

Implementing Virtually-Embedded Logical Reasoning in Animat Agents

K.A. Hawick, H.A. James and C.J. Scogings
Institute of Information and Mathematical Sciences
Massey University – Albany
Private Bag 102-904 North Shore, Auckland, New Zealand
Email: {k.a.hawick, h.a.james, c.scogings}@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

14 February 2006

Abstract

Applications of collectively intelligent systems in a physical system can be simulated using a set of animat agents. Conventional programming methods aid expression of the physical spatial system but make it difficult to implement sophisticated reasoning capabilities. Artificial Intelligence languages are well suited to expressing agent reasoning but fare less well in systems of a very large number of agents and are also less well-suited to expressing spatial models. We report on some experiments using a combined approach whereby virtual reasoning engines including Prolog or similar reasoning and planning systems are “virtually-embedded” software components in a spatial simulation system. We report on some performance and scalability results and discuss the associated integration issues for very large-scale simulations where we can investigate emergent behaviours not otherwise attainable.

Keywords: animat agents; physical navigation and reasoning; AI integration.

1 Introduction

Many applications that make use of multi-agent system simulations require large numbers of agents involved. Managing all agent knowledge in such a way that agents can effectively reason about their own knowledge and local findings is not trivial to implement in a software framework, particularly when agents are modelled to have long term “memory” [15] or state.

In this paper we consider some specific tradeoffs involved in using a Prolog [11] or Prolog-like knowledge back-end for our agents and the various different configuration scenarios one can use when setting up such a system. We use the term “Virtually-Embedded Logical Reasoning” to emphasise the fact that we have not embedded within our simulation system any specific commercial implementation of the Prolog engine and that a single such engine can in fact be shared between all agents in our simulation. We discuss this idea in section 5.

While this paper is primarily concerned with the task of efficiently representing knowledge in general multi-agent simulations, the example that we use for discussion throughout this paper is a relatively simple animat [20] model. This model is discussed in section 4, but we believe our ideas are equally applicable to any simulation involving many agents that in-

habit a physical space such as computational physics simulations [5], traffic simulations or sensor network simulations [9].

2 Trails in The Animat World

In figure 1 we illustrate our general architecture for an animat agent. Other animat agents are encountered at locations during the animat’s travels through the environment. Each animat can make use of its own internal clock and memory of where it has been and what it has found so far. This internal state can be used as inputs to reasoning about deciding its subsequent behaviour and hence its output actions.

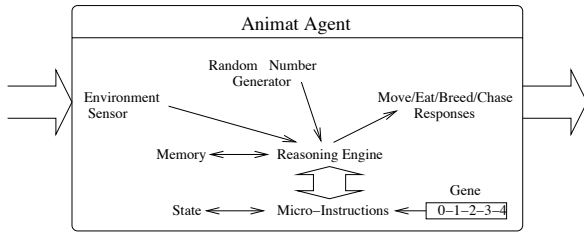


Figure 1: Animat Internal architecture

This notion forms the basis of many robotic control systems and models of simulated life-forms. The reasoning core of a system can be either a specifically coded set of instructions or a logical reasoning engine. In the “many animat” regime a simulation is particularly important as it is not yet economical to experiment with very large numbers of sophisticated hardware robots.

In this paper we are also interested in the interplay between different software coding and implementation styles – namely between mixing declarative knowledge base languages like Prolog with low-level encoded instructions or behaviours that can be acted upon with genetic algorithmic operations. To fix ideas we first consider an animat in a two dimensional grid and we discuss some of the general ideas in terms of an animat “predator-prey” system of “rabbits and foxes”.

Figure 2 shows the trail left by an animat as it moves (possibly randomly) around the world. It has its own internal clock and might make one move or action every unit of time. It has its own frame of reference so its personal origin might be its original position at

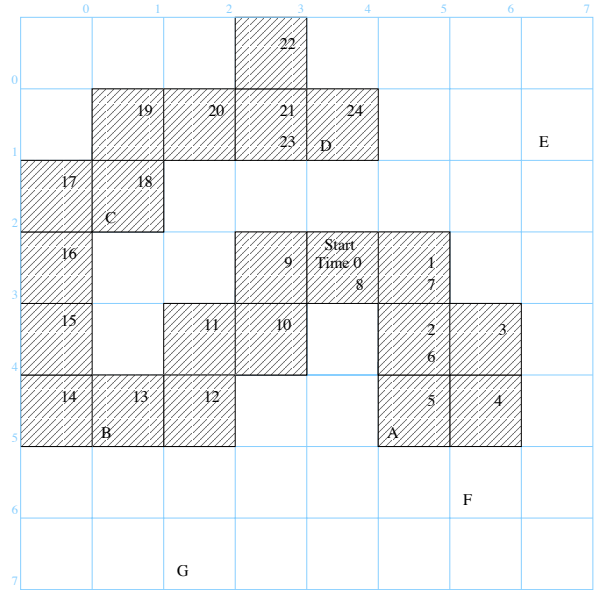


Figure 2: An Animat trail of movements starting at position (4, 3) and encountering various other objects A,B,C,... (other animats) at various times.

coordinates (4, 3). It encounters other objects or animats on its travels. It may record in its own memory that it encountered object *A* at position (x, y) and time t according to its own units and system of reference.

We need to consider the range of interactions inside our model. In our current simulation system, all species have the ability to perceive other animals up to a given radius. Beyond that, nothing can be perceived. One of the major implementation issues we have faced is that of how to efficiently code the potentially $O(n^2)$ checking of animats in the system for proximity. Also, due to our current simulation being “memory-less”, knowledge of any animats that are identified as being nearby, but not close enough to take immediate action (i.e. prey being predated), is lost at the start of the next update cycle. We expect that the introduction of an animat memory will lead to interesting medium range interaction effects, such as prey actively avoiding predators, which will change the observed life-forms we have observed in previous studies [6].

One important question is that of an animat’s event horizon. How long is it reasonable for an animat to expect that the knowledge it has garnered will be

relevant to its current situation? Further treatment of this topic is beyond the scope of this paper; we are considering whether we need a temporal logic for an animat to be able to reason about agents at particular locations at particular times.

3 Animat Internal State

One possible model for the animat internal state uses the BDI (Belief, Desire, Intention) [17] formalism, which also supports specification of agent goals. There are relatively well theories in AI and distributed AI [4] to build upon in this area.

We can view *belief* as a relatively straightforward property that all agents can be imbued with so that animats are able to “see” or “perceive” other animats in the system that are within some radius of perception. In our animat model this information is used to judge whether to move closer to, or further away from, the other animats. The other major contributor to an agent’s beliefs is the external (fixed) environment. Changes in physical conditions, such as topology shape, terrain types, or even chemical concentrations, or signal strengths, can be perceived by appropriate agent sensors in simulations. All these pieces of information must be stored within some kind of knowledge database in order to decide upon an appropriate course of action.

In simple cases, such as searching algorithms, the goal is quite easy to express: stop searching when the target item has been found. However, in a more general case such as when simulating a large number of animats, *desires* (or goals) are possibly the hardest of all properties to express. For example, what does an animat predator (e.g. a fox) want? Does it want to keep its belly full? Does it want to reproduce as often as possible to increase the predator population? Even these two simplistic goals can be seen as conflicting: the more prey a predator eats the fewer there will be when it becomes hungry, and adding to the gene pool by reproducing will have the effect of introducing more competition for food.

We find it convenient to distinguish between long-term goals and short-term goals. For example, the prey in our example system (e.g. a rabbit) may just want to eat or reproduce. These two goals may be considered to be a long-term goal while a short-term goal might be to not be eaten by a predator through

the action of running away. We might argue that it is not likely that when asked, a rabbit would specify the goal of running away as a particularly long-term goal.

An Agent’s *intentions* are effectively just the moves generated. So the animat has “B” as sensor data, “D” as the convolution of encoded behaviours (listed in section 4) and “I” as planned actions/outcome or moves/actions.

One of the most common examples of animat BDI systems is Russell and Norvig’s Wumpus program [19]. This example is popular because it is quite easy to discern the properties of the system and enumerate the moves available to the agent(s) inside the system at each timestep. Essentially the Wumpus system consists of a maze in which a Wumpus (deadly monster) lives, guarding some gold. The aim of the system is to retrieve the gold and return to the exit alive. There are environmental indicators when the agent is near either a pit (a breeze) or the Wumpus (a stench). Some systems also feature a “glimmer” indicator when the agent is near the gold.

When presented with an environmental indicator and information about the current position within the maze, the agent is able to traverse the maze, avoiding pits and the Wumpus, and hopefully locating the gold, before returning to the maze entrance.

It is clear that to be able to do this, the agent must have some sort of memory in which it is able to store knowledge of both the path that has been previously taken, but also the environmental conditions (hazards) that have been identified along the way, which would make one path more appropriate than another.

Finally we draw attention to the fact that inside the Wumpus example neither the pits nor the Wumpus move between locations, although the Wumpus may be killed by (any) agent. Thus, when an agent in the Wumpus world encounters a given percept, that knowledge will be valid forever. This is a condition that does not hold in the more general case that we consider in this paper.

4 Microscopic Rules

We have constructed a predator-prey model [10, 13, 14] consisting of two distinct groups of animats: predators (known as “foxes”) and prey (known as

“rabbits”). The animats exist on a flat grid which expands as they move out from the centre. More than one animat can occupy the same point and new animats (offspring) are placed at the same position as one of the parents. The global environment is controlled by a number of fixed parameters, for example how long a fox takes to get hungry, at what age a rabbit will die of old age, etc. These parameters were established in order to provide a stable model that will continue for several thousand time-steps [13]. An important parameter that directly affects animat behaviour is the perception radius, currently set to 80 pixels for foxes and 20 pixels for rabbits.

The model is initialised by placing a few animats randomly near the origin and each animat is provided with a fixed set of rules to govern its behaviour. At each time-step, each animat takes precisely one action by executing one rule decided upon by local conditions and the priority of the rules. The rules for the two types of animats are given in table 1.

The order of these rules is important. Each rule has a condition and if that condition is true, the rule is executed and any later rules will be ignored. Thus rule 0 has a higher priority than rule 1 and so on. For example, if a fox is not hungry and is adjacent to another fox but not adjacent to a rabbit, than rule 2 (breeding) will be executed and rules 3 and 4 will be ignored. Note that the conditions for rules 0 and 1 will be false in this case. The “move randomly” rule is a default and does not have a condition. Thus if none of the other rules are executed the animat will move randomly.

We have conducted experiments with changing the order of priority of the rules [8]. These experiments do not fall under genetic evolution as every animat is an exact clone of its parents. However, in each experiment, a different rule set was created by changing the rule priorities.

The model typically runs with large numbers of animats, for example 25,000 rabbits and 8,000 foxes by time-step 1000.

The predator prey model outlined above resulted in some fascinating emergent behaviour in which animats formed clusters and spirals. This behaviour is analysed in [6] and the type of clusters that result are classified in [7]. However, this behaviour only occurs when the model contains over 20,000 animats and is run for hundreds of time-steps. For this reason we

are primarily interested in the run-time behaviour of readily accessible Prolog and Prolog-like systems when the knowledge base they have to maintain is of the order of 20,000 unique animats.

5 Reasoning Engines

It would be possible to employ any of the recent Prolog-derived logical planning systems but for simplicity we only describe our ideas in terms of a general Prolog-like system. Prolog itself is a logic programming language built around the notion of first-order predicate calculus: facts, and relationships between different facts can be stated and queries can be posed using those relationships to prove or disprove statements. Prolog was primarily developed for natural language processing but has found its way into most branches of Artificial Intelligence programming for its ability to goal-seek. Most Prolog implementations are based on the Warren Abstract Machine (WAM) [1] which provides a memory architecture and abstract machine.

Due to Prolog’s declarative knowledge-based nature, facts, such as Beliefs, are easily stored in a Prolog database, and providing it is possible to clearly define what is true and, by inference, what is not. Desires are able to be coded as a goal, which evaluates to true if the Prolog engine can satisfy all the constraints imposed by the predicates in the goal definition.

This has meant that simpler problems, such as the Wumpus example, above, can be coded to run under a Prolog engine with relative ease: the facts of the system (the percepts) are easily represented and the goal (find the gold and get out alive) are also easily represented. However, the same is not necessarily true for the types of simulations we are considering: as stated above, how does one define complete success for our animats?

While our current production simulation code is written in C/C++, we have implemented a version in Java for the purpose of graphical interactivity and Prolog experimentation. The approach we are taking is to combine the reasoning engine capabilities of Prolog with the imperative features of Java. This allows us to implement an efficient multi-agent update algorithm (and graphical display) using Java and a knowledge representation back-end with Prolog. We hope, that by combining both programming methodologies,

A fox will:		A rabbit will:	
0	eat a rabbit if adjacent;	0	move away from an adjacent fox;
1	move towards a visible rabbit if hungry;	1	breed if adjacent to another rabbit;
2	breed if adjacent to another fox;	2	move towards another visible rabbit;
3	move towards a visible fox if not hungry;	3	or move randomly.
4	or move randomly.		

Table 1: Rules for two types of animat - rabbits and foxes in our example predator-prey model.

we will be able to only use the most appropriate features of both systems, introducing few inefficiencies.

In the remainder of this section we consider two main approaches to implementing the multi-agent simulation systems under consideration. As in so many other situations, there are various trade-offs to be taken into account: it is more efficient to implement the (imperative) simulations management code in traditional C or C++? Is the 'knowledge' to be represented by the animats of sufficient complexity that it is beneficial to use a logic programming language such as Prolog?

We are primarily interested in free systems that can support upwards of 20,000 unique agents while still providing good run-time performance. Consider the very simple animat trail in figure 2: this animat has already moved 24 times in this snapshot and has encountered four objects in the world. If we only consider the predicates necessary to store knowledge of these objects, and each animat in the 20,000-strong system has encountered four objects, then this requires the Prolog reasoning system to be able to easily handle 80,000 predicates. Of course, this figure does not include the data necessary to represent an animat's internal state nor the animat's precise movement history. We use these figures as ball-park estimates of the predicates in our system for the purposes of testing.

Figure 3 shows two fundamental ways to link the simulation program manager with the prolog engine. In figure 3a) the factual database pertinent to a single animat is loaded and a move is generated. The facts are unloaded and facts relevant for the next animat are loaded into context. This way the prolog engine does not need to be concerned with an overly large database. There is however a latency that arises from the loading and unloading process. Because this approach uses a separate database of facts for each animat, it is a requirement that all facts (and rules)

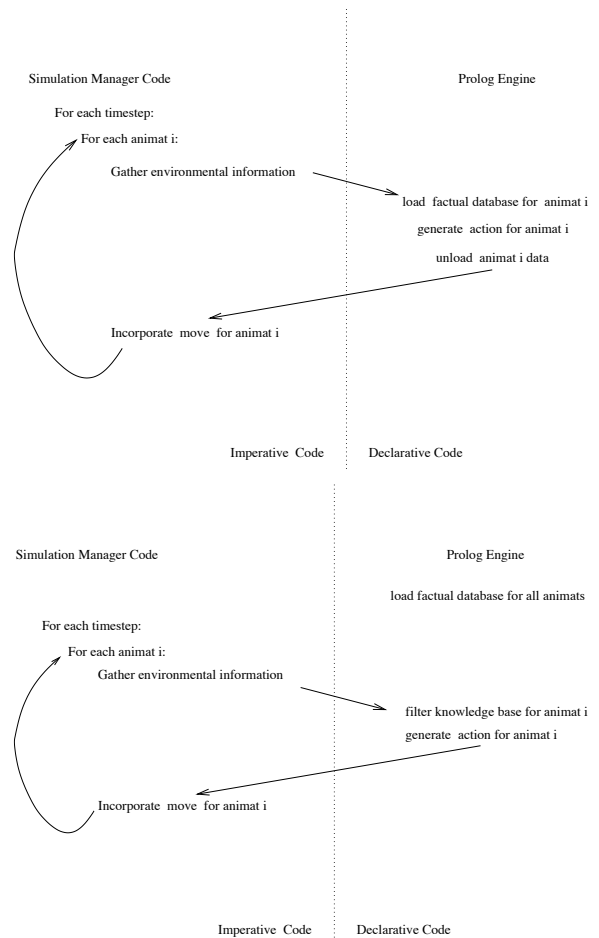


Figure 3: Two models of simulation program architecture. Above) Model 1: Granularity of transfer is just one animat: this requires a context switch between animats. Below) Model 2: Granularity of transfer is the whole database: knowledge is filtered according to the animat under consideration.

common to the system under consideration must be replicated inside each database. Depending on the complexity of the simulation system, this may be a considerable overhead.

Alternatively, one can load all facts into the same database and let the WAM (Prolog engine) deal with the non-scalability. This case is illustrated in figure 3b), where the database is loaded once and before each animat's state can be updated a filter must be imposed to only take into account the facts pertinent to that animat. This approach does have the distinct advantage that the more general logic production rules and system facts do not have to be replicated for each agent, but it does add a slight complexity to the way in which individual animats' knowledge must be stored in order to make them apply only to one agent.

A third approach that we have trialled is to use a system such as CKIProlog [16], which is a Prolog-like library implemented completely in the Java programming language. The library implements a Prolog interpreter as a single first-class object; as such, one has the option of declaring a `Prolog` object inside either each animat *or* inside the simulation, this neatly implements the above two cases.

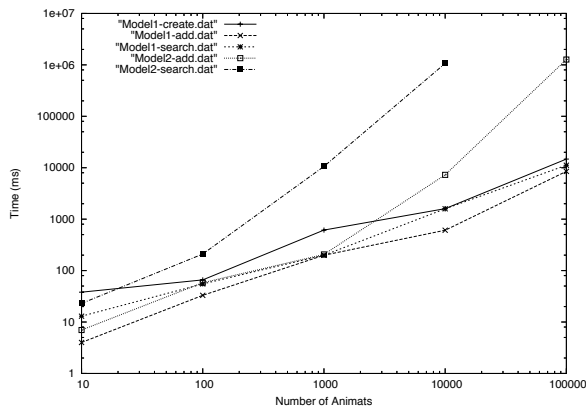


Figure 4: Performance of the CKIProlog Java library when implementing models 1 and 2 in figure 3. The time to create 100,000 Prolog engines, and add to, and query all engines is less than 10 seconds; when compared with a simulation run that takes up to 24 hours for 1000 timesteps this is an acceptable overhead.

Table 2 shows the representative times taken by the CKIProlog Java library to perform various simple op-

erations on our multi-agent simulation systems, and graphed in figure 4. Times reported in the table, and shown on the graph, are in milliseconds, as recorded on a dual-processor G4 Macintosh. It is worth noting that in order to create 100,000 Prolog interpreters (Model 1) it was necessary to increase the default memory size of the Java interpreter, thus making this approach slightly less attractive, although this overhead is slight compared with the time taken when using a single Prolog interpreter.

The overwhelming result from the experiments reported in table 2 is that while there is a considerable memory overhead in maintaining a separate Prolog engine for each animat in our Java implementation, it is significantly faster to assert new facts and search the existing knowledge base for only one animat. The memory saved by only maintaining a single Prolog engine is dwarfed by the amount of time that it takes to do a search when there are even 10,000 predicates in the system.

This agrees with our theory of optimising the most common case: that of having an animat decide what should be its next action based on its history. We are currently experimenting with a further hybrid approach of maintaining a small number of agents in the same Prolog engine so as to cut down on the amount of time spent in establishing the predicates and rules for the agents' physical systems. We anticipate this approach may also be useful when we update our model to pass on information to new 'young' animats created by parent reproduction in the process of model evolution.

In our initial investigations we looked at semi-commercial packages such as Minerva [12] but as the style of programming was quite radically different from the existing C/C++/Java implementations we have decided to stay with a package that provides Prolog as a language extension, rather than as a completely new programming style. In addition, even though Minerva is supposedly implemented in Java, we found the Java APIs to Minerva's Prolog engine rather problematic, at best, and despite Java's language independence, was reluctant to work properly on our Macintosh G4 machine. The performance figures we were able to obtain agree, in general, with those reported in table 2.

number of animats	Prolog Interpreter/Animat			Single Prolog Interpreter		
	create	assert	query	create	assert	query
10	39	4	13	39	7	23
100	66	33	55	39	58	210
1,000	614	199	199	39	206	10582
10,000	1600	609	1587	39	7260	1067635
100,000	14736	8484	11202	39	1270826	

Table 2: Time (ms) to perform simple actions using the CKIProlog Java library: creating Prolog interpreters, adding assertions to the Prolog engine, and querying the Prolog engine. The creation time is constant for the single Prolog interpreter as there is only ever one instantiated. The time to query the single interpreter for 100,000 assertions took longer than overnight so the query was cancelled.

6 Discussion and Conclusions

The convolution of declarative and reasoning engines in large-scale multi-agent simulation systems is an exciting area of research that we believe will show the way forward for many of the more complex problems facing researchers who wish to add more complexity into their simulations, while retaining their run-time efficiency.

There exist many off-the-shelf Prolog systems that allow the use of Prolog knowledge management and searching techniques with imperative languages (such as CKIProlog [16], JIProlog [3] and Minerva [12]) without having to incur significant overheads such as swapping process contexts. Applications Programmer Interfaces are provided to many systems, and some, such as CKIProlog, are already implemented in an imperative language and accessible by native programming code.

We have seen in this paper that while it is logically more elegant to only have a single Prolog engine that stores all the facts for a given system, the explosion in run-time access when the number of individual agents numbers more than 1,000 makes this approach infeasible. While it is more expensive, in terms of memory overhead, it is more efficient to maintain a separate Prolog engine for each agent in the system.

This new approach, incorporating Prolog with the existing Java/C/C++ code does not make the code any messier, although the model set-up time is increased because of the need to distribute out to all animats information on the physical bounds of their environment. Also somewhat tricky are the neighbour-checking routines that determine friend/foe proximity.

The next logical step for this work is how we can use this approach to implement self-modifying rules for our animats to “breed a better animal” and pass some (limited) knowledge on to future generations of predators or prey.

Acknowledgements

This work has benefited from the use of Massey University’s “Monte” compute cluster and thanks also to the Allan Wilson centre for Molecular Biology for the use of the “Helix” Supercomputer.

References

- [1] Ait-Kaci, H. “Warren’s Abstract Machine: A Tutorial Reconstruction”, MIT Press, ISBN 0262691469, April 1991. Available at <http://www.vanx.org/archive/wam/wam.html>
- [2] Bak, P. “How Nature Works: The Science of Self-Organised Criticality” Pub. New York, NY: Copernicus Press, 1996, ISBN 0-387-94791-4
- [3] Chirico, U. “JIProlog 3.0.1”. Available from <http://www.ugosweb.com/jiprolog>
- [4] Ferber, J. “Multi-Agent Systems An Introduction to Distributed Artificial Intelligence”, Addison-Wesley, 1999, ISBN 0-201-36048-9.
- [5] Hawick, K.A. “Agent Formulation of the Ising Model”, Computational Science Technical Note CSTN-004, Massey University, December 2003.

- [6] Hawick, K.A., Scogings, C.J. and James, H.A. “Defensive Spiral Emergence in a Predator-Prey Model” in Proc. Complexity 2004, Cairns, Australia, December 2004.
- [7] Hawick, K.A., James, H.A. and Scogings, C.J. “A Zoology of Emergent Patterns in a Predator-Prey Model”, Computational Science Technical Note CSTN-015, Massey University, March 2005.
- [8] K.A., James, H.A. and Scogings, C.J. “Roles of Rule-Priority Evolution in Animat Models”, Computational Science Technical Note CSTN-020, Massey University, June 2005. Submitted to ACAL’05, Sydney, 2005.
- [9] Hawick, K.A. and James, H.A. “Small-World Effects in Wireless Agent Sensor Networks”, to appear in Int J. Wireless and Mobile Computing (IJWMC) Special issue on Mobile Systems, E-Commerce and Mobile Agents, Accepted in October 2004.
- [10] Hawick, K.A., James, H.A. and Scogings, C.J. Web site, containing Model Snapshots and Movie sequences: <http://www.massey.ac.nz/~kahawick/alife> Massey University 2004-5
- [11] International Standards Organisation “Prolog Standards, ISO/IEC 13211-1:1995”, 1995.
- [12] IF Computer. “MINERVA” Available from <http://www.ifcomputer.co.jp/MINERVA>
- [13] James, H.A., Scogings, C.J. and Hawick, K.A. “A Framework and Simulation Engine for Studying Artificial Life”, in Research Letters in the Information and Mathematical Sciences, Vol 6, May 2004, pp143–155, ISSN 1175-2777. At: <http://iims.massey.ac.nz/research/letters/-volume6/>
- [14] James, H.A., Scogings, C.J. and Hawick, K.A. “Parallel Synchronisation issues in Simulating Artificial Life”, in Proc. 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS), pp815–820, Boston, Nov 2004.
- [15] Lotka, A.J. “Elements of Physical Biology”, Baltimore: Williams & Wilkins Co, 1925.
- [16] van Otterloo, S. “CKI Prolog” Available from <http://www.csc.liv.ac.uk/~sieuwert/-programs.html>
- [17] Rao, A.S. and Georgeff, M.P. “BDI Agents: From Theory to Practice”, in Proc First Int Conf on Multi-Agent Systems (ICMAS-95), San Francisco, USA June, 1995.
- [18] Ronald, E.M.A., Sipper, M. and Capcarrère, M.S. “Testing for Emergence in Artificial Life”, in Advances in Artificial Life: Proc. 5th European Conference on Artificial Life (ECAL’99), Switzerland, 1999 Ed. Dario Floreano and Jean-Daniel Nicoud and Francesco Mondada, Pages 13-20, Pub Springer ISBN 3-540-66452-1.
- [19] Russell, S. and Norvig, P. “Artificial Intelligence: A Modern Approach” (2nd Ed), Prentice Hall, 2003, pp. 197–200.
- [20] Wilson, S. “The Animat Path to AI” in *From Animals to Animats 1: Proceedings of The First International Conference on Simulation of Adaptive Behavior*, (pp. 15-21); Meyer, J-A. & Wilson, S. (eds), Cambridge, MA: The MIT Press/Bradford Books (1991).
- [21] Volterra, V. “Variazioni e Fluttuazioni del Numero d’Individui in Specie Animali Conviventi”, Mem. R. Accad. Naz. dei Lincei, Ser VI, Vol 2, 1926.