ANSWER SET PROGRAMMING (DLV – CLINGO):CONNECT 4 SOLVER

Rosalba Cuapa C.[1] and Fernando Zacarias F.[2] and Adair Ponce.[1]

[1]Faculty of Architecture, Universidad Autónoma de Puebla, Puebla, México
[2]Department of Computer Science, Universidad Autónoma de Puebla, México

ABSTRACT

Answer set programming is an approach that has recently gained a lot of popularity in solving complex problems due to its efficient knowledge representation and reasoning. Knowledge is represented as answer set programs, and reasoning is performed by answer set solvers. Answer set programming enables default reasoning, which is required in commonsense reasoning. In this paper, we discuss the potential role of answer set programming (ASP) in the context of approaches to the development of agents especially in the realm of Computational Logic. So, we focus on the usage of DLV for solving problems in a declarative manner particularly, in the solution of the game "connect four". Additionally, we present the solution to this same problem from Clingo's perspective. We address this challenge with the ASP system clingo and its grounding and solving components by equipping them with well-defined generic interfaces facilitating the manifold integration efforts.

KEYWORDS

Asnwer Set Programming, Clingo, Dlv, Logic, Agents,

1. INTRODUCTION

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 [3] [4]. 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.

Agent technology is a methodology to realize an autonomous decentralized system with interactions among agents that model each element of the system. However, that efficiency has been reached by development of computational logic based on Answer set programming. 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]. Agent modelling under the logic programming paradigm is done in a simple and transparent way, regardless of the resolver or tool we use for modelling (DLV or Clingo) [3], [4].
The paper is structured as follows. In Section 2 we briefly introduce answer set programming as the paradigm for the knowledge interpretation by the agent. In Sections 3 and 4 In 3 and 4 we

present the implementation of our agent, in the first instance in dlv and in a second moment using the clingo paradigm, in which we incorporate a heuristic function. Finally, we give our conclusions and future work about this proposal.

2. KNOWLEDGE INTERPRETATION BY THE AGENT

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" [5], [12].

2.1. Construction of The Reasoning Module of an Agent for Games

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. The agent concept has recently increased its influence in the research and development of computational logic–based systems [6]. For this reason, it is important to consider the modelling of the reasoning module of an agent for games in general and for zero-sum games. Analyse the possibilities of a data type so that the agent can participate by applying an appropriate strategy. To form an effective reasoning module, the following is required:

- Objective:
  - The agent has a knowledge base conformed by the following rules.
- Start:
  - the 7 columns must be evaluated to decide which has a greater value
- The middle:
  - The agent must have a strategy with well-defined priorities.
  - The agent's main goal is to win the game.
  - The agent develops in a medium formed by a board where another independent player (an opponent) moves in turns.
  - Each player has a different kind of chip.
  - There may be time constraints to choose the next move.
  - The agent can perceive the game state through the board before making each move.

- Actions:
    - The agent can propose a valid move of the game.
- Knowledge
    - The agent needs a strategy that allows him to propose a valid move according to the rules.
    - The agent needs to prioritize his moves to win the game.
        - ✦ Win
        - ✦ Block in case the opponent has three continuous chips.
        - ✦ Build the game based on the strategy defined by the agent (applying heuristics).
        - ✦ Define the start of the game by choosing the middle row.

Considering the above points, 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 to decide in every moment which is the best move, so, in this way reach partial objectives that leads to the main objective.

For this reason, the proposed agent must consider the following:1) Objectives; 2) Environment; 3) Perceptions; 4) Actions and 5) its Knowledge. All these necessary and relevant characteristics for the proper functioning of our agent can be summarized as follows:

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 independent 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 or Clingo file) before performing every move.
4. **Actions**. The player can propose valid movements for the game.
5. **Knowledge**. The player needs a strategy, that allow him to 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 or Clingo the calculus is efficient. A basic case could be limited 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 or Clingo), 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.

3. IMPLEMENTATION OF THE AGENT IN DLV

Our agent has a knowledge base 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. Besides, also, our agent has another file that allow him to maintain a general environment about the development of the game [7], [12].

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

1. 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 Dlv.
{b(6,0,a), c(1,4,r), s(6,0,r)}

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

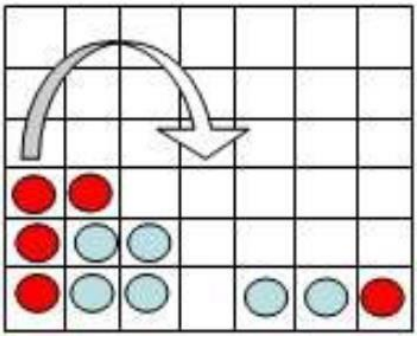

Figure 1. Connect four winner in Dlv

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, not 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.

As can be seen, choosing the paradigm based on answer set programming gives us a simple, clear, and adequate opportunity to model complex problems in a transparent way. As shown in the solution presented in dlv, the way in which this type of complex problem is modeled is simple and clear [11].

4. IMPLEMENTATION OF THE AGENT IN CLINGO

Based on the abstraction of logic programming, Answer Set Programming has become an established paradigm for knowledge representation and reasoning, particularly when it comes to solving complex problems. At this point, we return to the game previously presented and implemented in Dlv called "connect 4", however, from this moment we make use of a logic program solver called "Clingo", also based on answer set programming. Clingo has an easy and expressive way to model these problems. using constraints to find solutions that satisfy the constraints of the game.

In Clingo [9], unlike Dlv, the evaluation of all possible movements is done through the minimax heuristic. The heuristic called minimax [8] is used to evaluate positions by recursively examining all possible future moves to a certain depth and returning the best evaluated move. To find out which is the best move among all the possible options, the heuristic function evaluates the horizontal, vertical, and diagonal positions by looking for a specific number of consecutive pieces of the same type in a row. If a win is found, the score is added by 100 for player 1 or subtracted by 100 for player 2. If no win is found, the score is set to 0.

4.1. Mini Max Algorithm

The Minimax algorithm (Schiffel & Thielscher 2009) is a method used in artificial intelligence to make optimal decisions in two-player game situations with perfect information, such as chess, checkers, Connect 4, among others. The algorithm works by generating a search tree that represents all possible moves and responses to those moves, from the current state of the game to a final state. The algorithm determines the best move a player can make at each level of the search tree, assuming the opponent is also playing optimally. At the top level of the tree, the

algorithm considers all possible moves by the current player and simulates the opponent's optimal response to each move. It then evaluates each of these possible outcomes to determine which is the best option for the current player, using a heuristic evaluation function. This process is repeated recursively for all levels of the search tree, alternating between the current player and the opponent, until an endgame state is reached. The Minimax algorithm then returns the best move that the current player can make at the top level of the search tree out of all possible moves that will be evaluated.

The Connect Four game is considered a zero-sum game because each player competes to reach the goal of connecting four tiles of the same colour in a row, column, or diagonal. If one player manages to connect four tiles, they win the game, while the other player loses. Thus, one player's profit comes at the other player's expense, and the sum total of wins and losses will always be zero. It is important to note that although the game is zero-sum, this does not necessarily mean that it is a zero-sum game, that is, that there are no external benefits or secondary effects for the players or third parties.

4.2. Alpha-Beta Pruning to Optimize.

Minimax works exploring all possible moves and game states, building a search tree. Each tree node represents a game state, and each arc represents a possible move. The score at each leaf node is the final score of the game. Scoring is assigned so that MAX players try to maximize and MIN players try to minimize. The depth search technique is used to explore all possible moves. However, the number of nodes in the search tree grows exponentially with the depth of the tree, making full exploration impossible for complex games.

To solve this problem, the alpha-beta pruning technique is used, which reduces the number of explored nodes. The alpha-beta pruning algorithm maintains two values at each tree node: alpha and beta. Alpha represents the minimum score that MAX can achieve when exploring the left subtree of the node, while beta represents the maximum score that MIN can achieve when exploring the right subtree of the node.

Alpha-beta pruning works by removing subtrees that do not need to be explored. If the value of beta at one node is less than or equal to the value of alpha at another sibling node, then the entire right subtree of the first node can be deleted, since MIN would never choose that branch, since MAX has a better option available. Similarly, if the value of alpha at one node is greater than or equal to the value of beta at another sibling node, the entire left subtree of the first node can be removed, since MAX would never choose that branch, since MIN has a better branch. option available.

The implementation in CLingo is based on the ASP paradigm, where we must create lines of code made up of a preposition followed by the conditions that will allow it to be fulfilled. In order to start a connect four game we must declare the essential parts of the game, which are a board, the players, the tiles and the cells where it is possible to shoot. In connect four we have a board that is a 7x6 matrix where the pieces will be placed with respect to a position defined by an ordered pair X, Y. Our board will be created with a predicate named cell that allows us to know where the pieces can be moved. chips according to the dimensions specified above.

```
cell(1..7, 1..6).
```

Game board definition connect 4 in clingo

The next step is to give the player a turn, which will change each time a player move is made.

We use currentPlayer(_) to indicate if the current player is the one making the move, if this condition is met it will be given a boolean value of True and otherwise it will be False. Each player will be recognized by chips, “X” in the case of the human player and “O” in the case of artificial intelligence, and they will only be able to shoot in those cells evaluated as empty. To recognize empty cells, a predicate with two conditions is used: empty(X, Y) will be true only when there is no value within said cell and it is a cell within the range (X,Y).

empty(X, Y) :- cell(X, Y), **not** value(X, Y, _).

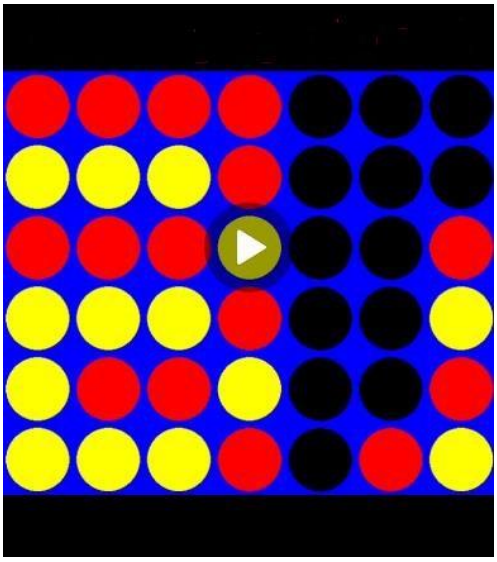

Figure 2. Connect four winner in Clingo

Once the bases of the game have been defined, we must define the actions that can be carried out on them. The movements are the most vital part of a connect four game, because depending on the movement that our opponent makes, we can make a movement that can bring us closer to victory or block the opponent. The actions that we carry out before a movement is to think of several possible movements, evaluate which of these is the best and once decided which one to carry out, then we proceed to carry it out. For each of these actions, a predicate is created with its respective conditions, starting with the possible movements. Instead of making a movement over the entire matrix, we will make it over a vector which will only move between the columns of the X axis, leaving the axis stable. movement on one of the six rows of the Y axis, that is, an X value will be chosen under two conditions.

This code defines the game state and implements a minimax algorithm with a heuristic function to evaluate the position. The game state is represented by a cell(X,Y) predicate that defines the position of each cell on the board, and the value(X,Y,P) and empty(X,Y) predicates that define the value(color part) of a cell and if it is empty, respectively. The current player is represented by the predicate currentPlayer(P). Possible moves are generated by the move(X) predicate, which looks for empty cells in the bottom row and top row of any existing pieces. The predicate evaluates the score of a position score(X, Score), which first checks if the current player can win with move X and assigns a score of 100 or -100 accordingly. If there is no immediate win, the algorithm calls minimax to recursively evaluate the position with a lower depth. The position score is the maximum or minimum of the scores of the possible moves, depending on the player. Finally, the makeMove(X) predicate updates the game state by placing a piece of the current player in cell X. The undoMove(X) predicate undoes the move by removing the piece from cell X. The heuristic function is defined by the heuristic(Score) predicate), which checks for

horizontal, vertical, and diagonal gains and assigns a score of 100 or -100 accordingly. If there is no win, the score is 0.

4.3. Next, we Show Part of the Code in Clingo

```
% Evaluate a position using minimax algorithm
minimax(Depth, Score) :- Depth = 0, heuristic(Score).
% Evaluate a position using a heuristic function
heuristic(Score) :- horizontalScore(Score); verticalScore(Score); diagonalScore(Score).

% Evaluate a position based on the number of consecutive pieces in a row
% Check for horizontal win horizontalScore(Score) :- currentPlayer(true), value(X, Y, "X"), value(X+1, Y, "X"), value(X+2, Y, "X"), value(X+3, Y, "X"), Score = 100.

horizontalScore(Score) :- currentPlayer(false), value(X, Y, "O"), value(X+1, Y, "O"), value(X+2, Y, "O"), value(X+3, Y, "O"), Score = -100.

horizontalScore(Score) :- Score = 0.
```

5. CONCLUSIONS AND FUTURE WORK

A Computational logic and particularly Answer set programming 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 behaviour. In this work, we have presented an interesting example called connect four, but it is also presented in two paradigms based on answer set programming such as Dlv and Clingo. The two grounders performed perfect on both samples. All tested solvers had no problems solving disjunction-free encodings (admissible, stable, grounded and complete) and it would be possible to even go further. When it comes to disjunctive encodings (semi-stable and preferred) only dlv and clingo were able to solve all. Finally, we can mention that by incorporating the minimax algorithm and alpha-beta pruning to optimize we have managed to obtain an infallible solution, that is, the version of the game connects four developed in clingo guarantees to win if the machine starts the game. In addition, we have shown how both proposals complement each other naturally with front-end languages, i.e., java and python with dlv and clingo respectively.

In this presentation, two proposals for solutions to a complex problem such as connect four have been shown.A future line of research will be to implement in both tools (Dlv and Clingo) a problem that allows us to measure which one performs better in the model's calculation. Finally, we can comment that both tools allow connectivity with front ends. In our proposal, in the case of dlv, we use java as the front-end to interact with the end user. While in the case of Clingo we use python as the front-end part that allows interaction with the end user,

ACKNOWLEDGEMENTS

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] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, Sven Thiele. A User's Guide to gringo, clasp, clingo, and iclingo (ver. 3.x)

[4] Abeer Dyoub, Stefania Costantini and Giovanni De Gasperis. Answer set programming and agents, Cambridge University Press: 05 November 2018.

[5] Zacarias F. "Belief revision and Updates under Nonmonotonic Reasoning: An answer set approach". UDALP Thesis, 2004.

[6] De Vos Marina and Vermeir Dirk. Logic Programming Agents and Game Theory. AAAI Technical Report SS-01-01. Compilation copyright © 2001, AAAI (www.aaai.org). All rights reserved.

[7] Zacarias Fernando, Cuapa Rosalba, Contreras Meliza, Garcia Oscar. Modelling Agents with AProlog. International Journal of Engineering and Applied Sciences (IJEAS).ISSN: 2394-3661, Volume-5, Issue-6, June 2018.

[8] Stephan Schiffel and Michael Thielscher. Automated Theorem Proving for General Game Playing. Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence IJCAI-09, Publisher AAAI Press. 2009.

[9] Potassco project 2000. the Potsdam Answer Set Solving Collection. Tools for Answer Set Programming developed at the University of Potsdam. Germany https://potassco.org/

[10] Höschler Enrico. Answer set programming: Sudoku solver. Jul 10, 2018. https://ddmler.github.io/asp/2018/07/10/answer-set-programming-sudoku-solver.html

[11] Fernando Zacarias, Rosalba Cuapa, Guillermo De Ita, J.C. Acosta and Daniel Torres. PLANNING SOLUTIONS IN THE REAL WORLD. International Journal of Artificial Intelligence & Applications (IJAIA), Vol. 5, No. 3, May 2014.

[12] Zacarias, Fernando and Cuapa, Rosalba and Jimenez, Luna and Vazquez, Noemi, Modelling of Intelligent Agents Using A–Prolog (March 31, 2019). International Journal of Artificial Intelligence and Applications (IJAIA), Vol.10, No.2, March 2019, Available at SSRN: https://ssrn.com/abstract=3383952 or http://dx.doi.org/10.2139/ssrn.3383952

AUTHORS

Dr. Fernando Zacarias is full-time professor in the computer science department into the Universidad Autónoma de Puebla. He has directed and participated in research projects and development in the area of artificial intelligence and Computer science from 1995.

Dra. Rosalba Cuapa Canto is a full-time professor in the faculty of Architecture at Universidad Autónoma de Puebla. She has participated in research projects and development in the area of Computer Science