Modelling Agents with A-Prolog

Fernando Zacarias Flores, Rosalba Cuapa Canto, Meliza Contreras, Oscar Garcia Pérez

***Abstract*— This Nowadays, research in computational agents capable of rational behaviour has grown widely. The formalizations of agents and their implementations have proceeded in parallel in different areas. In the games theory the behaviour of agents is relevant and necessary. We presented a novel approach about a computational agent that plays efficiently the well-known game connects four. This agent includes a reasoning module for decision-making based on A-Prolog (Answer Set Programming). Our aims in this article are firstly to briefly summarize the key concepts of decision theory and game theory. Next, we present a novel implementation about an agent in the game connect four that shows a perfect union between two different paradigms that have shown efficiency (A-Prolog & Java). This article shows the effective use of A-Prolog as a modeling language. We show that our A-Prolog-based approach can naturally satisfy the above requirements, through an A-Prolog encoding of the connects four game.**

***Index Terms*— A-Prolog, Computational agent, Game Theory, Reasoning.**

I. INTRODUCTION

In recent years, the agents paradigm has virtually invaded every subfield of computer science. Influencing a broad spectrum of disciplines such as reasoning, planning, among others. Agent technology is a methodology to realize an autonomous decentralized system with interactions among agents that model each element of the system. Although commonly implemented by means of imperative languages, mainly for reasons of efficiency. However, that efficiency has been reached at the moment by development of computational logic based on A-Prolog. In addition, the computational logic has shown to have a clear specification and correctness. Logic programming and non-monotonic reasoning have shown a growing interest in development of intelligent agents.

Logic programming, by virtue of its nature both in substance and method, provides a well defined, general, and rigorous framework for systematically studying computation, or attending implementations, environments, tools, and standards [1], [2]. The computational logic provides solutions, at a sufficient level of abstraction so that they generalist from problem domain to problem domain, afforded by the nature of its very foundation in logic. Particularly, we can to say that the computational logic has its major asset in both substance and method.

In this paper, we address our approach to the implementation of agents with this piece of reasoning used to play. We use extended logic programs under the answer set semantics, some very good recent reviews with many references are [3] and [4].

With respect to decision theory, decision theory allows us to analyze which of a series of options should be taken when we do not have a function to evaluate exactly what will be the result of taking the option. Decision theory allows us to identify the "best" decision option, although the meaning of the word “best” has diverse connotations, of which the most common is that which maximizes the expected utility of the decision maker.

On the other hand, game theory is a close relative of decision theory. In particular, it studies the problems of how interaction strategies can be designed that will maximize the welfare of an agent and how protocols or mechanisms can be designed that have certain desirable properties. The interest for these two theories has grown so much and particularly in the construction of agents that nowadays, there are many conferences, workshops, books, etc., in this direction.

II. KNOWLEDGE INTERPRETATION BY THE AGENT

Next, we consider the next interpretation under ASP [5]. Given a theory T, its knowledge is understood as the formulas F such that F is derived in T using intuitionistic logic. This makes sense, since in intuitionistic logic according to Brouwer, F is identified with “I knows F"” (or perhaps some reader would prefer to understand the notion of “knowledge” as “justified belief”).

An agent whose knowledge base is the theory T believes F if and only if F belongs to every intuitionistically complete and consistent extension of T by adding only negated literals (here “belief” could be better interpreted as “coherent” belief).

Take for instance: $\neg a \rightarrow b$. The agent knows $\neg a \rightarrow b$, $\neg b \rightarrow \neg\neg a$ and so on and so forth. The agent does not know however a. Nevertheless, one believes more than one knows, but a cautious agent must have its beliefs consistent to its knowledge. This agent will then assume negated literals able to infer more information. Thus, in our example, our agent will believe $\neg a$ and so he/she can conclude b. It also makes sense that a cautious agent will believe $\neg a$ or $\neg\neg a$ rather than to believe a (recall that a is not equivalent to $\neg\neg a$ in intuitionistic logic). This view seems to agree with a point of view by Kowalski, namely that “Logic and LP need to be put into place: Logic within the thinking component of the *observation-thought-action* cycle of a single agent, and LP within the belief component of thought”". As Pearce noticed, if we include strong negation we just have to move to Nelson logics [6]. However, if we want to stay in intuitionistic logic we can make a simple renaming as in [7].

III. A-PROLOG AS MODELING LANGUAGE

A-Prolog is an answer set solvers based on answer set programming paradigm [8]. It supports a powerful language

Fernando Zacarias Flores, Computer Science, Universidad Autónoma de Puebla, Puebla, México, +52 2222295500
Rosalba Cuapa Canto, Faculty of Architecture, Universidad Autónoma de Puebla, Puebla, México, +52222295500
Meliza Contreras González, Computer Science, Universidad Autónoma de Puebla, México.
Oscar García Pérez, Coputer Science, Universidad Autónoma de Puebla, Puebla, México.

extending Disjunctive Datalog with many interesting features such as strong and weak constraints, functions, etc. We illustrate its input language and give indications on how to use it for representing knowledge in applications where advanced knowledge modeling capabilities are necessary.

ASP is a kind of logic programming, which represents solutions to a problem by answer sets, and not by answer substitutions produced in response to a query, as in conventional logic programming, in many of the occasions represented by Prolog. The answer sets for a logic program can be described as the satisfying interpretations for a set of propositional formulae.

A. *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, …;
connectives: $\wedge$, $\vee$, $\leftarrow$, $\bot$; and *auxiliary symbols*: "(", ")", ".", where $\wedge$, $\vee$, $\leftarrow$ are 2-place connectives and $\bot$ is a 0-place connective. Formulae are defined as usual. The formula $\top$ is introduced as an abbreviation of $\bot \leftarrow \bot$, not F as an abbreviation of $\bot \leftarrow F$, 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 L_F, 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*, $\bot \leftarrow B$. Facts and constraints are sometimes written as H and $\bot \leftarrow B$ respectively.

IV. Evolving Agents to Play Connect Four using A-Prolog

The Connect four is a two players game identify as the black chips player (agent) and the white chips one (user), this game take place in a rectangle shaped board with seven columns and six rows. Each player has twenty-one chips. Turns or movements develop the game, where each movement attach a chip by the player in the board. The player with white chips starts the game. If the player puts a chip in a column, this one must go to the lowest free cell of that column. As soon as a column has six chips, it can not be placed any other chip in that column. The objective of each player is place chips until get four chips connected in a line horizontally, vertically or diagonally. The first player that reach the objective wins the game. By the other side, if the forty-two chips are placed in the board and non-player has reached four chips connected, the game finishes in tie.

V. The Agent Playing Connects Four

An intelligent agent is a computer program placed in an environment and it is able to act in an autonomous way in this environment with the main objective of win. The agent must show reasoning actions in such way that it can reach the main objective. For this, the agent uses a set of rules that allow him decide in every moment which is the best move, so, in this way reach partial objectives that leads to the main objective. Next, we describe the general strategy modeled for our agent.

A. *The Agent's General Strategy*

We suppose that our agent (player) of connects four is defined by the next elements:

1) Objectives; 2) Environment; 3) Perceptions; 4) Actions and 5) its Knowledge.

Among the objectives, the main objective is to win the game, as well as secondary objectives that include, block, build and learn from lost games.

1. **Objectives**. The objective of the agent is win the game.
2. **The Environment**. The player moves in an environment compound by a board where another independed player (an opponent - the user) moves by turns. Each player has one different chip placement. Time limitations may exist to choose the next move.
3. **Perceptions**. We assume that the agent is capable of to perceive the current state of the game (thought a DLV file) before perform every move.
4. **Actions**. The player can propose valid movements for the game.
5. **Knowledge**. The player needs a strategy, that allow him propose a valid movement following the game rules. The capacity to reach the objective will depend of the physical limitations (for example, the calculus time) in DLV the calculus is efficient. A basic case could be limit to give one valid movement, maybe selected randomly. The capacity of the game may improve if we have knowledge to evaluate the board positions (We have it in DLV), and more on, we can perform a more effective search. In the last case the calculus of a movement can take many resources, so is usual for the player has limited quantity of time to execute the ideal search of a move.

Then, we can set up a preliminary general vision of the relation between the different kinds of players in base of the next diagram.

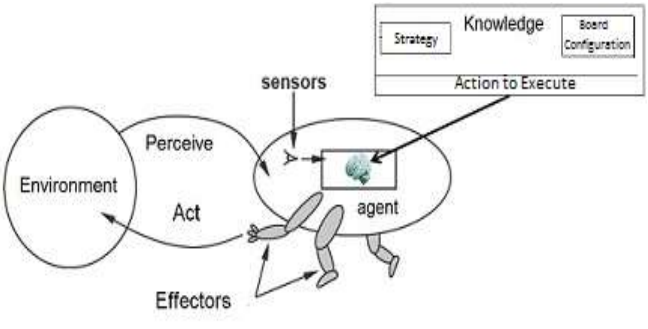

Figure 1. Agent: General Architecture

In figure 1, we can observe how our agent counts with the follow configuration: Our agent has a knowledge base conformed by five files containing: First, a simple strategy, it contains two basic rules of game. One of these it consists on blocking a line containing two chips. In addition, another rule that blocks a hole between a line of two and a chip to form the line of four. Second, the complex strategy, it contains several

rules such as: Winning shot, tree alignment block shot, two alignment block shot, building a tree alignment, building a two alignment and first bottom shot. Third, also, our agent has another three files that allow him to maintain a general environment about the development of the game.

Next, we present some of the rules implemented for the agent are:

1. **Simple strategy:** Set of rules for a basic level game some of the employed rules in the strategy are:
   g(X,Y,r):-cell(X,Y,v), vertical win move
   cell(X,Z,r), Y=Z+1,cell(X,W,r), Y=W+2,
   cell(X,R,r), Y=R+3.
   t(X,Y,r):-cell(X,Y,v), horizontal left alignment
   cell(Z,Y,r), Z=X+1,cell(W,Y,r), W=X+2.
2. **Complex strategy:** Set of rules for an expert level game
   b(X,Y,r):-cell(X,Y,v), cell(Z,Y,a), Z=X+1, cell(W,Y,a),
   W=X+2, cell(R,Y,a), R=X+3.
3. **Available cells:** File with information about available cells: Set of available cells in every moment.
   cell(0,3,v). cell(1,4,v). cell(3,2,v). cell(4,1,v). cell(5,1,v). cell(6,0,v).
4. **Busy cell:** File with a set of unavailable cells that may contain black or white chips.
   cell(0,1,r). cell(0,2,a). cell(1,0,r). cell(1,1,r). cell(1,2,r). cell(1,3,a). cell(2,0,a). cell(2,1,a). cell(2,2,a). cell(2,3,r). cell(2,4,r). cell(2,5,r). cell(3,0,a). cell(3,1,a). cell(4,0,a).
5. **Tokens to move:** Set of chips that can be moved by the answer sets calculus provided by A-Prolog.
   {b(6,0,a), c(1,4,r), s(6,0,r)}

Tokens to move correspond to the group of answer sets calculated by A-Prolog. This set allows our agent to execute the best action.

VI. Implementation in A-Prolog

We have developed a DLV program as Back-End and a Java program as Front-End that presents the game scenery, this is, the game execution between a person and a procedure acting as an opponent player (agent).

The variables “a” and “r” point to the white and black respectively. The variable “v” points to the empty state.

Is understood as cell(X,Y,E) a position (X,Y) of the game board compose with an E attribute that defines who is taking that position. For example:

```
cell(0,0,a)        %(0,0) is taken by a white chip
cell(0,0,r)        %(0,0) is taken by a black chip
cell(0,0,v)        %(0,0) is empty
```

The structure is described as follow:

```
F(X,Y,e):- cell(X,Y,v),
        ...,
        ...,
        ...,
```

Is derived the consequent F(X,Y,e) if the exposed conditions in the predecessor are satisfied.

F can be:

1. g(X,Y,r): if it is a winning play.
2. b(X,Y,r): if it is a blockade to a line of 3.
3. c(X,Y,r): if it is a blockade to a line of 2.
4. t(X,Y,r): if it is to advance one it lines from 2 to 3.
5. s(X,Y,r): if it is to advance one it lines from 1 to 2.
6. p(X,0,r): if it is to make a play in the first line.

The exposed conditions determinate if the empty cells make vertical, horizontal or diagonal lines with the taken cells, such way that can serve us to block our opponent, move forward or win the game.

For example:

```
g(X,Y,r):-cell(X,Y,v),
          cell(X,Z,r), Y=Z+1,
          cell(X,W,r), Y=W+2,
          cell(X,R,r), Y=R+3.
```

It indicates the following:

We deduce g(X,Y,r) if the cells cell(X,Y,v) exist and cell(X,Z,r), cell(X,W,r), cell(X,R,r) exists too, and are taken by black chips and form a vertical line that lead us win the game, Y=Z+1, Y=W+2, Y=R+3.

In this way, we describe the rest of sentences to win, but considering horizontal and diagonal lines. The sentence to block and move forward are derived for the sentence to win. For example, the sentences for block an alignment of tree are the same statement to win, the difference is that we ask for the white ones, no the black. The sentence to move forward in two chips alignment to a tree chips alignment is ask for two of the predecessor cells of an empty one. The sentence to make a move on the first line only check the existence of the cell with Y equals to zero and if is empty.

VII. Strategy based on A-Prolog

We try that A-Prolog give us a set of answers with the most appropriate cells to perform the next move. For this, we give the current condition of the game, this means, the cells that are empty taken by both players and cells where is valid a move.

We receive the results in the follow order:

1. Winning shot.
2. Tree alignment block shot.
3. Two alignment block shot.
4. Building a tree alignment.
5. Building a two alignment.
6. First bottom shot.

The reason of this order is the priority assigned of the movements If exists a possibility to win the game in the current move we will not waste it. In the same way, if is necessary block, we will not make any other movement.

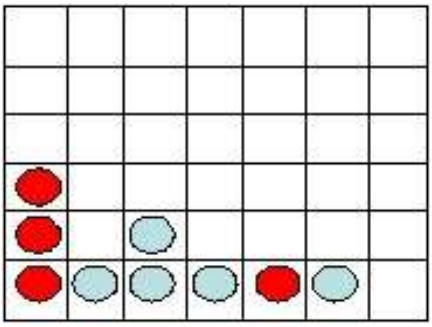

Figure 2. Connects Four

Let us suppose the following conditions (figure 2):

The red chips player turn. Viewed in A-Prolog perspective the current game are described by:

cell(0,0,r). cell(0,3,v). cell(0,1,r). cell(1,1,v). cell(0,2,r). cell(2,2,v). cell(1,0,a). cell(3,1,v). cell(2,0,a). cell(4,1,v). cell(2,1,a). cell(5,1,v). cell(3,0,a). cell(6,0,v). cell(4,0,r). cell(5,0,a).

With the deductive rules attached to these facts, the answer set will be:

{g(0,3,r), c(2,2,r), t(0,3,r), s(3,1,r), p(6,0,r)}

This means:
A winning shot.
A two line blocking shots, and so on.

The agent counts with two A-Prolog files that bounds the player level.
a) Square.dlv beginner
b) Square.dlv expert

The first level, corresponds to beginners, and excludes the next conditions:

- To block one lines of two.
- Block the empty cell between a two alignment and a chip to avoid a four-alignment chip.
-

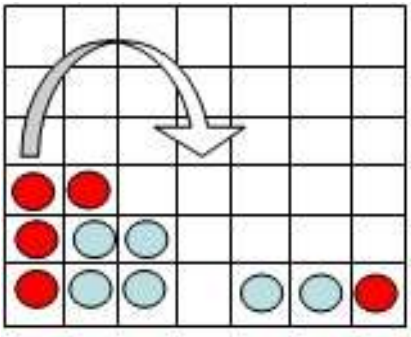

Figure 3. Connects-Four winner

As we can see (figure 3), the opponent performs his shot to move forward getting a tree line alignment and leaving the blue chips player in the position to make a four line alignment.

- Winning with an empty cell between two-line alignment and a chip in order to create a four-line alignment.

Just like in the previous case, the opponent let past the opportunity to win the game. This type of deficiencies are makes them the beginner level. However, in level two, we consider all possible ways to achieve a four-line alignment. In addition, the importance of reduce the opportunity that the opponent have to form several tree line alignments by blocking all two line alignments. So, in the state shown in the figure 3, win the one that has the black cells.

VIII. Incorporating Learning to the Agent

In this section, we present a new component of learning that we have add to our agent. So that an agent is autonomous, agents must employ some form of learning capacity. This capacity will allow being possible to build true intelligent agents. A learning component will allow the agents to improve the efficiency of its game, just as it happens in the case of people. In the general model of discounted repeated games with imperfect information, the set of payoffs attainable via pure-strategy sequential equilibrium becomes larger as the observability of the past actions increases. We use the information generated in game that has gotten lost previously, with the purpose that the agent can learn of this.

This component allows to our agent to learn by means of a process based on the repetition. It is important to point out that this process needs to be continued working incorporating a more robust methodology.

IX. Conclusion and Future Work

A Computational logic and particularly A-Prolog proved to be a successful approach to several aspects of agent systems design. Knowledge and reasoning are important for artificial agents and form the cornerstone of successful behavior. In this work, we have presented an interesting example about logic programming-based agent reasoning applied to intelligent agents. At the same time, from the computational logic side we are witnessing a growth in the interest for agent-systems and multi-agent systems. This is important, and at the same time outstanding, since there are many examples like the one presented in a paper that demonstrate its efficiency and clarity. Our application using ASP is a recent direction of research seems to push toward a new idea of intelligent systems combining two paradigms: the object oriented and logic programming, eliminating the traditional high gap between theory and practice; this integration represents an important added value to the design of intelligent systems based on agents and supported by formal theories.

As future work, we are working in the agent's design that allow us to be carried out tasks similar to the one presented here, but another games such as: The One game, The Domino game, etc, with the purpose of being able to transfer this methodology to applications of e-bussines, e-commerce, e-knowledge, etc. For later on, to incorporate update processes to agent's knowledge base. This is very important for applications where our agent's environment is dynamic.

Acknowledgment

We thank the anonymous referees for their useful suggestions. The academic group of combinatorial algorithms and learning supports this work.

References

[1] M Gelfond, V Lifschitz. The stable model semantics for logic programming. ICLP/SLP 88, pp.:1070-1080, 1988.
[2] Robert Kowalski. Computational Logic and Human Thinking: How to be Artificially Intelligent. Cambridge University Press. 978-0-521-19482-2, 2011.
[3] Chitta Baral, Gregory Gelfond, Tran Cao Son and Enrico Pontelli. Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge, pp. 259-266, 2010
[4] Tran Cao Son, Enrico Pontelli, Gregory Gelfond and Marcello Balduccini. Reasoning about Truthfulness of Agents Using Answer Set Programming. Proceedings, Fifteenth International Conference on Principles of Knowledge Representation and Reasoning, 2016

[5] Mauricio Osorio, Fernando Zacarias. On Updates of Logic Programs: A Properties-Based Approach. Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS 2004, Wilhelminenberg Castle, Austria, February 17-20, Proceedings. Lecture Notes in Computer Science 2942, Springer, pp.231-241, ISBN 3-540-20965-4,2004
[6] V. Lifschitz, D. Pearce and V. Valverde. Strongly Equivalent logic programs. ACM Transactions on Computational Logic, pp. 526-541. 2001
[7] Helena Rasiowa. An Algebraic Approach to Non-Classical Logics Journal of Symbolic Logic. Volume 42, Issue 3, 1977
[8] Tran Cao Son, Enrico Pontelli, Gregory Gelfond and Marcello Balduccini. Reasoning about Truthfulness of Agents Using Answer Set Programming. Proceedings, Fifteenth International Conference on Principles of Knowledge Representation and Reasoning, 2016.

Fernando Zacarias Flores is researcher and professor of Computer Science at the Autonomous University of Puebla. He is a researcher in practical and theoretical Computer Science and Mobile Technologies, and has conducted R&D projects in this area since 2000. Results from these projects have been reported in more than 60 national and international publications. Professor Zacarias serves in the editorial board of the following journals: IEEE Latin America Transactions, Engineering Letters, International Transactions on Computer Science and Engineering, Common Ground Publishing - Technology, Learning and Social Sciences. He holds a Ph.D. degree in Computer Science from UDLAP, M.Sc. in Electrical Engineering from CINVESTAV-IPN and B.Sc. in Computer Science from BUAP.

Rosalba Cuapa Canto is full-time professor at the Autonomous University of Puebla. She is a researcher in practical and theoretical mobile technologies. She holds a PhD degree in Computer Science from Universidad de Puebla. M.E. in Universidad de Puebla and B.Sc. in Computer Science from BUAP.

Meliza Contreras Gonzlez is full-time professor´in Computer Science at the Autonomous University of Puebla.