Program Generation for Firms Simulations in Competitive Environments

Tristan Cazenave
LIP6
Case 169
Université Pierre et Marie Curie
4, place Jussieu, 75252 Paris Cedex 05, France
Tristan.Cazenave@lip6.fr

Abstract

In a competitive environment, the results of a firm strongly depend on the behaviors of the other firms. Therefore, the valuation of a firm must take into account its competitors. We propose a methodology to automatically create a program to assess the potential of a firm given a particular model of the firms. The rules forecasting the future behavior of the firm are generated under the assumption of the strongest possible resistance of its competitors. These rules can be used to evaluate the future minimal performance of the firm.

Keywords

Metaprogramming, Competition, Valuation of a Firm.

1 Introduction

Introspect [Cazenave 1996b] is a system that automatically create programs that enable to achieve some goals, given the definitions of the goals and a theory of the domain in which the goals apply. Introspect has been applied to the management of a firm [Cazenave 1996a]. It is given a program that models a firm, and some goals related to the good management of a firm (essentially have a positive cash and have a good return on investment). Using a metaprogram it automatically creates program that enable to find rapidly the good decision to achieve the defined goals under the assumption that the model is valid. Our current model is used to teach management [Alia 1992], but our approach is general and can be applied to other models. This work is a part of a more general research on the usefulness of metaknowledge (knowledge about knowledge) [Pitrat 1990].

In a first part, we describe how knowledge about the firm is represented. In a second part, we show how rules used by opponent firms to prevent a firm to achieve a goal can be automatically created. Then we explain how the generated program can be used to model the evolution of a firm in a competitive environment.

2 Knowledge representation

In order to generate a program, a metaprogram must be given a declarative domain theory of the market and of the firms. The declarativity of knowledge presents many aspects. The first one is the explicit representation of knowledge, it allows the program to manipulate the knowledge it uses. This aspect is typical of logic programming techniques, as in Prolog programs or in expert systems. The second constraint that declarativity imposes on knowledge representation is that the instructions of a program must be given without the order to execute them. A declarative program is both explicit and given without a defined order [Pitrat 1990]. This second aspect is very important because it facilitates metaprogramming, enabling the unfolding of programs without constraints and the reordering of the conditions . We chose to represent knowledge using predicate logic.

Rule_Cash_1: SellIncome (T2, N3) :- Quantity_Sold (T, N1), Sell_Price (T N2), Delay_of_payment_sell (T1), T2 = T + T1, N3 = N1 * N2.	Rule_Cash _2: Cash (T1, N5) :- Cash (T, N), Sell_Income (T, N1), Quantity_Work (T, N3), Buy_Outcome (T, N4), T1 = T + 1, N5 = N + N1 - N3 - N4.

Table 1

Table 1 gives two examples of rules making of the Monetary level domain theory that compute the cash of a firm.

Cash (0, 5000) Quantity_Sold (0, 10) Quantity_Sold (1, 10) Quantity_Sold (2, 10) Quantity_Sold (3, 10) Quantity_Sold (4, 15) Quantity_Sold (5, 15) Sell_Price (0, 170) Sell_Price (1, 170) Sell_Price (2, 170) Sell_Price (3, 170) Sell_Price (4, 160) Sell_Price (5, 160)	Quantity_Products_Bought (0, 10) Quantity_Products_Bought (1, 10) Quantity_Products_Bought (2, 10) Quantity_Products_Bought (3, 10) Quantity_Products_Bought (4, 15) Quantity_Products_Bought (5, 15) Price_Product (0, 70) Price_Product (1, 70) Price_Product (2, 70) Price_Product (3, 70) Price_Product (4, 70) Price_Product (5, 70) Quantity_Work (0, 700)	Quantity_Work (1, 700) Quantity_Work (2, 700) Quantity_Work (3, 700) Quantity_Work (4, 1050) Quantity_Work (5, 1050) Quantity_Sold (0, 10) Quantity_Sold (1, 10) Quantity_Sold (2, 10) Quantity_Sold (3, 10) Quantity_Sold (4, 15) Quantity_Sold (5, 15) Delay_of_payment_buy (3) Begin_activity (0)

Table 2

Table 2 gives an example of a working memory for the monetary level. Working memories contains only predicates and constants. Whereas rules can also contain metapredicates and variables.

3 Generating programs to prevent from achieving a goal

The use of metaprograms to generate programs to prevent from achieving goals has already been described for the game of Go [Cazenave 1998]. In this section, we explain how this kind of metaprograms can be used in the context of firms simulation.

The following metarule enables to create rules concluding on actions of a firm that prevent another firm to have benefits, given rules concluding on how to make benefits.

```
addconclusion (R1, PreventBenefits ( FIRM, AMOUNT )),
```

```
addrule (R1) :-
  rule (R),
  conclusion (R, MakeBenefits ( FIRM, AMOUNT )),
  setofconditions (R, SET2),
  setofmodifyingconditions (SET1, SET2),
  newrule (R1),
  addsetofconclusions (R1, SET1),
  addsetofconditions (R1, SET2).
```

The rules concluding on how to make benefits are also created by Introspect as described in a previous paper [Cazenave 1996a].

The first condition of this metarule, 'rule (R)', selects all the rules and instantiate them in the variable R. The second condition selects among the rules those that conclude on the goal MakeBenefits for the firm denoted by the variable FIRM. The next condition 'setofconditions (R, SET2)' puts into SET2 the set of conditions of the rule R selected. Then the condition 'setofmodifyingconditions (SET1, SET2)' calls a metaprogram specific to the simulation of firms that fills SET1 with a set of actions that prevents the conditions in SET to be achieved. After that, the metarule, creates a new rule R1 and adds SET1 to the set of conclusions of R1 and SET2 to the set of conditions of R1. Eventually, the metarule adds in the conclusion of the rule R1 the predicate about the goal of rule R1 and adds R1 to the set of rules.

Note that rules about achieved goals are used to create rules about action to take, that are themselves used to create rules about actions to prevent from achieving a goal, that are in turn used to create rule about achieved goals (when no preventive action works). This enables the system to incrementally create more and more complex rules, until the size of the created rules is set to the maximum available memory for the generated program.

4 Using the generated programs to forecast

The generated programs have to be used over a long term simulation in order to return valid results. There is often a possibility for a firm to damage its competitors at its own expenses. For example by fixing the price of its manufactured products lower than the real prices, loosing money over a short period of time but preventing the other firms to sell their products. However this strategy cannot be used in the long term, or even if it can be used on a period of time, it may have catastrophic consequences for all the firms. That is why a long term simulation of the competition between the firm gives a better indication on their respective reasonably expected performances than a short term one.

The main advantage of the generated programs is that they reduce the number of reasonable possible actions for each firm, therefore enabling to forecast on a longer period of time.

5 Conclusion

We have described how to generate program for efficiently simulating firms in a competitive environment. The main interest of the generated program is that they enable to forecast efficiently over a long period of time the behavior of a firm in a simulated competitive environment. Therefore giving insights on the performance of the firm on the market and on its valuation.

References

Alia C. (1992). *Conception et réalisation d'un modèle didactique d'enseignement de la gestion en milieu professionnel*. Ph.D. Thesis, Montpellier II University, 1994.

Cazenave T. (1996a). *Learning to Manage a Firm.* Proceedings of the International Conference on Industrial Engineering Applications and Practice, Houston, 1996.

Cazenave T. (1996b). *Système d'apprentissage par auto-observation. Application au jeu de Go.* Thèse de l'Université Paris 6, Décembre 1996.

Cazenave T. (1998). *Metaprogramming Forced Moves.* Proceedings ECAI'98.

Pitrat J. (1990). *Métaconnaissances*. Hermès, 1990.