Constraint Model-based Exploration of Simulation Trajectories in a MABS Model

Oswaldo Terán[*†], Bruce Edmonds[*]

[*]Centre for Policy Modelling, Manchester Metropolitan University,
Aytoun Building, Aytoun Street, Manchester, M1 3GH, UK.
Tel. +44 161 247 6478 *Fax*. +44 161 247 6802
b.edmonds@mmu.ac.uk

[†]Department of Operation Research and Centre for Simulation
and Modelling, Universidad de Los Andes. Venezuela
Tel. +58 274 240 2879
oteran@ula.ve

Abstract. This paper presents a method used for systematically investigating the content of a simulation model [see 17-18] in terms of constraint logic programming, more specifically as a forward chaining (model-based) constraint exploration of simulation trajectories. Possible applications of this exploration in MAS-based modelling are: proving tendencies in a fragment of the theory of a simulation model, to tease out what affects the envelope of a tendency, and more exhaustive scenario analysis than traditional ones. The proposed exploration allows for all simulation trajectories (possible worlds) associated to and constraint by a range of parameters of the simulation model and a range of choices of the agents. The characteristics and advantages of SDML, a declarative MAS-builder simulation language, for doing this exploration are explained. It is verified that the exploration is *coNP-complete*. This paper represents an effort in bringing closer the constraint logic community and the simulation community.

1 Introduction

There is a call for relating the content of a simulation with a theory the simulation model corresponds to, in someway. Discussions about this subject can be found, for instance, in the list of the social simulation community (http://www.jiscmail.ac.uk/lists/SIMSOC.html). It is of interest for the social simulation community the analysis of 'emergent' tendencies observed in the simulation output, more specifically in simulation trajectories. Common methods for studying simulation outputs are scenario analysis and Monte Carlo Techniques, or a combination of both (*e.g.*, Carley's Virtual Experiments). Recently, researchers have been wandering about proving aspects in the content of a simulation model, a task traditional methods cannot support. More precisely, there is a need, specially in those works related with elaborating or testing theories [4, 5, 14], for studying and proving (emergent) tendencies in the simulation of social systems.

On the other hand, Constraint Logic Programming (CLP) appeared as the first answer to the need for a declarative programming with a more flexible manipulation of the semantic than traditional Logic Programming (LP) and forward chaining systems. The

idea is to allow a semantic driven search using backward inference, initially using Prolog as a platform and as a programming style [8].

A second answer, namely Constraint Forward Chaining (CFC), came from Rule Based Forward Chaining methods. Examples are Constraint Handling Rules (CHR and its improved version CHR^{v}; [1]), Constraint Rule Base Programming (CRP) [9], Satchmo and CPUHR-tableux calculus [2-3]. Among the advantages of these systems over CLP there are: allowing alternative logical extensions via split and backtracking (*e.g.*, Satchmo, CHR^{v}, CRP), introduction of user defined constraints (*e.g.*, CHR) and Meta and Higher-Order reasoning via re-writing of rules (*e. g.*, Satchmo).

Usually CLP systems are aimed at looking for a proof (like in LP) while CFC systems are intended at finding a "logical model"[1] satisfying certain conditions. In the former situation, a conclusion is based (in some sense) in a whole exploration of the space of possibilities while in latter situation only one among the possible solutions is searched for (one logical extension). However, this is not always the case, particularly in this paper the interest is in a CFC exploration of all possible logical extensions – more specifically, in a constraint exploration of simulation trajectories where a trajectory would correspond to a logical extension.

Consequently, CLP and especially CFC hold a potential for exploring the content of a simulation model due to their flexibility, expressiveness and the correspondence of a logical extension to a simulation trajectory allowing formalisations and a promise for formal proofs. Thus, declarative programs open new ways for exploring the content of a simulation model.

Constraint logic programming would allow a systematic and controlled constraint search for alternative trajectories in a simulation, authorising stronger conclusions and the possibility of a more assertive and fruitful comparison between the content of a simulation model and theoretical descriptions of the modelled phenomena than traditional methods.

In a previous paper [18], a hierarchy of computational architectures for searching for and proving tendencies in Multi-Agent Based Simulation (MABS) models has been proposed. The first architecture, that at the higher level, consists of the MABS model where tendencies will be searched for by the modeller itself. After a tendency is found, at a second architectural level, a constraint logic model proof of the envelope of the simulation trajectory is proposed. In [18-19] a computational technique for doing this proof efficiently is implemented and illustrate by using an example. And, at a third architectural level, a more general proof of the envelope of the found tendency would be implemented by exploring a wider fragment of the simulation theory by using a syntactic driven search.

The *aim of the present paper* is to describe the second architectural level and its computational complexity in terms of constraint logic programming, contributing in bringing closer the simulation and the logic programming communities.

The research of the authors is concerned with modelling social and organizational systems, being of especial interest the observation of the dynamics of the simulation of the model and particularly the identification and proof of tendencies, theorems, or properties of the system, valid under a range of parameters of the model and a range of choices of the agents. Their models are implemented and displayed in a declarative simulation

[1] The term "logical model" means model in the logical sense, which is different to the idea of model in modeling and simulation theory. In the terminology used in this paper, a logical model corresponds to a simulation trajectory.

language: SDML: Strictly Declarative Modelling Language; [13]. This language is suited for constraint searches because of its facilities for backward and forward simulation, backtracking, re-writing of rules (by using a meta-agent) and an internal assumption manager allowing certain predefined manipulations of constraints, has been used for experimenting. Because of this, also the main features of SDML for Constraint Logic Programming are reported in this paper.

Thus the novel contribution of this paper in comparison of previous papers [17-18] is the description of the search presented in those papers, and of SDML, in terms of constraint logic programming, as well as the presentation of the computational complexity of such a search.

The paper is organised as follows. First, in section 2, the proposed constraint exploration of simulation trajectories and its usefulness for analysing the content of a simulation model are described. Then, in section 3, the main features of SDML for a constraint search are described. Following, in section 4, it is verified that the proposed exploration is *coNP-complete*. Finally, in section 5, some conclusions are drawn.

2 Constraint Model-based Exploration of Simulation Trajectories in a MABS Model

2.1 Logical Model-Constrained Exploration of Simulation Trajectories

In [17-18] to study the *emergence of tendencies* in a simulation via exploring a subspace of the set of trajectories in the 'theory' of a simulation model has been proposed. A logical model-based constraint search where constraints stand for selected parameters and choices was implemented. Such exploration allows to explore that fragment of the simulation theory constrained by the selected range of parameters and choices. The resulting conclusions and proofs are valid over the covered fragment of the theory and, under appropriate justifications, they could be extrapolated to the bigger part of the simulation theory.

We understand a *simulation trajectory* as a logical model embedded in a simulation program (a 'possible world' in semantic terms), involving trajectories of entities (*e.g.*, agents) inside the simulation and, hence, different from trajectories of these entities. It is a cross-product of all settings of the structure of the simulation model and all processes (*e.g.*, agents' choices) *into one path* through a high-dimensional space (see Figure 1. In this figure, each possible path represents a simulation trajectory). It is assumed that the transition function of the processes, such as the agent behaviour or the simulation model behaviour, are nondeterministic.

The character of the search implemented in our models [17-18] has been predominantly logical model, constraint, and forward-chaining oriented, as well as clausal ordered. A logical model is generated for each combination of parameters and choices, and for a finite iteration number, *n*. Given a combination of parameters and choices a deterministic transition function may be defined to generate the logical model by iterating from the initial state until reaching the iteration number, *n*.

In the suggested exploration, first, each combination of parameters provides a different structure of the simulation model (see Figure 2). Following, 'paths' representing trajectories are generated for each structure. Then, while the simulation is going on,

choices produce branch points where alternative settings for each choice turn out into a different simulation trajectory.

For instance in a Trader-Distributor model (see Figure 3), for a setting of parameters of the model, at each time iteration, branch points might be due to Distributor's choices, each Distributor can choose among the different Traders. At any time step, Trader-1 could choose among Producer-1, Producer-2, etc., each choice defining a different simulation trajectory from this time instant on. One trajectory is explored at once but as soon as a simulation trajectory has been explored completely (for all given time steps) the program backtracks and takes on another trajectory from a branch point. In a model of this kind,

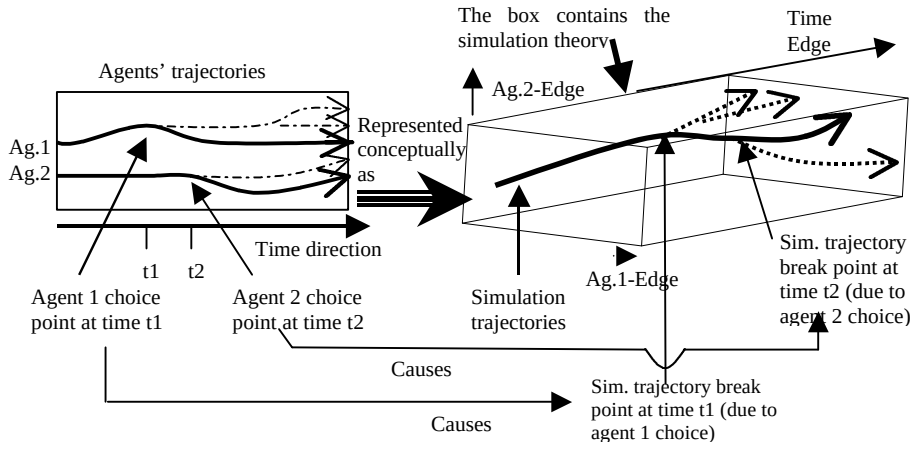

Figure 1. Simulation theory in terms of the simulation trajectories, and of these in terms of agents' choices (for a single parameter-setting and two agents)

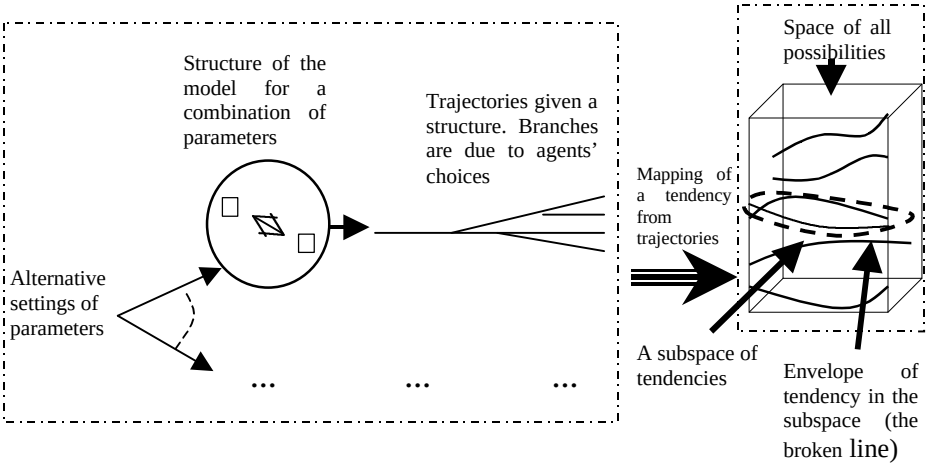

Figure 2. A model constraint-based exploration of the content of a simulation model

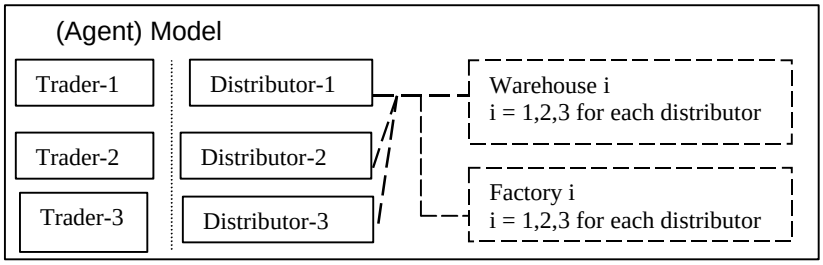

Figure 3. An example of interacting agents

implemented in [19], the agents' main tasks are: for traders, price-setting, price-imitating, and sale-setting; and, for distributors, order-setting. Parameters might be defined by alternative size of warehouses, for instance.

2.2 Enveloping Tendencies

Usually, a *tendency* is a patter observed in one or in a combination of descriptive variables of a simulation model, that is, a tendency is a distinction that persist over time and have some sense for a modeller. For instance, a tendency in a Trader-Distributor model would be prices tending to be closer and closer over time in the sense that the difference between the higher price and the lower price (taken at each time instant) *over time* decreases. A tendency might also be a qualitative distinction, for example, in a cellular automata model, particular arrangements of neighbours. In social simulation, patterns of particular interest are tendencies representing properties of the model resembling properties of the target system.

The idea of enveloping a tendency is to enclose or encircle in some sense several instances of the simulation output, by using a language, a mathematical description of a certain contour. Enveloping represents an alternative to statistical summaries of such a tendency. Given that in a simulation trajectory the data appears discontinuously, discretely, as a sequence of values over time for each descriptive variable (we do not have continual mathematical descriptions of variables allowing to get a value for each variable at every real value of time the trajectory is defined for), it does not seem convenient to use the strong concept of envelope managed in mathematics. Rather, an envelope will be chosen considering the trade-off between practical usefulness (for a modeller) and precision.

By precision we mean how close the concept is to the ideal mathematical notion of a tangent curve/surface. For example, in mathematics given a family of functions (which might correspond to several runs of a simulation output *Y*, got, for example by varying some parameters), let us say $y_j(t)$, $j = 1,2,.,.k$, an envelope of this family will be a curve *E* being tangent at each point to a member of that family (let us say it is tangent to y_j *at* $(t_i, y_j(t_i))$. Obviously, in simulation only as a casualty the instances of a simulation output could conform a family of curves having such a kind of tangent - so we have to use a more relaxed concept of envelope.

Consider the case of *enveloping a single simulation output, Y*. Each *trajectory* will generate a sequence of real values over time, *Y*. Calling y_{ij} the output value at time instant *i* for trajectory *j*, an envelope might consist of two sequences of values over time: E_{upper}

and E_{lower}, which in some sense cover all trajectories. The value of E_{upper} at time instant i must be greater than or equal to y_{ij} for all j, and E_{lower} at time instant i must be lower than or equal to y_{ij} for all j. That is, the envelope would be given by two sequences of values over time, where for each time instant all values generated by the simulation trajectories are enclosed by the two values given by these two value sets. More precisely, if the outputs y_{ij} are given for the simulation trajectories $j = 1,\ldots,k$, and for the time instants $t_i = 1,2,\ldots l$, then the envelope of interest at t_i might be defined by the two values: $E_{upper\,,\,i} = max_j\,(y_{ij})$ and $E_{lower\,,\,i} = min_j\,(y_{ij})$.

To illustrate the practical use of analysing the simulation outputs by enveloping the output think about the simulation of a chaotic system, where the envelope might help in defining the area where a chaotic attractor is placed.

As said above, enveloping a tendency of interest is proposed for analyzing simulation outputs, as an alternative to central statistical measures. A constraint exploration of simulation trajectories is useful to tease out what affects the envelope of a tendency, *i.e.*, to find out the factors affecting the shape, the size or any other aspect of the envelope of a certain tendency observed in the content of a simulation model. The range of parameters and choices would be selected in accordance to this goal.

In Figure 2, *mapping* from the setting of trajectories, as given by the alternative arrangement of parameters and agents' choices, to certain knowledge (maybe properties) about the tendency is represented ((trajectories for a range of parameters and choices)→ (knowledge of the tendency)).

2.3 Proving Tendencies

Probably the most remarkable application of the constraint exploration of simulation trajectories is proving tendencies in the content of a simulation model. The idea is to generalise about tendencies going from the observation of individual trajectories to observation of a group of trajectories generated for certain parameters and choices. In particular, it is intended to know if a certain tendency is necessary or contingent in the explored trajectories.

If a modeller suspects of a tendency, this might be introduced in the simulation as a hypothesis in terms of an envelope and, if this expression is not contradicted in all trajectories displayed for a range of choices of the agents, a range of parameters of the model, and a limited number of time steps, then the envelope is a theorem for the content of the model constrained by the range of parameters, choices and time steps defining such an exploration.

This exhaustive constraint-based search over a range of possible trajectories makes it possible to establish the necessity of postulated emergent tendencies. In [17-19], following a procedure similar to that used in theorem-proving [6,10,12,21,22], an emergent tendency (a theorem) is prearranged in the form of a negative constraint and then an automatic search is performed over all possible trajectories constrained by a subset of parameterisations of the model, a range of agents' choices, and a certain iteration number. Tendencies are shown to be necessary for a finite number of iterations, a range of parameterisations of the model, and a range of non-deterministic choices of the agents, by, first, finding a possible trajectory without the negative constraint to show the rules are consistent, and, second, showing that all possible trajectories violate the negation of the hypothetical tendency (this is added as a further constraint).

2.4 Scenario Analysis

Obviously, the proposed exploration of simulation trajectories would also be useful for scenario analysis. In this case, the interest is not in proving but more in investigating likely tendencies in pertinent and possible trajectories. For doing this, the range of parameters of the model and choices of the agents to be examined would be defined by expert domains [7].

2.5 Exploration of Simulation Trajectories and Constraint Forward Chaining

The relation between the described search and CFC is obvious. Constraints are defined trough a range of parameters, a range of choices, and a subset of time steps. The described search is forward oriented. At the beginning of the simulation the model is placed in an initial state defined for the initial time step, next a state transition is generated bringing the model into a new state at the following time step defined increasingly and sequentially. If a branch point appears, one among the possible simulation trajectories or paths is chosen for a first exploration. Once all time steps have been assumed in a path and the model has been brought trough all corresponding states (making a certain assumption at each branch point found), *i.e.*, a trajectory is generated, the program backtracks and takes on another simulation trajectory from the previous closer branch point.

Understanding the search in these terms, associating it with CFC, brings to social simulators all potential CFC offers, *e.g.*, flexibility, expressiveness, potential for formalizations and formal proofs, opening new ways for exploring the content of a simulation model. The presented search already allows a systematic and controlled constraint search for alternative trajectories in a simulation, authorising stronger conclusions and the possibility of a more assertive and fruitful comparison between the content of a simulation model and theoretical descriptions of the modelled phenomena than traditional methods. Known traditional methods are Scenario Analysis and Monte Carlo Techniques. The former can be implemented using a constraint search, as explained above, but this sort of search does not explore all possibilities for a range of parameters and choices, and thus, the allowed conclusions are weaker, less general, than those allowed by when proving the envelope of tendencies, as this last is based in a more exhaustive search. Among the potential ways of exploring the content of a simulation model a syntantic driven search has been suggested in previous papers [17-18], which represents an alternative to the semantic driven search proposed in this paper.

3 Towards the Implementation of a Suitable Platform for a Constraint Envelope of Trajectories Using SDML

The experiments reported in [17-19] were developed in the MAS builder SDML (Strictly Declarative Modelling Language; [13]). As a source of comparisons the model was also programmed in the Theorem Prover OTTER [12]. In [17-18] a MABS model is reduced via a sort of *unencapsulation* of the hierarchy of agents into an appropriate architecture. In the original MABS architecture, each agent has its own rulebase (RB) and database (DB). In the proposed architecture, the hierarchy of agents disappear and is replaced by a single

DB-RB. This section does not aims to explain SDML in detail, but rather to offer an overview of those aspects of SDML more related to constraint programming, for more details about SDM see [13].

3.1 SDML Characteristics and Features for Constraint Exploration of Simulation Trajectories

- Good underlying logical properties. SDML's logic corresponds to the Strongly Grounded Autoepistemic Logic (SGAL) described by Kurt Konolige [11, 15].
- Its backtracking procedure facilitates the exploration of alternative trajectories via the splitting of simulation paths according to agents' choices and parameters of the model.
- The assumptions manager tracks the use of assumptions. Assumptions result, for instance, from agent's choices.
- A good collection of useful primitives relevant to, for instance, social simulation.
- The type meta-agent used here *not* as an agent *per se* but as a module to 'compile' and re-write rules into an efficient form.

3.3 Syntax of Agent Rules (see [13])

A rule in SDML has *antecedents* and *consequent*s. It may be read declaratively as stating that, if the antecedents are true, then the consequents are also true. Both antecedents and consequents consist of *clause*s, which may be conjoined, disjoined, or negated in the case of antecedents, and conjoined in the case of consequents. A clause consists of a *functor* and a number of *argument*s. Typically the arguments in rules are variables which are unified with ground terms and thus instantiated when rules fire. Where the antecedents of a rule contain variables, the rule specifies that the consequents are true for all bindings of the variables such that the antecedents are true.

The object-oriented features of SDML require that terms used as functors and arguments must be defined as objects. Clause definitions specify a functor name and the number of associated arguments and type restrictions for each. Clauses are defined to be either forward- or backward- chaining.

Corresponding to forward- and backward-chaining clauses in SDML are forward- and backward-chaining rules which have these clauses in their consequents. Forward-chaining rules are primary, and agents may have a number of forward-chaining rulebases corresponding to different time levels specified by the user. When forward-chaining rules fire, their consequents are asserted to a database, there being a database corresponding to each such rulebase. Each agent has a single rulebase for each backward-chaining clause definition. Backward-chaining rules are effectively procedures called by the antecedents of forward-chaining rules. Important uses of backward-chaining rules are list-processing procedures similar to those in Prolog and for substituting a mnemonic clause for more cumbersome but related groups of clauses in forward-chaining rules.

SDML's rules are fired using forward chaining by retrieving clauses from databases in order to determine the bindings of variables such that the antecedents are true, substituting those bindings in the consequents, and asserting the resulting clauses to databases. The order in which rules are fired is not specified by the user, but is determined automatically on efficiency grounds. SDML orders the rules according to dependencies among them,

dividing the rulebase into partitions whereby rules in a later partition are dependent upon, rather than determiners of, rules in an earlier one.

3.2 Internal Manipulation of Constraints in SDML

In SDML, a partition is a grouping of rules in accordance to their dependencies. Dependencies among rules in different partitions determine dependencies among partitions. Rules in a partition do not have dependencies on the subsequent partitions. Assumptions are made for each partition in accordance to agents' choices in such a partition. SDML's assumption manipulator is a sort of Truth Maintenance System (TMS) for each partition (see Figure 4). For instance, shall a variable get a certain value deduced under two different assumptions, then a disjunction of the two original assumptions is placed for this datum in the database.

Once a contradiction (*e.g.*, the predicate *false* in the consequent of a rule) is found the system backtracks and a new choice is made in that partition. When all choices have been unsuccessfully tested in a partition the system backtracks to a previous partition to make a different choice.

As said above, "meta-module" or "meta" is a module to write rules in the system at the beginning of the simulation. It allows semantic-driven manipulation increasing flexibility of the exploration, *e.g.*, to write rules referring explicitly to instances (individual occurrence of a type) rather than to types of instances. Such a manipulation will be very useful both, for driving the search and for making it efficient in terms of computational time.

In SDML, choices are introduced via *built in* predicates. For example, the primitive *randomChoice* allows choosing randomly from several alternative values. Each choice will define a different simulation path labelled with a certain assumption. Another interesting primitive is *notInferred*, one of the primitives allowing non-monotonic reasoning. *NotInferred* allows generation of data when a certain fact is not present in the database, *e.g. notInferred b, implies c*, will put c in the database if *b* is not in the database. If this value, *b*, is written later during the search in the database, then the assumption becomes false. In case the rule yielding *b* is in a different partition from the one where the

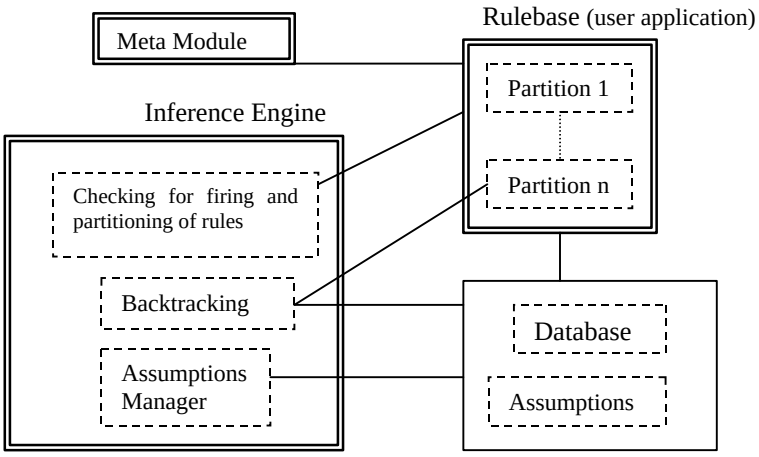

Figure 4. Overview of SDML's framework.

assumption was made, then *c* and any other data supported by the assumption is withdrawn via backtracking to the partition where the assumption was made.

3.3 Comparing SDML with Satchmo and Other Constraint Logic Systems

While some of Satchmo's and other constraint programming languages' facilities are similar to SDML's, for example, *backtracking* and the *false* predicate, other facilities, *e.g.*, some built-in facilities for manipulation of constraints, are not present in SDML. Among these, we have reasoning about terms in CLP(X) or consistence techniques to prune the range of trajectories in other CLP systems [8]. Instead, SDML allows facilities to introduce alternative values for some manipulated entities (*e.g.*, predicates, clauses, integer variables) which can be used as *constraints* (clauses for choosing, *e.g.*, *randomChoice*) as well as a *meta module* able to reason about terms or rules. Because of all this, SDML is able to control the manipulation of constraints flexibly and transparently for the user.

4 Determining the Complexity of a Constraint Model-based Proof of (the Envelope of) a Tendency

The *aim* of this section is to demonstrate that the exploration of trajectories proposed in the previous sections applied over an infinite (theoretically) number of simulation iterations is *coNP-complete* [16, 20]. To make clearer the exposition, the simulation exploration subject of this paper will be called the *target problem*. As is usual for this sort of verification, two steps are followed: First, it will be proved that the target problem is in *coNP* by expressing it as a binary tree of depth *n.* Second, it will be proved that the problem is also *coNP-complete* by translating the *validity (of Boolean expressions) problem*, a typical *coNP-complete* problem, into the target problem.

For the first part of the proof the aim is to form a Boolean quantified expression:

$$\forall x_1 \forall x_2 \forall x_3 \forall x_4 \forall x_5 \ldots \forall x_{2n-1} \forall x_{2n} (F) \qquad (1)$$

where F is the formula to be evaluated over the variables $x_1 \ldots x_{2n}$ and n is the number of iterations.

The deterministic part in the state transition of the simulation will be called environment's actions, and it will be assumed that it corresponds to the impair variables in *(1)*. It captures changes not associated with *agents' choices* – and basically that part of the simulation where "agents are placed". Consequently, there is only one alternative action for the *impair* variables[2]. The *even* variables correspond to the agents' choices (which are going to be called *agents' actions*). More precisely, for iteration i, $i = 1, 2, \ldots, n$, there are two subsets of variables: $\{x_{2i-1}\}$ and $\{x_{2i}\}$, where $\{x_{2i-1}\}$ is used to represent the environment actions and $\{x_{2i}\}$ stands for the agent's actions. Thus, a whole simulation path or *simulation trajectory is represented* by a concatenation of branches, where each branch corresponds to a unique assignment of values to each variable in the whole set $\{x_i\}$.

[2] Environment's actions are assumed deterministic. The results of this paper are easily extendible to the case the environment's actions are non-deterministic.

Finally, *F* will be the question: whether the searched *tendency* has occurred in a simulation trajectory. The whole expression *(1)* is true if for all possible assignments of values to the variables the tendency occurs. As each particular assignment of values to the whole set of quantified variables corresponds to a simulation trajectory, the proof is successful if this expression is valid for all possible values the quantified variables can take! (*e.g.*, for all possible agents' choices[3]).

To check if the proof is successful, a Boolean circuit, where an *AND* gate stands for the $\forall$ quantifier, can be written. A leaf in this circuit is evaluated to *true* if the tendency is found in the corresponding simulation path and to *false* otherwise. The whole circuit will be *true* if and only if the tendency appears in all simulation paths. Hence, the proof is successful if and only if the circuit is true (*e.g.*, the tendency is found in *all* paths).

These two expressions of the problem (that is, the Boolean circuit and the expression of equation *(1)*) are sufficient to prove that the target problem is *coNP*.

The next task is to prove that the problem is *coNP-complete*. It is easy to see the similarities between the target problem and the validity of a Boolean expression. A Boolean expression is an expression: (a) *x*, where *x* is a Boolean variable (variable that takes the values True and False), (b) $\neg f$,where $\neg$ is the *logical not*, and f is a Boolean expression c) $f_1 \vee f_2$, where f_1 and f_2 are Boolean expressions and $\vee$ is the logical symbol *or* (d) $f_1 \wedge f_2$, where f_1 and f_2 are Boolean expressions and $\wedge$ is the logical symbol *and*. Validity of a Boolean expression f, consists in determining whether the Boolean expression f is valid under all truth assignments (interpretations). If f is not a valid formula, it can be disqualified by exhibiting a truth assignment that does not satisfy it.

We may evaluate a Boolean expression by using a Boolean circuit (see Figure 5). A first variable is chosen from the Boolean expression f and represented by the first node, and then two branches are generated from this node: one the case the variable is given the *false* value and the other for the case the variable is given the *true* value. Then a node is aggregated to each of these branches representing a second selected variable, and two branches from each of these new nodes will represent the *true* and *false* value assignments to this second variable. Imagine that this procedure is continued until all variables in the expression f are considered. A leaf of this tree (*i.e.*, a path) will be evaluated to true if the Boolean expression f is true for the particular (an unique) value assignments the variables have in that path of the Boolean tree. Consequently, the expression f is valid iff all leaves of the Boolean tree have been evaluated to true.This tree corresponds to a target problem, where:

a. The number of iterations, *n*, corresponds to the number of variables in the Boolean expression f,
b. The environment decisions are not considered (do not change the state of the system),
c. The agents have only two nondeterministic choices: true or false (corresponding to the two possible assignment of values a Boolean variable can be given),
d. The question: is the Boolean expression f true for the assignment of values the variables hold in a certain path?, corresponds (in the translation of the validity problem into the target problem) to the question: does the tendency appears in the

[3] And, for all environment's actions, in case of a nondeterministic environment.

corresponding path where agents take decisions in accordance to the assignment of values to the variables?,

e. The tendency appears in a simulation path if the expression f is true for the assignment of values to the variables in accordance to the decisions of the agents in that path.
f. Finally, the expression f is valid iff the tendency appears in all simulation paths.

Thus, the output of the *validity* problem has been reduced to the target problem. The *validity of a* Boolean expression f can be checked simulating the equivalent MABS problem. Therefore, the target problem is *coNP-complete*.

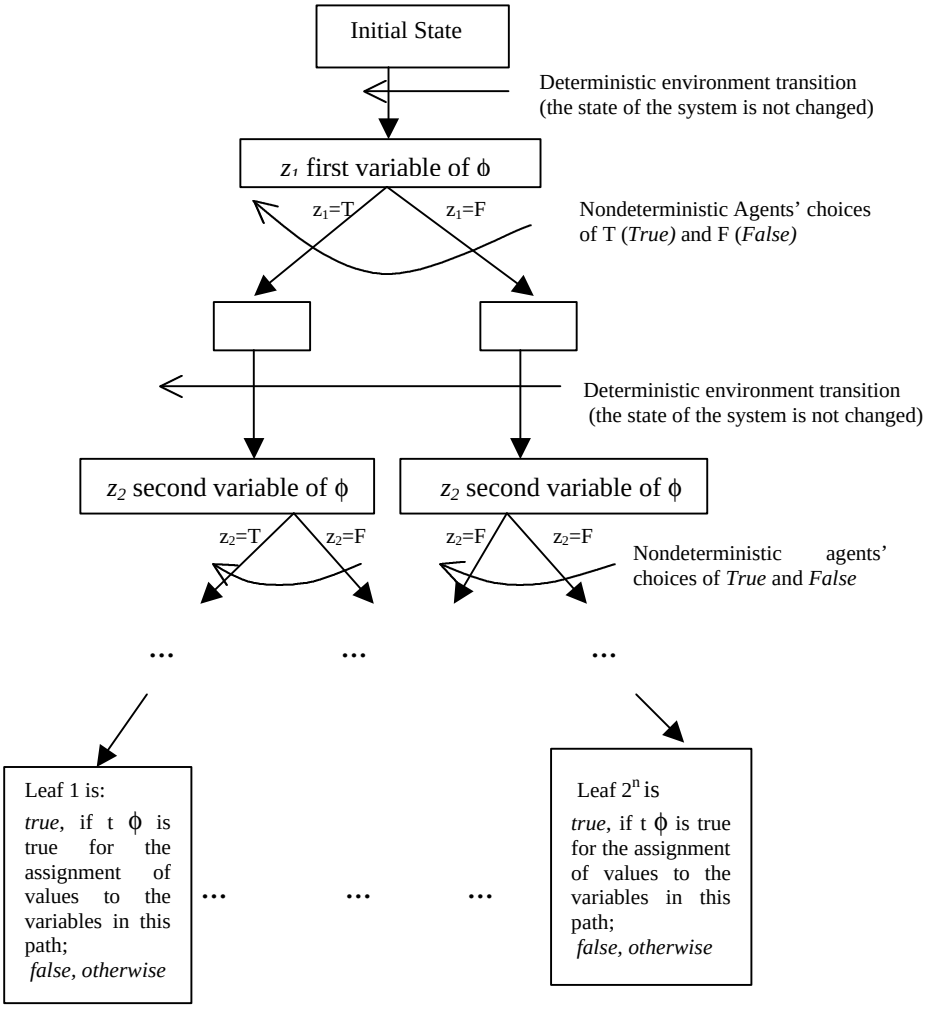

Figure 5. Boolean circuit for the validity problem

5 Conclusions and Further Work

This paper proposes a methodology for studying simulation outputs as a complement to traditional methods dealing with *post-hoc* analysis of simulation trajectories, namely a constrained exploration (of the envelope) of simulation trajectories. In particular a forward chaining semantically constrained generation of trajectories is suggested.

Like in Constraint Logic Programming a constraint generation of simulation trajectories (or extensions, in some way) responds to a need for a systematic and controlled exploration of the content of a model.

The language SDML was used for experimenting. It enjoys many desirable properties for the aimed task, including control of the search and rewriting of rules via a *meta-module*, forward and backward inference, backtracking, and an assumption manager. In particular, the *meta-module* makes the constraints manipulation flexible, transparent and handy for the user. Constrains are context-dependent (over the semantic of the trajectory itself) as the *meta-module* is *able to access the semantics* of the simulation and to set up, in advance, one among the possible combination of agents' choices.

This paper also verifies that the complexity of the suggested constraint model based exploration of simulation trajectories for proving (the envelope of) tendencies in relation to the content of a MABS model is *coNP-complete*.

As explained better in [19], constraint exploration of simulation trajectories brings closer the simulation and the logic programming communities. This paper contributes in making clearer this relationship.

Acknowledgements. The research reported here was funded by the CDCHT (the Council for Scientific, Humanistic and Technological Development) of the Universidad de Los Andes, Venezuela, under project I-524-AA, by CONICIT (the Venezuelan Governmental Organisation for promoting Science), and by the Faculty of Management and Business, Manchester Metropolitan University.

References

1. Abdennadher S., "Constraint Handling Rules: Applications and Extensions", Invited Talk, 2nd International Workshop on Optimization and Simulation of Complex Industrial Systems. Extensions and Applications of Constraint-Logic Programming and 7th International Workshop on Deductive Databases and Logic Programming, Tokyo, Japan, in conjunction with the 12th International Conference on Applications of Prolog, INAP'99.
2. Abdennadher. S. and H. Schütz, "Model Generation with Existentially Quantified Variables and Constraints", Sixth International Conference on Algebraic and Logic Programming, Springer LNCS, 1997.
3. Abdennadher, S., F. Bry, N. Eisinger and T. Geisler, **"**The Theorem Prover Satchmo: Strategies, Heuristics, and Applications - System Description", Journées Francophones de Programmation en Logique, JFPL'95, Dijon, 1995.
4. Axtell, R., R. Axelrod, J. M. Epstein, and M. D. Cohen, "Aligning Simulation Models: A Case Study and Results", *Computational Mathematical Organization Theory*, 1(2), pp. 123-141, 1996.

5. Carley K., M. Prietula, and Z. Lin, “Design Versus Cognition: The Interaction of Agent Cognition and Organizational Design on Organizational Performance”, *Journal of Artificial Societies and Social Simuation* 1(3), 1998 (accessible at: http://www.soc.surrey.ac.uk/JASSS/1/3/4.html).
6. Chiang, C.L., and R. C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, London, UK, 1973.
7. Domingo C., G. Tonella and O. Terán, “Generating Scenarios by Structural Simulation”, in AI, Simulation and Planning High Autonomy Systems, The Univ. of Arizona, pp 331-336, 1996.
8. Frühwirth, T., A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy and M. Wallace. Constraint Logic Programming - An Informal Introduction”, Chapter in Logic Programming in Action (G. Comyn et al., Eds.), Springer LNCS 636, September, 1992.
9. Liu, B., Jaffar J. and Yap R., "Constraint Rule-Based Programming", School of Computing, National University of Singapore, Singapore. Accessible at: `http://www.comp.nus.edu.sg/~joxan/res.html.`
10. Loveland, D. W., *Automated Theorem-proving: A Logical Basis*, North-Holland Pub., Amsterdam, 1978.
11. Konolige, K., “Autoepistemic Logic”, in *Handbook of Logic in Artificial Intelligence and Logic Programming (4)*,Oxford Science Publications, pp. 217-295, 1995.
12. McCune, W., *OTTER 3.0 Reference Manual Guide*, Argonne National Laboratory, Argonne, IL, 1995.
13. Moss, S., H. Gaylard, S. Wallis, B. Edmonds, “SDML: A Multi-Agent Language for Organizational Modelling”, *Computational Mathematical Organization Theory*, 4(1), 43-69, 1998.
14. Moss, S., “Social Simulation Models and Reality: Three Approaches”, *MAB’s 98: Multi-agent Systems and Agent-Based Simulation*, Paris, 1998 (accessible at http://www.cpm.mmu.ac.uk/cpmrep35.html).
15. Moss, S., B. Edmonds, S. Wallis, "Validation and Verification of Computational Models with Multiple Cognitive Agents", Centre for Policy Modelling, 1997, CPM report 97-25, `http://www.cpm.mmu.ac.uk/cpmrep25.html`
16. Papadimitriou, Christos, *Computational Complexity*, Addison-Wesley Publishing Company, California, USA, 1994.
17. Terán Oswaldo, Bruce Edmonds and Steve Wallis, “Mapping the Envelope of Social Simulation Trajectories”, MABS2000 @ ICMAS-2000: The Second Workshop on Multi Agent Based Simulation, Boston, July 9, 2000. Published in: Moss, Scott and Paul Davidsson (Editors), *Multi Agent Based Simulation (MABS-2000), Lecture Notes in Artificial Intelligence, Vol. 1979*, Springer Verlag, Berlin
18. Terán Oswaldo, Bruce Edmonds and Steve Wallis, **“**Determining the Envelope of Emergent Agent Behaviour via Architectural Transformation”, ATAL-2000: The Seventh International Workshop on Agent Theories, Architectures, and Languages, Boston, July 7-9, 2000. Published in: Castelfranchi, C. and Y. Lesperance (Editors), *Intelligent Agents VII. Agent Theories, Architectures, and Languages. Lecture Notes in Artificial Intelligence*, *Vol. 1986*, Springer-Verlag, Berlin.
19. Terán Oswaldo, *Emergent Tendencies in Multi-Agent Based Simulations Using Constraint-Based Methods to Effect Practical Proofs Over Finite Subsets of Simulation Outcomes*, Doctoral Thesis, Centre for Policy Modelling, Manchester Metropolitan University, 2001 (http://papers.ssrn.com/sol3/papers.cfm?abstract_id=292408).
20. Woolridge, Mike “The Computational Complexity of Agent Design Problems”, in *Proceedings Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, Boston, MA, USA, July 10-12, 2000, pp. 341-348.
21. Wos, L., *Automated Reasoning*: *Introduction and Applications*, Prentice Hall, London, 1984.
22. Wos, L., Robinson, G. A., and Carson, D. F., “Efficiency and Completeness of the Set of Support Strategy in Theorem Proving”, J. ACM 14, N° 4, 698-709, 1965.