIONS: jump_l(f,3)
STATE 15: on_l(d,1), on_l(e,2), on_l(f,3), -on_l(f,4), on_r(a,5), on_r(b,6), on_r(c,7), occupied
(1), occupied(2), occupied(3), occupied(5), occupied(6), occupied(7)
PLAN: jump_l(d,4); jump_r(c,5); jump_r(b,3); jump_l(d,2); jump_l(e,4); jump_l(f,6); jump_r(c,7);
jump_r(b,5); jump_r(a,3); jump_l(d,1); jump_l(e,2); jump_l(f,4); jump_r(b,6); jump_r(a,5); jump
_l(f,3)
    
```

Figure 4. The frog’s problem solution plan

Goal configuration is defined as follows:

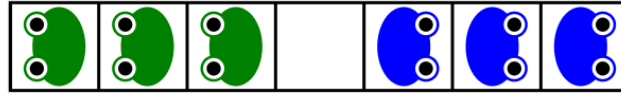
goal: on\_l(d,1), on\_l(e,2), on\_l(f,3), on\_r(a,5), on\_r(b,6), on\_r(c,7) ? (15)

Goal configuration describes the final state that frogs of both teams must reach after a sequence of actions, i.e.,  $a_1, a_2, \dots, a_n$ . Specifically, the frogs of the blue team (a, b and c) must be in the stones on the right (5, 6 and 7), in addition, the frogs of the green team (d, e and f) must be in the stones of the left (1, 2 and 3). In addition, ? (15) serves to indicate that we want to obtain a plan that is achieved in 15 steps.

As you can be seen in figure 4, obtaining the plan calculated by Dlv<sup>K</sup> that includes the transitions among the fifteen different states that show the changes reflected with each of the actions carried out to achieve the objective. Figure 4 also shows, for example, state 14, where frog f is on stone 4, that is, it has not yet reached its goal. Next, the final action to reach the objective is jump\_l (f,

3), with which state 15 is reached, that is, this state reflects that all the frogs have reached their objective, thus completing the plan.

### Jumping frogs puzzle



**You did it! Well done!**

Moves: 15

Figure 5. Solution to Jumping frog's puzzle

The plan obtained is the one shown below, in which you can see the 15 actions (jumps) necessary to complete the plan that solves the frog's problem. In figure 5, we can see the solution obtained by applying the plan generated by Dlv<sup>K</sup>. In addition to obtaining this plan, the Dlv<sup>K</sup> software can obtain more plans, if they exist, as shown in figure 6. In this case, the symmetric plan to the solution presented above, that is, the blue frogs initiating movements.

PLAN: jump\_l(d,4); jump\_r(c,5); jump\_r(b,3); jump\_l(d,2); jump\_l(e,4); jump\_l(f,6);  
 jump\_r(c,7); jump\_r(b,5); jump\_r(a,3); jump\_l(d,1); jump\_l(e,2); jump\_l(f,4);  
 jump\_r(b,6); jump\_r(a,5); jump\_l(f,3)

```

C:\Windows\System32\cmd.exe - dlv ranas.dl ranas_world.plan ranas.plan -FP -N=15
STATE 13: occupied(4), occupied(5), occupied(6), on_l(f,5), occupied(1), occupied(2)
, on_l(d,1), on_l(e,2), -on_l(e,3), occupied(7), on_r(a,4), on_r(b,6), on_r(c,7)
ACTIONS: jump_l(f,3)
STATE 14: occupied(3), occupied(4), occupied(6), -on_l(f,5), occupied(1), occupied(2)
), on_l(d,1), on_l(e,2), on_l(f,3), occupied(7), on_r(a,4), on_r(b,6), on_r(c,7)
ACTIONS: jump_r(a,5)
STATE 15: on_l(d,1), on_l(e,2), on_l(f,3), -on_r(a,4), on_r(a,5), on_r(b,6), on_r(c,
7), occupied(1), occupied(2), occupied(3), occupied(5), occupied(6), occupied(7)
PLAN: jump_r(c,4); jump_l(d,3); jump_l(e,5); jump_r(c,6); jump_r(b,4); jump_r(a,2);
jump_l(d,1); jump_l(e,3); jump_l(f,5); jump_r(c,7); jump_r(b,6); jump_r(a,4); jump_l
(e,2); jump_l(f,3); jump_r(a,5)

Check whether that plan is secure (y/n)? y
The plan is secure.

Search for other plans (y/n)?
    
```

Figure 6. Second solution found by Dlv<sup>K</sup>

## 5. DISCUSSION & FUTURE WORK

One of the most frequent questions that arise regarding this problem is the following: if it's possible to do the same thing with 4, 5, 6, or n frogs of each color? The answer to this question is yes, in the case of our proposal, it is enough to just give our proposal more frogs and stones. In the following table 1 we can summarize the number of movements required to solve the frog's problem for configurations with more frogs.

Table 1. Table of values for n frogs

n	1	2	3	4	5	6	7	...
N	3	8	15	24	35	48	63	...

In our proposal, the modification that must be made is only to increase the stones and frogs, as well as the initial and final configurations. In this way, the generalization of this problem is given naturally.

initially: on\_r(a,1). on\_r(b,2). on\_r(c,3). on\_r(d,4). on\_l(f,6). on\_l(g,7). on\_l(h,8). on\_r(i,9).

goal: on\_r(a,6), on\_r(b,7), on\_r(c,8), on\_r(d,9), on\_l(h,1), on\_l(g,2), on\_l(f,3), on\_l(e,4) ? (24)

This is one of the advantages of using languages based on logic, i.e., when modelling a problem based on logic programming. This happens when the proposals are based on general rules such as those that logic allows.

On the other hand, the definition of general rules such as those defined for the modelling of this problem serve as the basis for modelling other equally complex problems. As future work, we must extend the use of agents to problem modelling based on multi-agent systems.

## 6. CONCLUSIONS

The modelling of this problem using the action language  $Dlv^K$  has shown its expressive power, simple and with novel characteristics in terms of planning and reasoning about actions, allowing to encode even difficult planning problems with alternative preconditions of actions and effects of non-deterministic actions.  $Dlv^K$  has many advantages as a language for planning even under incomplete initial knowledge.

An experience resulting from this work is that the modelling of this problem solves the general problem, and also solves it for any number of frogs. Additionally, our proposal allows minor modifications to be able to solve different configurations.

## ACKNOWLEDGEMENTS

The authors would like to thank the support received by the academic body combinatorial algorithms and learning at Autonomous University of Puebla. The authors would like to thank everyone, just everyone! And that includes to Rene, my little frog for a day!

## REFERENCES

- [1] M. Gelfond and V. Lifschitz: The Stable Model Semantics for Logic Programming - Proceedings of the Fifth International Conference on Logic Programming (ICLP), pages 1070-1080, 1988.
- [2] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Christoph Koch, Cristinel Mateis, Simona Perri, Francesco Scarcello. "The DLV System for Knowledge Representation and Reasoning", October 2002.
- [3] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, and F. Zini. Logic programming and multi-agent system: A synergic combination for applications and semantics. In *The Logic Programming Paradigm – A 25 Year Perspective*. Springer, 1999.
- [4] F. Sadri and F. Toni. *Computational logic and multiagent systems: A roadmap*. <http://www.compulog.org>, 1999.
- [5] I. Niemelä and P. Simons. Smodels: An Implementation of the stable model and Well-founded and Semantics for normal lp. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th*



- International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97), pages 28–31, Berlin, Germany, July 1997. Springer Verlag.
- [6] S. Konstantinos, S. Terrance, and D.S. Warren. XSB as an Efficient Deductive Database Engine. In Proc. of the Thirteenth ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems (PODS'94), pages 442–453, Minneapolis, Minnesota, May 1994. ACM Press.
- [7] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv System: Model Generator and Application Front ends. In Proceedings of the 12th Workshop on Logic Programming (WLP '97), Research Report PMS-FB10, pages 128–137, M'unchen, Germany, September 1997. LMU M'unchen.
- [8] J.J. Alferes, L.M. Pereira, Reasoning with Logic Programming, Springer, Berlin, 1996.
- [9] C. Baral, M. Gelfond, Logic programming and knowledge representation, J. Logic Programming 19,20 (1994) 73–148.
- [10] J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer Academic, Dordrecht, 2000.
- [11] Potassco: The Potsdam Answer Set Solving Collection
- [12] Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A., A Logic Programming approach to Knowledge-state Planning, II: The DLVK System. Tech Rep. INFSYS RR-1843-01-12. Tu-Wien, 2003.
- [13] Gelfond M. (2004) Answer Set Programming and the Design of Deliberative Agents. In: Demoen B., Lifschitz V. (eds) Logic Programming. ICLP 2004. Lecture Notes in Computer Science, vol 3132. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-27775-0\\_2](https://doi.org/10.1007/978-3-540-27775-0_2).
- [14] Tiihonen, J.; Soininen, T.; and Sulonen, R. 2003. A Practical Tool for Mass-Customising Configurable Products. In Proceedings of the 14th International Conference on Engineering Design, 1290–1299. Edinburgh, UK: The Design Society.
- [15] Martin Gebser Torsten Schaub. Clasp: Answer Set Programming, the Solving Paradigm for Knowledge Representation and Reasoning. University of Potsdam, <http://potassco.sourceforge.net/>
- [16] Fernando Zacarias, Rosalba Cuapa, Luna Jimenez and Noemi Vazquez. Modelling of Intelligent Agents using A-Prolog. International Journal of Artificial Intelligence and Applications (IJAIA), Vol.10, No.2, March 2019.

## AUTHORS

**Dr. Fernando Zacarias** is full time professor in the computer science department into the Autonomous University of Puebla. He has directed and participated in research projects and development in this area. He is a researcher in practical and theoretical Computer Science and Mobile Technologies, and has conducted R&D projects in this area since 2000.



**Dra. Rosalba Cuapa** is a junior researcher into Autonomous University of Puebla. She is a researcher in Computer Science and Learning since 2005.

