Computational Science Technical Note **CSTN-022**

A Virtual Prolog Approach to Implementing Beliefs, Desires and Intentions in Animat Agents

K. A. Hawick and H. A. James and C. J. Scogings

2005

Simulating a system of agents that navigate in a physical space is a challenging problem when the environment is sophisticated and the number of agents involved is large. We describe experiments in establishing a "virtual Prolog engine" in each agent in a simulation and the scalability and performance issues with such an approach. We report on experiments with a simple predator-prey animat model and discuss how this approach allows us to impart a degree of reasoning and intelligence to what are otherwise very simple animat agents.

Keywords: animat agents; beliefs, desires and intentions; physical navigation and reasoning

A Virtual Prolog Approach to Implementing Beliefs, Desires and Intentions in Animat Agents

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

Abstract

Simulating a system of agents that navigate in a physical space is a challenging problem when the environment is sophisticated and the number of agents involved is large. We describe experiments in establishing a "virtual Prolog engine" in each agent in a simulation and the scalability and performance issues with such an approach. We report on experiments with a simple predator-prey animat model and discuss how this approach allows us to impart a degree of reasoning and intelligence to what are otherwise very simple animat agents.

Keywords: animat agents; beliefs, desires and intentions; physical navigation and reasoning.

1   Introduction

The problem of representing per-agent knowledge in a multi-agent simulation system is not an easy one, especially if one considers the situation in which there are a very large number of independent agents. This problem is easily solvable if one makes the simplifying assumption that agents are homogeneous and they are "memory-less" agents. However, as soon as individual agents are enabled with a memory this problem becomes far more complex.

In this paper we consider the benefits and drawbacks of using a Prolog [5] 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 "Virtual Prolog" simply to highlight the fact that we may or may not have embedded within our simulation system a commercial implementation of the Prolog engine, and that a single such engine may or may not be shared between all agents in our simulation.

We consider a general architecture for an animat agent. The animat encounters the environment in terms of other animats of a particular kind at particular locations during its travels. This, along with an internal notion of time, its own encoded behaviour (either learned or inherited), and a memory of where it has been in its own particular reference frame can combine to produce a decision on where to move next or what actions to perform. This idea is not new and forms the basic model for many robots and simulated life forms.

We consider the implementation issues of using a coded set of instructions or a logical reasoning engine as the core of the animat. We are especially interested in exploring the many-animat regime which is not yet economically feasible to explore for hardware robots with their own on-board computers. Finally, 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.

We have constructed a predator-prey model consisting of two distinct groups of animats: predators (known as "foxes") and prey (known as "rabbits"). 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 [7]. In our current simulation system, all species have the ability to perceive other animals up to a given radius; this perception radius is currently set to 80 pixels for foxes and 20 pixels for rabbits. Beyond that, nothing can be perceived. As our current simulation is "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 [3]. We have conducted experiments with changing the order of priority of the rules [4]. These experiments do not qualify as genetic evolution experiments as every animat is an exact clone of its parents. The predator-prey model outlined above has resulted in some fascinating emergent behaviour in which animats formed clusters and spirals. This behaviour is analysed in [3]. However, interesting behaviours only occur 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 individual animats.

The BDI (Belief, Desire, Intention) [9] formalism is useful for representing the fully cognitive agent approach in which agents are able not only to store a searchable history, but are also able to formulate (or at least have specified) some sort of goals. This is a relatively well understood theory in AI and distributed AI [2]. Beliefs and intentions are relatively simple to express in our agent-based system: beliefs are based on an agent's perception of the environment, including similar and dissimilar types of animal and the physical environment; intentions are the rules invoked in response to the beliefs and desires of the animals. In a 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.

2 Declarative and Reasoning Engines

Prolog 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. We have implemented a version of our simulation in

Java for the purpose of graphical interactivity and Prolog experimentation. 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 anticipate that by combining both programming methodologies, we will be able to only use the most appropriate features of both systems, introducing few inefficiencies.

Consider the very simple case in which each of our 20,000 unique agents has moved only four times, encountering a different object at each step. 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.

We have considered two fundamental ways to link the simulation program manager with the Prolog engine: a) 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) 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 Prolog engine deal with the non-scalability. This is the second case under consideration: 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.

We have also trialled CKIProlog [8] and JIProlog [1], Prolog-like libraries 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. Table 1 shows the representative times taken by the CKIProlog Java library to perform various simple operations on our multi-agent simulation systems. Times reported in the table 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. Note that the time taken to query the single interpreter for 100,000 assertions took longer than over-night, so the query was cancelled. We have also considered semi-commercial packages such as Minerva [6] but as the style of programming was quite radically different from the existing C/C++/Java implementations we have not continued to use them.

The overwhelming result from the experiments reported in table 1 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

Table 1: 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.

number of animats	Prolog Interpreter per 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	

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.

3   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 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 shown 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.

References

[1] Chirico, U. "JIProlog 3.0.1". Available from http://www.ugosweb.com/jiprolog

[2] Ferber, J. "Multi-Agent Systems An Introduction to Distributed Artificial Intelligence", Addison-Wesley, 1999, ISBN 0-201-36048-9.

[3] 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.

[4] Hawick, K.A., James, H.A. and Scogings, C.J. "Roles of Rule-Priority Evolution in Animat Models", To appear *Proc ACAL'05*, Sydney, 2005.

[5] International Standards Organisation "Prolog Standards, ISO/IEC 13211-1:1995", 1995.

[6] IF Computer. "MINERVA" Available from http://www.ifcomputer.co.jp/MINERVA

[7] 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. Available from http://iims.massey.ac.nz/research/-letters/volume6/

[8] van Otterloo, S. "CKI Prolog" Available from http://www.csc.liv.ac.uk/~sieuwert/-programs.html

[9] 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.