Ary, a general game playing program

Jean Méhat, Tristan Cazenave

Abstract

We present our program Ary that won the 2009 and 2010 General Game Playing competition. In General Game Playing (GGP), players have to play games they never encountered before. The rules are transmitted as a description in a specific language and the player have to automatically analyse the rules and select a method of playing.

1 Introduction

The field of program playing games is an extensively explored field of computer science and more particularly of artificial intelligence as it is a domain where common methods of reasonning are to be applied, while offering a closed world, completely described by the game rules. Moreover the quality of the play is easy to evaluate quantitatively by observing the matches outcomes, and a human expertise has been developped. Great successes have been obtained: in many games, including chess, programs routinely outplay even the best human player.

The program usually embed a important body of knowledge that is specific of the game they play. This knowledge is used by the designer beforehand and limit somewhat the generality of the program. While a program like Deep Blue is able to play well chess, it can not play a match of Checker, or even Tic Tac Toe: while an expert in its domain, the playing program is limited to one game in its abilities, and these are not easily extended to other domains or even to other games.

The *Logic Group* at the university of Stanford addresses this limitation with GGP. In a GGP match, the players receive the rules of the game they have to play in a specific langage called *Game Description Language* from a Game Master. The players have a set time, usually between 30 seconds and 20 minutes, to analyse the game. After that analyse phase, every player repeatedly selects a move in a fixed time, usually between 10 seconds and 1 minute, and sends it to the Game Master that combines them in a joint move transmitted back to all the players.

The *Logic Group* organize an annual competition at the summer conference of the Association for the Advancement of Artificial Intelligence (AAAI) [3].

As they do not know beforehand the games that will be played, General Game Player have to analyse the rules of the game to select a method that work well for the game at hand, of use only methods that work well for all the conceivable games.

In the following of this article, we will present the Game Description Language, sketch the method used by our program and present some examples of games played in the past competitions.

2 The Game Description Language (GDL)

The GDL is used to describe the status of a match, and its consequences, using only a very small set of properties that are specific of game description. It is semantically equivalent to Datalog, a restricted version of the Prolog logic programming language[5].

It is expressed in Knowledge Interchange Format (KIF), whose syntax is reminiscent of the Lisp programming language. For example, to indicate that the X player has marked the center cell of a Tic Tac Toe grid may be expressed with `(cell 2 2 x)`. Note that cell is *not* a special world in the GDL; in the following, special words will be noted in UPPERCASE.

2.1 The elements of the GDL

The set of game playing specific properties is as follows:

- `(ROLE` *player*`)` is used to identify the players that take part in a game. The GDL can be used to describe games with any number of players, including one player games that amount to puzzles and many players games.
- `TERMINAL` is a property that holds when the game is finished.
- `(GOAL` *player reward* `)` describes what a player obtain when a game is finished. The reward is an integer varying from 0 to 100. In a zero sum two players games, the winner obtains a reward of 100 and the looser 0 and a draw is usually translated by reward of 50 for all the players. As the reward is defined independantly for the players, it is possible to describe non zero sum games, where whe players best interest may be to cooperate.
- `(LEGAL` *player move*`)` is the property that describes what the legal moves are for a given player in a given status. Every player has to play a move at each step of a game. For turn taking games, there is usually a no-operation move to be played by the inactive players.
- `(DOES` *player move*`)` indicates the move played by a specific player in the last turn.

There is also a set of properties that are used to manage a base of facts that may be valid during the games, used to describe the match status. These are:

- `(INIT` *fact*`)` for facts that are true at the beginning of the match. It is used to describe the starting position in the game.
- `(NEXT` *fact*`)` describes the facts that will be true after a move. It is used to define the new fact base to be used to describe the new game status.

- `(TRUE` *fact*`)` allows one to verify that a fact is true in the present status.

There is set of theorems that allows one to produce deductions from the game rules and the fact base. It is expressed as `(<=` *property list-of-properties*`)`, indicating that the indicated property holds if all the properties of the list are true. There is also a `NOT` property allowing one to make deduction on the basis of properties that are *not* true in a given status. For example, a theorem to prove that the player O has completed the first row at Tic Tac Toe may be:

```
(<= (row o)
      (TRUE (cell 1 1 o))
      (TRUE (cell 1 2 o))
      (TRUE (cell 1 3 o)))
```

which translates into Datalog with

```
row(o) :- true(cell(1, 1, o)),
          true(cell(1, 2, o)),
          true(cell(2, 2, o)).
```

and in natural language with *If the first, second and third cells of the first row all contain an O, the O player has completed a row.*

The right part of the theorem is actually a way to express a logical conjonction (an `AND`). In the current GDL, there is nothing specifically used to describe a logicial disjonction (an `OR`). For example to indicate that a Tic Tac Toe player has completed a line if it has a row or a column or a diagonal, one has to use three theorems as in:

```
(<= (line o)(row o))
(<= (line o)(column o))
(<= (line o)(diagonal o))
```

These elements are theorically sufficient to describe a game: every game status can be described with a set of properties in the base fact, every legal move and the effects of joints moves in any given status can be described with a theorem. However, in interesting games, the number of possible status is too big for this extensive description to be practical. The GDL description of a game may include *variables* to generalize the theorems. For example, instead of using three theorems to describe the completion by the X player of one of the three columns of the board, one theorem with a variable can be used, as in:

```
(<= (column x)
    (TRUE (cell 1 ?n x))
    (TRUE (cell 2 ?n x))
    (TRUE (cell 3 ?n x)))
```

The underlying machine will try to unify the n variable with the every possible value, and `(column x)` will be demontrated if for one of these values, the three facts are verified.

The presence of variables make necessary the property *DISTINCT*, used to verify that a variable does not has a given value. For example, stating that a marked cell will keep its mark until the end of the match can be done with the following theorem:

```
(<= (NEXT (cell ?x ?y ?status))
    (TRUE (cell ?x ?y ?status))
    (DISTINCT ?status empty))
```

(Empty cells cannot be propagated so simply, as the one selected by the active player will not be empty in the next step.)

2.2 Limitations of GDL

The games described in GDL can not be infinite: this is practically realised by adding a counter to the game description, that is incremented after each move. When the counter attain a given value without reaching a terminal state, the game is declared finished and scored.

The GDL is limited to first order logic, in contrast to the Prolog programming language. The variables can not be bound to predicates, so it is impossible to define arithmetic in GDL. This makes necessary do define arithmetic in the game description, when necessary, with an explicit enumeration of the possible values as in:

```
(<= (greater ?x ?y) (succ ?x ?y))
(<= (greater ?x ?y) (succ ?x ?z) (greater ?z ?y))
(succ 2 1)
(succ 3 2)
...
```

2.3 Examples of game descriptions in the GDL

Turn taking games are described with the GDL elements, for example with the following excerpt:

```
(ROLE white)
(ROLE black)
(INIT (control white))
(<= (NEXT (control white)) (TRUE (control black)))
(<= (NEXT (control black)) (TRUE (control white)))
(<= (LEGAL white noop) (TRUE (control black)))
(<= (LEGAL black noop) (TRUE (control white)))
```

The two `ROLE` properties enumerate the two player names and the `control` predicate is used to indicate who is the active player on this step. The two `NEXT` theorems ensure that the player in control will be swapped after each step. We included the theorems stating that the (only) move of the player not in control is a *no operation*.

With the help of an `other` property and variables, these four theorems can be reduced to two, as in:

```
(other white black)
(other black white)
(<= (NEXT (control ?player)) (TRUE (control ?x)) (other ?player ?x))
(<= (LEGAL ?player noop) (TRUE (control ?x)) (other ?player ?x))
```

Instead of listing it extensively as in the preceding excerpt, the `other` property can itself be deduced via a theorem, using `DISTINCT`:

```
(<= (other ?x ?y) (role ?x) (role ?y) (DISTINCT ?x ?y))
```

2.4 Modifying game descriptions

Once one has the description of game in the GDL, is is easy to modify the description to obtain a variant of this game.

Inverting the goals in a game is easy, but leads to a complete change in the strategy. For example, replacing in the Tic Tac Toe description the lines:

```
(<= (goal ?player 100) (role ?player) (line ?player))
(<= (goal ?player 0) (other ?player ?x) (line ?x))
```

by:

```
(<= (goal ?player 0) (role ?player) (line ?player))

(<= (goal ?player 100) (other ?player ?x) (line ?x))
```

one changes the game to a version were the goal is to *not* complete a line.

It is possible to have many boards of the same game and let the player choose the board it will play on. In the simple *snake* single user game, the player pilots a snake on a 5×5 board, the goal being to pass once and only once in every cell of the board. The 2008 competition, included a variant of this game with 16 boards. The player has to choose one board and play on this board. This way, while the structure of the game is identical to the one board version, the branching factor is multiplied by 16 (figure 1).

The complete description of checkers in the GDL is about 500 lines of GDL (plus 200 for defining the arithmetic for the move counter). By simply adding the property

```
(adjacent col8 col1)
```

one obtain a game of checkers on a cylindric board, which significantly modify the strategy.

2.5 GDL extensions

Mickael Thielscher has recently proposed an extension to the GDL that allows the description of games incoporating random elements (like backgammon) and where the information is not completely shared between the players (like most of card games, incuding poker) [6].

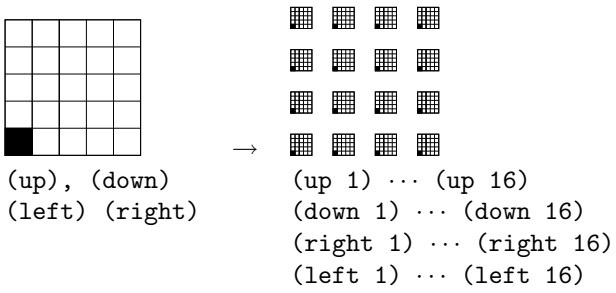

Figure 1: Instead of having to play a simple game of *Snake*, the player has to choose one of the 16 boards where she will play the next move of the same *Snake* game. The branching factor is multiplied by 16.

3 Ary, our GGP program

In this section, we present Ary, our GGP program.

3.1 Using Prolog for interpreting GDL

Ary translates the rules of the game from the GDL into Prolog, and transmits them to a Prolog interpreter. This interpreter is then used as a inference engine for

- generating legal moves
- applying moves
- detecting end of game
- determining the score for each player

3.2 Exploration: Monte Carlo, UCT

In the 2007 qualifications, Ary used a simple Monte-Carlo method to explore the game tree. The program excutes random playouts by playing random legal moves until the end of the game. At the end of the thinking time, the move with the best mean reward is played.

This simple method was direct to implement and gave good results in the 2007 qualification where Ary was classed fourth.

Since the final phase of 2007, Ary uses an extension of Monte Carlo named *Monte-Carlo Tree Search* (MCTS). The program builds a tree of explored status, adding one node for each playout. It uses the results of the previous playouts to select a branch to explore and updates the value in the nodes of the tree according to the result of the playout. Ary uses UCT, a method that balances between deepening the search in promising branches and scouting underexplored subtrees [4].

	1	4	3	5	9	2	6	
5		2	1		6	3		8
9	3		8	7	2		1	5
7	6	5		3		9	2	1
	8	1	9		7	6	5	
2	4	9		1		8	7	3
6	2		7	8	1		4	9
1		8	5		4	7		2
	5	7	2	9	3	1	8	

Figure 2: The player has to fill this easy sudoku grid.

CadiaPlayer, winner of the 2007 and 2008 GGP competition, also uses MCTS [1]. Most competitors since the 2009 GGP competition also use some variant of MCTS.

4 Some games of the 2009 competition

In this section, we present some of the games played in the 2009 competition. All these games were new ones, designed by Jim Clune, author of ClunePlayer, the winner of 2005 competition [2].

4.1 A one player game: sudoku

The goal is to Complete a simple Sudoku grid (see figure 2). The reward is proportional to the number of structures (rows, columns, boxes) containing all digits. The problem to solve looks easy, but the branching factor is huge if one does not detect that the order of moves is irrelevant. The best program scored 54 points (of 100 possible) on this game.

4.2 Two players, turn taking

The game *Capture the king* was a variant of chess (no castling, no *en passant*, no promotion). The goal is to *capture* the adverse king.

The *Pawn whopping* is played on an chess board, with only the pawns at their usual place. The goal is to be the first to promote one of its pawns.

4.3 Many players, Simultaneous play

The *Four Way Battle* is played on a rectangular board. Each player owns two kings (moving like a king in chess) and four pawns (moving only orthogonally.

At each step, every player has the choice between moving or defending one of its pieces. When a player try to capture a defended piece, the attacker disappears. When two players move a piece on the same square, both disappear.

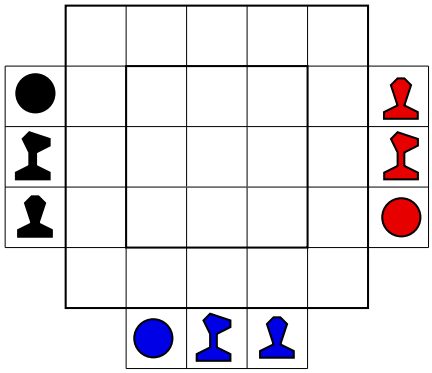

Figure 3: The starting position for *ttcc4* with three players

.

When one capture another (adverse) piece, 10 points are gained. The game stops when one of the players has gained 100 points of after a fixed number of steps.

The *smallest* is a variant of *mediocrity*: five players name a figure between 1 and 10. The one naming the smallest *unique* figure wins ten points. This step is repeated until one of the players has gained 100 points. The organisation provided a *random* player as the fifth player, that won all its matches.

4.4 A game of the 2008 qualifications : TTCC4

The game named *Ttcc4* was conceived by Eric Schkufza for the 2008 qualifications, by mixing together *Chess*, *Checkers*, *Connect 4* and *Tic tac toe*.

The game is played on a 5×5 board (see figure 3). Each player has three pieces. One of the pieces moves like a king of *checkers*, moving diagonally and capturing opponent pieces bu jumping over them. The other one moves and capture like a knight of *Chess*. The third one is reminiscent of the pawn of *Chess*, moving orthogonally and capturing diagonally. On its turn, each player choose between moving one of his pieces and dropping a token on a colunm that descend into that column until encoutering an obstacle, as in *Connect 4* ; it then stays in that position until captured. When a piece is captured, it is actually resent to its starting position. The goal of the game is to align three of its pieces en the central 3×3 square, like in *Tic Tac Toe*.

5 Conclusion

We have presented the context of the General Game Playing, and the Game Description Language that is used to describe the games. We have sketched the structure of our program Ary that won the 2009 and 2010 competition, and presented examples of games used in these competitions.

References

[1] Yngvi Björnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, 2009.

[2] James Clune. Heuristic evaluation functions for general game playing. In *AAAI*, pages 1134–1139, 2007.

[3] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[4] L. Kocsis and C. Szepesvàri. Bandit based Monte-Carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[5] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General game playing: Game description language specification. Technical report, Stanford University, 2006.

[6] Michael Thielscher. A general game description language for incomplete information games. In *AAAI*, 2010.