A Logic Based Approach to Multi-Agent Systems

J. J. Alferes * P. Dell'Acqua+ E. Lamma† J. A. Leite*
L. M. Pereira* F. Riguzzi†

* Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
{jja,jleite,lmp}@di.fct.unl.pt

+ Department of Science and Technology, Campus Norrköping,
Linköping University, Norrköping, Sweden
pier@itn.liu.se

† Dipartimento di Ingegneria, Universitá di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy
elamma@deis.unibo.it
friguzzi@ing.unife.it

1 Research Work

The research group at CENTRIA UNL in Lisbon, in collaboration with Ferrara and Linköping, has been quite active in the field of Logic Programming (LP) geared towards rational agents over several years, and has investigated a spate of rational agent functionalities and their implementations. In particular, the Lisbon group has been active in the fields of learning, abduction, updating, argumentation, paraconsistency, belief revision, soft LP, contradicition removal, diagnosis, and debugging[1]. These research directions are all included in the broader long term research avenue of building evolving and collaborative rational epistemic agents, whose overarching objective is to establish a flexible declarative language for the specification and implementation of dynamic knowledge construction in a society of agents. In [32, 33], the philosophical foundations and the general line of approach to the definition of rational agents, based on and building upon the strengths of LP, have been put forward.

Other recent results by the team include the combination of various pieces of knowledge, possibly originating in various agents, each of which with ascribed priorities over the others, the totality being organized via an acyclic graph [25, 28].

The long term objective of the group specifically includes the building of rational epistemic agents capable, in an integrated fashion, of reacting to a dynamic environment comprising other agents; managing and revising their goals,

[1] Cf. project MENTAL "An Architecture for Mental Agents". Available at http://centria.di.fct.unl.pt/~ica/mental/

knowledge, and beliefs; making hypotheses about the world in order to plan and achieve the goals; learn with experience; and enabled with introspective capabilities. For this LP has been successfully used both as the unified encompassing theoretical basis, and as the implementation vehicle.

A declarative language for specifying dynamic knowledge, also capable of combining knowledge and its production in a dynamically configurable society of agents, is a key ingredient for the long term goal. Agents receive information from various sources, and this information must be combined [27]. An agent must be able to adapt to a dynamic environment, and its knowledge must contain information on how the agent can modify and communicate with it.

Herein we offer an overview of the agent related results and research directions of the group, complemented by pointers to publications, prototypes, project reports, and ongoing work. We begin by introducing the foundational results in Section 2, followed by portraying the general agent architecture in Section 3, and by a specific agent theory in Section 4. Next, in Section 5, we present an innovative concrete logic based genetic learning functionality, appropriate either for a single agent or for a population of agents with distributed knowledge. A subsequent section 6 envisages how the different rational agent encompassing functionalities can work in unison to bring about a society of epistemic agents. Finally, a Conclusion section summarizes and draws out the importance of LP for the whole endeavour.

2 Logics and Languages for Engineering Agents

Till recently, *Logic Programming* could be seen as a good representation language for stactic knowledge. If we are to move towards a more open and dynamic environment, typical of the agency paradigm, we need to consider ways of representing and integrating knowledge from different sources which may also evolve in time. Moreover, an agent not only comprises knowledge about each state, but also some form of knowledge about the transitions between states. This knowledge can latter represent the agent's knowledge about the environment's evolution, as well as its own actions' results and its internal evolution. Since logic programs describe knowledge states, it's only fit that logic programs describe transitions of knowledge states as well. It is natural to associate with each state a set of transition rules to obtain the next state. Recent developments have opened *Logic Programming* to these otherwise unreachable dynamic worlds.

In [2, 3], *Dynamic Logic Programming* (DLP) was introduced by us, following the eschewing of performing updates on a model by model basis, but rather as a process of rule updates [29, 30]. There, we have studied and defined the declarative and operational semantics of sequences of logic programs (or dynamic logic programs). Each program in the sequence contains knowledge about some given state, where different states may, for example, represent different time periods or different sets of priorities. The introduction of *Dynamic Logic Programming* has extended Logic Programming, making it possible for a program to undergo a sequence of modifications. This opens up the possibility of incremental design

and evolution of logic programs, and therefore significantly facilitating *modularization* of programming in logic and, thus, modularization of non-monotonic reasoning as well.

Dynamic Logic Programming does not by itself provide a proper language for specifying (or programming) changes into logic programs. If knowledge is already represented by logic programs, dynamic programs simply represent the evolution of knowledge. But how is the evolving of knowledge specified? What makes knowledge evolve? It is natural to associate with each state a set of transition rules to obtain the next state. Consequently, an interleaved sequence of states and applicable rules of transition is obtained.

In [7, 6], the language *LUPS* – "*Language for dynamic updates*" – was introduced, designed for specifying changes to logic programs. Given an initial knowledge base (as a generalized logic program) *LUPS* provides a means for sequentially updating it, by means of update commands. Besides simple unconditional assertions and rectractions of rules, *LUPS* also includes commands that make such assertions/retractions dependent on conditions presently verified in the knowledge base, persistent commands that are applicable at every transition, commands for cancelling persistent commands, and event assertion/retraction commands valid only for the next transition. The declarative meaning of a sequence of sets of update commands in *LUPS* is defined by the semantics of the corresponding dynamic logic program generated by those commands.

Imperative programming specifies transitions and leaves states implicit. Logic programming, traditionally, could not specify state transitions. *Dynamic Logic Programming*, together with the language of dynamic updates *LUPS* makes both states and their transitions declarative.

Even though the main motivation behind the introduction of *Dynamic Logic Programming* was to represent the evolution of knowledge in time, several updating dimensions may combine simultaneously, with or without the temporal one, such as specificity (as in taxonomies), strength of the updating instance (as in the legislative domain), hierarchical position of knowledge source (as in organizations), credibility of the source (as in uncertain, mined, or learnt knowledge), or opinion precedence (as in a society of agents). For this to be possible, *DLP* needs to be extended to allow for a more general structure of states. In [25, 28] we have introduced *Multi-dimensional Dynamic Logic Programming* ($\mathcal{MDLP}$) to generalize *Dynamic Logic Programming* to cater for collections of states organized by arbitrary acyclic digraphs, not just sequences of states. $\mathcal{MDLP}$ assigns semantics to sets and subsets of logic programs, depending on how they stand in relation to one another, as defined by the acyclic digraph (*DAG*) that represents the states and their configuration. By dint of such natural generalization, $\mathcal{MDLP}$ affords extra expressiveness, thereby enlarging the latitude of logic programming applications unifiable under a single framework. The generality and flexibility provided by a *DAG* ensures a wide scope and variety of new possibilities. By virtue of the newly added characteristics of multiplicity and composition, $\mathcal{MDLP}$ provides a "societal" viewpoint in Logic Programming, important in these web and agent days, for combining knowledge in general.

Based on the strengths of $\mathcal{MDLP}$ as a framework capable of simultaneously representing several aspects of a system in a dynamic fashion, and on the strengths of *LUPS* (and its extension to the multi-dimensional case) as a powerful language to specify the evolution of DAGs, by means of transitions, we have launched ourselves into the design of an agent architecture, $\mathcal{MINERVA}$ [26], with the intention of providing, on a sound theoretical basis, a common agent framework grounded on the strengths of Logic Programming, so as to permit the combination of several non-monotonic knowledge representation and reasoning mechanisms developed in recent years, provided by the functionalities of diverse agents.

3 Agent Architecture

In our opinion, to carry out their tasks, rational agents will require an admixture of any number of the reasoning mechanisms mentioned in the Introduction. To this end, a $\mathcal{MINERVA}$ agent is based on a modular design where a common knowledge base is concurrently manipulated by specialized sub-agents (schematically depicted in Fig. 1). The common knowledge base contains all knowledge shared by more than one sub-agent. It is conceptually divided in the following components: *Capabilities*, *Intentions*, *Goals*, *Plans*, *Reactions*, *Object Knowledge Base* and *Internal Behaviour Rules*, together with an internal clock. Although conceptually divided into such components, all these modules will share a common representation mechanism based on $\mathcal{MDLP}$ and *LUPS*, the former to represent knowledge at each state and *LUPS* to represent the state transitions, i.e. the common part of the agent's behaviour. Every agent is composed of specialized function related sub-agents, that execute their various specialized tasks. These sub-agents constitute the labour intensive part of the $\mathcal{MINERVA}$ architecture. Their tasks reside in the evaluation and manipulation of the *Common Knowledge Base* and, in some cases, in the interface with the environment in which the agent is situated and possibly with private specialized procedures. They differ essentially in their speciality inasmuch as there are those implementing the reactive, planning, scheduling, belief revision, goal management, learning, dialogue management, information gathering, preference evaluation, strategy, and diagnosis functionalities. Whilst some of those sub-agent's functionalities are fully specifiable in *LUPS*, others will require private specialized procedures where *LUPS* serves as an interface language to them. Each of these sub-agents, therefore, contains a *LUPS* program encoding its behaviour and its interface with the private procedures and environment These LUPS programs are "executed" by a meta-interpreter, producing a sequence of states in the structures of the Common Knowledge Base. The collection of all such sequences of states, produced by all the sub-agents will constitute the states of the $\mathcal{MDLP}$.

Conceiving the architecture based on the notion of, to some extent, independent sub-agents allows some degree of modularity, useful in what concerns the adaptability of the architecture to different situations since not all sub-agents are required for all situations.This basic architecture affords us, we believe, with

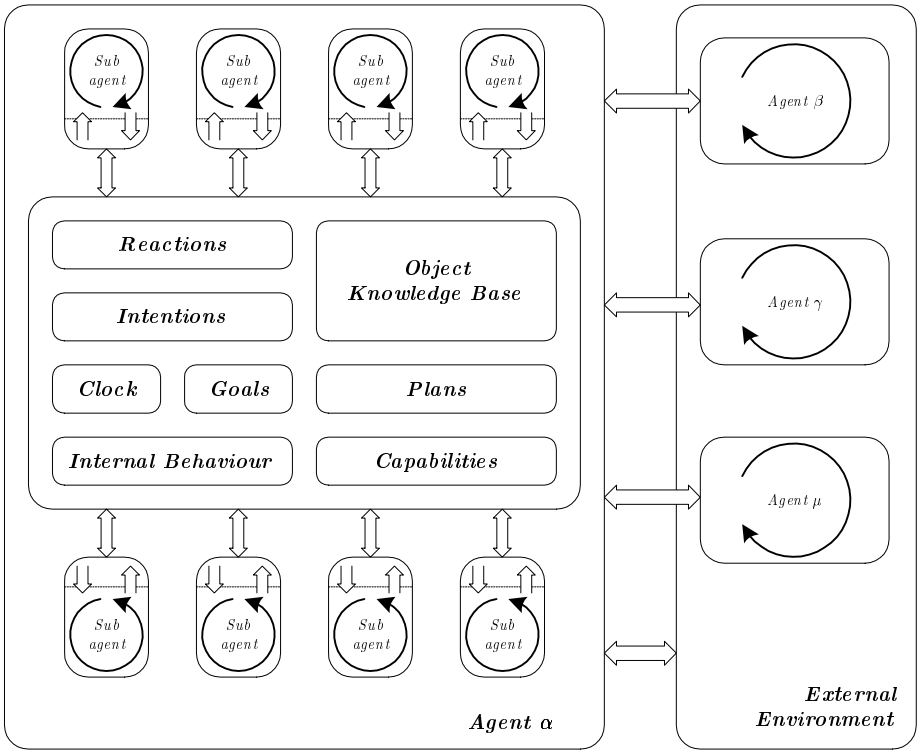

Fig. 1. The $\mathcal{MINERVA}$ agent architecture

the elasticity and resilience to further support a spate of crucial ancillary functionalities, in the form of additional specialized agents, via compositional, communication, and procedural mechanisms.

4 Engineering Agent Theories

In [14, 15] we developed a specific logical formalization of a framework for multi-agent systems and defined its semantics, based on the foundational work described in section 2, and in [19, 5]. In it we can embed a flexible and powerful kind of agent. In fact, these agents can be rational, reactive, abductive, able to prefer and they can update the knowledge base of other agents (including their own).

The knowledge state of each agent is represented by an abductive logic program in which it is possible to express rules, integrity constraints, active rules and priorities among rules. This renders the agents able to reason, to react to the environment, to prefer among several alternatives, to update both beliefs and reactions, and to abduce hypotheses to explain observations made.

These agents can be embedded into a multi-agent system in such a way that the only form of interaction among them is based on the notions of project and update. A project of the form $\alpha{:}C$ of an agent β denotes the intention of β of proposing to update the theory of an agent α with C. Correspondingly, an update of the form $\beta \div C$ in the theory of α denotes the intention of β to update the current theory of α with C. It is then up to α whether or not to accept that update. For example, if α trusts β and therefore α is willing to accept it, then α has to update its theory with C. The new information may contradict what α believes and, if so, the new believed information will override what is currently believed by α. β can also propose an update to itself by issuing an internal project $\beta{:}C$.

The semantics of the multi-agent system provides a logical description of the agent interactions. The definition of the semantics depends also on the goal that each agent has to prove at a certain moment. In fact, when proving a goal the agent may abduce hypotheses that explain the goal. These in turn may trigger reactive rules, and so on. Hypotheses abduced in proving a goal G are not permanent knowledge, rather they only hold during the abductive proof of G. To make them permanent, an agent can issue an internal project and then update its own knowledge base with those hypotheses.

The engineering of our agent society is based exclusively on the notions of projects and updates to model the agent interactions, and permits though simple, building autonomous and distributed agent systems. This form of interaction is powerful and flexible, and a number of communication protocols can be built on top of it. In [16] we argue that our present theory of the type of agents is a rich evolvable basis suitable for engineering agent societies. In fact, the framework assembles essential ingredients of an agent architecture needed to engineer open, dynamic agent societies where the agents can self-organize themselves with respect to their goals, and self-evolve. The overal "emerging" structure will be flexible and dynamic: each agent will have its own explicit representation of its organization which, furthermore, is updatable by preferring.

Currently, we are implementing a prototype of our agents in Java. In particular, we are designing and implementing the mechanism that models the interplay between the abilities of rationality and reactivity based on the notion of interrupts. The knowledge base and the updating mecanism of the agent are implemented in Prolog and run under the logic programming environment XSB system [11] under the well-founded semantics. The Prolog implementation can then be incorporated into the Java prototype via InterProlog [10], an interface between the XSB system and Java.

5 A Logic Based Genetic Learning Facility

Belief revision is an important functionality that agents must exhibit: agents should be able to modify their beliefs in order to model the outside world. Moreover, they need to perform this task in cooperation with other agents, because

access to knowledge and the knowledge itself are distributed in nature, i.e., each agent has only a partial knowledge of the world.

In [20, 22, 21] we considered a definition of the belief revision problem that consists in removing a contradiction from an extended logic program [34, 4, 8] by modifying the truth value of a selected set of literals called *revisables*. The program contains as well clauses with false ($\bot$) in the head, representing *integrity constraints*. Any model of the program must ensure that the body of integrity constraints be false for the program to be non-contradictory. Contradiction may also arise in an extended logic program when both a literal L and its opposite $\neg L$ are obtainable in the model of the program. Such a problem has been widely studied in the literature, and various solutions have been proposed [9, 13] that are based on abductive logic proof procedures.

A new approach was proposed in [20, 22, 21, 23, 24] for performing belief revision in a society of logic-based agents, by means of a (distributed) genetic algorithm. The problem can be modeled by means of a genetic algorithm, by assigning to each revisable of a logic program a gene in a chromosome. In the case of a two-valued revision, the gene will have the value 1 if the corresponding revisable is true and the value 0 if the revisable is false. The fitness function that is used in this case is represented in part by the percentage of integrity constraints that are satisfied by a chromosome.

Each agent keeps a population of chromosomes and finds a solution to the revision problem by means of a genetic algorithm. We consider a formulation of the revision problem where each agent has the same set of revisables and the same program, but is subjected to possibly different observations and constraints. Observations and constraints may vary over time, and can differ from agent to agent because agents may explore different regions of the world. Each agent by itself locally performs a genetic search in the space of possible revisions of its knowledge, and exchanges genetic information by crossing its revisable chromosomes with those of other agents. In this way, we achieve distribution in belief revision since chromosomes coming from different agents, through crossover, contribute to solve the problem.

We have performed experiments comparing the evolution in beliefs of a single agent informed of the whole of knowledge, to that of a society of agents, each agent accessing only part of the knowledge. The experiments have been performed on problems of model based diagnosis, a natural domain in which belief revision techniques apply [13], and on the n-queen problem. In spite that the distribution of knowledge increases the difficulty of the problem, experimental results [23] show that the solutions found in the multi-agent case are comparable in terms of accuracy to those obtained in the single agent case.

The genetic algorithm we propose, besides encompassing the Darwinian operators of selection, mutation and crossover, also comprises a Lamarckian operator. Darwin's theory is based on the concept of natural selection: only those individuals that are most fit for their environment survive, and are thus able to generate new individuals by means of reproduction. Moreover, during their lifetime, individuals may be subject to random mutations of their genes that they

can transmit to offspring. Lamarck's theory, instead, states that evolution is due to the process of adaptation to the environment that an individual performs in its lifetime. The results of this process are then automatically transmitted to its offspring, via its genes. In other words, the abilities learnt during the life of an individual can modify its genes.

Experimental evidence in the biological kingdom has shown Darwin's theory to be correct and Lamarck's to be wrong, although the more recently evolved concept of "meme" supports *cultural* Lamarckian evolution (cf. [12]). However, in the field of genetic programming, Lamarckian evolution has proven to be a powerful concept and various authors have investigated the combination of Darwinian and Lamarckian evolution [18, 1, 31, 17].

The Lamarckian operator we propose differs from Darwinian ones precisely because it modifies a chromosome coding beliefs so that its fitness is improved by experience rather than in a random way. We call memes those genes in the chromosomes that can be modified by the lamarckian operator. They may be a subset of all the genes, i.e., not all the genes may be modified by the Lamarckian operator. The Lamarckian operator modifies the memes by means of a (logic-based) procedure inspired by [35]: the logical derivations leading to the inconsistency of belief are traced so as to remove these derivations' support on the meme coded assumptions, effectively by mutating the latter. In our algorithm, therefore, computational logic is used in order to find good revisions that are then distributed by means of the crossover genetic operator.

The adoption of computational logic methods in a genetic algorithm provides an improvement over purely genetic approaches: in [20, 22, 21] we have shown that the addition of the Lamarckian operator to a single-agent version of the genetic algorithm improves the fitness that can be reached by the algorithm in a given number of generation. In [20, 22, 21] we have also shown that the addition of the Lamarckian operator to the multi-agent system has the same effect.

Moreover, in order to take into account the different nature of memes, we have modified the Darwinian crossover operator so that memes of an another agent can be acquired only if they have been checked for consistency, and possibly mutated by the Lamarckian operator, as a result of its own observational experience. Experimental results show that this special treatment of memes leads to improved results in terms of accuracy.

We believe our method to be important for situations where classical belief revision methods hardly apply: those where environments are non-uniform and time changing and especially those where the knowledge is distributed. These environments can be explored by distributed agents that evolve genetically to accomplish cooperative belief revision, if they use our approach.

6 A Dynamic Knowledge Framework

It is not too difficult to imagine how a combined process of rule generation, of systematic diagnosis, and of rule revision by updating, can be used to achieve

automated theory learning, in an integrated way, within the uniform setting of logic programming.

To initiate the learning, one starts with some fixed, already acquired, background knowledge in rule form, i.e. a theory, and with a rule generator to add to it new purported knowledge, in order to explain abductively known observations, whether positive or negative, in the form of facts and explicitly negated facts.

The goal is to generate rules that define a positive concept as well as its negated concept, so that they cover all known observation instances. This automatic generation of new rules is subjected to a pre-defined bias, i.e. only some rule forms, and predicates comprising them, are allowed in the generation process. Newly generated rules may contradict one another, on some of the observation instances, and so they must be subjected to a diagnosis, to identify alternative possible minimal revisions.

To decide which revision to adopt next, desirable new possible observations are conceived of, with respect to the ongoing available theory, and whose results, if known, would allow to decide among the competing revisions. The results of these so-called crucial observations are then obtained, either by the program soliciting them, or by a belief revision process subsequently executed to carry out the actions leading to the observation results

Once the desired revisions are selected on the basis of the results, and of also programmed preference criteria, the revisions are enacted by an update procedure. Note that revised rules can themselves be subjected to later revisions if needed.

Indeed, the whole process will be iterated on the basis of new incoming knowledge, or by knowledge confrontation of among differently evolved automated theories, with distinct backgrounds, biases, rule generators, diagnosers, revisors, preferences, planners, observations, and updating procedures, comprising a rational agent.

Agent epistemic confrontation relies on argumentation and mutual debugging. Besides the legislative and legal domain, the province of scientific discussion too relies on such procedures, and can benefit from their automation.

Tackling argumentation involves a set of tools similar to those of diagnosis and debugging. Arguments can attack another argument's assumptions either directly, by proving the negation of an assumption, or indirectly, by contradicting a conclusion of other argument which rests on its assumptions. However, such attacks by an argument upon another can, in turn, be counter-attacked in the same way, and may in response counter-counter-attack, etc. logic programming has shown how this process can be studied and conclusions drawn about competing mutually contradictory arguments, and how they each can be revised to reach agreement.

7 Conclusion

In both academia and industry it is increasingly felt that intelligent agents will be a key technology as computing systems become ever more distributed, interconnected, and open. In such environments, the ability of agents to autonomously plan and pursue their actions and goals, to cooperate, coordinate, and negotiate with others, and to respond flexibly and intelligently to dynamic and unpredictable situations will lead to significant improvements in the quality and sophistication of the software systems that can be conceived and implemented, and the application areas and problems which can then be addressed.

Our overall aim is that of establishing, on a sound theoretical basis, the design of a general architecture for mental agents (i.e. comprising knowledge, beliefs, and intentions) based on, and building on the strengths of logic programming.

The agents share a common changeable environment, itself modeled in logic programming. The whole framework is being conceptualized in logic programming, and experimented with by means of distributed logic programming [26]. The use of agents allows for information to be distributed, and shared, only as the need arises. Thus knowledge manipulation complexity can be reduced. Efficiency can be gained too by executing agents concurrently.

An agent must be able to manage its knowledge, beliefs, intentions (goals), and plans, as it receives new information and instructions, and to react to changing conditions in the environment. It must also be capable of interacting with other agents, by exchanging knowledge and beliefs, as well as of reacting to other agents' requests. Any two such agents will be able to cooperate either by diagnosing errors or missing information in each others' information base, or to cooperate in the use of common resources, in avoiding undesired mutual interference in their plans, and in joint belief revision to achieve a common goal.

Every agent is composed of specialized, possibly concurrent, function related subagents, that execute the various goals and instructions assigned to it by the agent. Examples of such subagents are those implementing the reactive, reasoning, belief revision, argumentation, explanation, learning, dialogue management, information gathering, preference evaluation, strategy, and diagnosis functionalities. The subagents are coordinated by a meta-level layer that ensures internal task distribution and belief revision, subagent communication, final decisions, and all interaction with the outside, i.e. with both the environment and other agents, including communication, interrupt handling, requests, and observations.

Although each agent has a meta-level coordination, a collection of interacting agents has no such abode, and their collective behaviour will manifest (emergent) properties that are difficult to foresee.

Because knowledge and belief are in general incomplete, contradictory, and error prone — and all the more so in the multi-agent setting — we make use of semantics, procedures, and implementations, for dealing with contradictory, incomplete, erroneous, imperfect, vague and default information, abduction, belief revision, debugging, and argumentation.

We attain reinforcement learning by using belief revision techniques for achieving an effect similar to back propagation. Such learning allows the evolving of the

strength of evidence attached to an agent's knowledge and beliefs, as a result of their contribution to correct or incorrect conclusions. Furthermore, agents may compare and combine their degrees of evidence, either to argue, to partake of information, or to reach consensus. The use of genetic algorithms for evolving belief "genes" (or "memes") is also a technique we have explored for learning, and it opens up a new emergent field: that of linking the genetic algorithm approach to logic based knowledge evolution in an agent. Moreover, the exchange of genetic material enables agents to cross-fertilize experiences.

The use of logic programming for the overall endeavour is justified on the grounds of it providing a rigorous single encompassing theoretical basis for the aforesaid topics, as well as an implementation vehicle for parallel and distributed processing. Additionally, logic programming provides a formal high level flexible instrument for the rigorous specification and experimentation with computational designs, making it extremely useful for prototyping, even when other, possibly lower level, target implementation languages are envisaged.

To conclude, Logic Programming is, without a doubt, the privileged melting pot for the articulate integration of functionalities and techniques, pertaining to the design and mechanization of complex systems, addressing ever more demanding and sophisticated computational abilities. For instance, consider again those reasoning abilities mentioned above. Forthcoming rational agents, to be realistic, will require an admixture of any number of them to carrying out their tasks. No other computational paradigm affords us with the wherewithal for their coherent conceptual integration. And, all the while, the very vehicle that enables testing its specification, when not outright its very implementation. Or is there?

Acknowledgements

The CENTRIA researchers acknowledge the support of PRAXIS projects MENTAL and FLUX. The work of J. A. Leite was also supported by PRAXIS Scholarship no. BD/13514/97.

References

1. D. H. Ackely and M. L. Littman. A case for lamarckian evolution. In C. G. Langton, editor, *Artificial Life III*. Addison Wesley, 1994.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic logic programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 98–111, San Francisco, June 2–5 1998. Morgan Kaufmann Publishers.
3. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1–3):43–70, September/October 2000.
4. J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *LNAI*. Springer-Verlag, 1996.

5. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmn, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Procs. JELIA'00*, LNAI 1919, pages 345–360, Berlin, 2000. Springer.
6. J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS : A language for updating logic programs. *Artificial Intelligence*. To appear.
7. J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS : A language for updating logic programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-99)*, volume 1730 of *LNAI*, pages 162–176, Berlin, December 2–4 1999. Springer.
8. J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20:107–142, 1998.
9. J. J. Alferes, L. M. Pereira, and T. Swift. Well-founded abduction via tabled dual programs. In D. De Schreye, editor, *Procs. of the 16th International Conference on Logic Programming*, pages 426–440, Las Cruces, New Mexico, 1999. MIT Press.
10. InterProlog. Available at http://www.declarativa.com/InterProlog/default.htm.
11. XSB-Prolog. Available at http://xsb.sourceforge.net/.
12. Susan Blackmore. *The Meme Machine*. Oxford U.P., Oxford, UK, 1999.
13. C. V. Damásio, L. M. Pereira, and M. Schroeder. REVISE: Logic programming and diagnosis. In *Proceedings of Logic-Programming and Non-Monotonic Reasoning, LPNMR'97*, volume 1265 of *LNAI*, Germany, 1997. Springer-Verlag.
14. P. Dell'Acqua and L. M. Pereira. Updating agents. In S. Rochefort, F. Sadri and F. Toni (eds.), Procs. of the ICLP'99 Workshop on Multi-Agent Systems in Logic (MASL'99), 1999.
15. P. Dell'Acqua and L. M. Pereira. Preferring and updating with multi-agents. Submitted, 2001.
16. P. Dell'Acqua and L. M. Pereira. Preferring and updating in abductive multi-agent systems. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *ESAW01*, pages 53–66, 2001. Available at: http://lia.deis.unibo.it/confs/ESAW01/.
17. J. J. Grefenstette. Lamarckian learning in multi-agent environment. In *Proc. 4th Intl. Conference on Genetic Algorithms*. Morgan Kauffman, 1991.
18. W. E. Hart and R. K. Belew. Optimization with genetic algorithms hybrids that use local search. In R. K. Belew and M. Mitchell, editors, *Adaptive Individuals in Evolving Populations*. Addison Wesley, 1996.
19. R. A. Kowalski and F. Sadri. Towards a unified agent architecture that combines rationality with reactivity. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases, Intl. Workshop LID'96*, LNCS 1154, pages 137–149. Springer, 1996.
20. E. Lamma, F. Riguzzi, and L. M. Pereira. Logic aided lamarckian learning. In *Proc. 5th Intl. Workshop on Multistrategy Learning*, 2000. Avaliable at `http://www.ing.unife.it/docenti/FabrizioRiguzzi/msl00.ps`.
21. E. Lamma, F. Riguzzi, and L. M. Pereira. Belief revision via lamarckian evolution. Submitted for publication, available at `http://www.ing.unife.it/docenti/FabrizioRiguzzi/ga2001.ps`, 2001.
22. Evelina Lamma, Fabrizio Riguzzi, and Luís Moniz Pereira. Belief revision by lamarckian evolution. In *First European Workshop on Evolutionary Learning (EvoLEARN2001)*, LNCS. Springer-Verlag, 2001.
23. Evelina Lamma, Fabrizio Riguzzi, and Luís Moniz Pereira. Belief revision by multi-agent genetic search. Submitted for publication, available at `http://www.ing.unife.it/docenti/FabrizioRiguzzi/cl2001.ps`, 2001.

24. Evelina Lamma, Fabrizio Riguzzi, and Luís Moniz Pereira. A system for multi-agent belief revision by genetic search. Submitted for publication, available at `http://www.ing.unife.it/docenti/FabrizioRiguzzi/cl2001demo.ps`, 2001.
25. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic logic programming. In F. Sadri and K. Satoh, editors, *Proceedings of the CL-2000 Workshop on Computational Logic in Multi-Agent Systems (CLIMA'00)*, pages 17–26, 2000.
26. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In J. J. Meyer and M. Tambe, editors, *Procs. of the Eighth International Workshop on Agent Theories, Architectures, and Languages ATAL'01*, 2001. To appear.
27. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - combining societal agents knowledge. Technical report, Dept. de Informatica, Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, Lisbon, Portugal, 2001.
28. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic knowledge representation. In T. Eiter, M. Truszczynski, and W. Faber, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, 2001. To appear.
29. J. A. Leite and L. M. Pereira. Generalizing updates: From models to programs. In J. Dix, L. M. Pereira, and T. C. Przymusinski, editors, *Selected Extended Papers of the ILPS'97 3th International Workshop on Logic Programming and Knowledge Representation (LPKR-97)*, volume 1471 of *LNAI*, pages 224–246, Berlin, 1997. Springer Verlag.
30. J. A. Leite and L. M. Pereira. Iterated logic program updates. In J. Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP-98)*, pages 265–278, Cambridge, June 15–19 1998. MIT Press.
31. Y. Li, K. C. Tan, and M. Gong. Model reduction in control systems by means of global structure evolution and local parameter learning. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*. Springer Verlag, 1996.
32. L. M. Pereira. The logical impingement of artificial intelligence. In Antonio Zilhao, editor, *Grazer Philosophische Studien (Internationale Zeitschrift Fur Analytische Philosophie), Analytical Philosophy in Portugal*, volume 56, pages 183–204. Amsterdam/Atlanta, 1999.
33. L. M. Pereira. Philosophical incidence of logic programming. In D. Gabbay and J. Woods (eds.), Handbook of History and Philosophy of Logic, Kluwer (in press), 2001.
34. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI92*, pages 102–106. John Wiley and Sons, 1992.
35. L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 316–330. MIT Press, 1993.