1

Philosophical Incidence of Logic Programming

Luís Moniz Pereira[†]

Abstract

We address the import of AI for philosophical logic and examine the requirements posed by knowledge representation and reasoning issues which has addressed, most especially through its logic programming field (LP), to more dynamic forms of logic, in order to deal with knowledge in flux. In the process, we consider its encroachments on the philosophy of knowledge.

Our aim is to entice the philosophically inclined reader, but also the LP veteran, to the emerging territories recently demarcated by ongoing research in LP knowledge representation and reasoning, by providing a *personal* bird's eye view of the region, and the flavour of a synthesis. The more technical parts of the exposition remain rather simple, and aim at an intuitive understanding, by means of motivating examples, of the general issues, how they are approached, and surmounted. The full-fledged technicalities are to be found in the references.

1 Introduction

We begin by addressing the general impingement of artificial intelligence (AI) on philosophical logic, by examining the requirements posed on logic by knowledge representation and reasoning issues which AI has addressed. We then outline some of AI's contributions, most especially via the field of logic programming (LP), to more dynamic forms of logic, in order to deal with knowledge in flux, namely: incomplete and contradictory information; hypotheses making through abduction; argumentation; diagnosis and debugging; updating; and learning. Along the way we delve into implications for the philosophy of knowledge.

[*]This chapter is a much extended and updated version of the paper titled "The Logical Impingement of AI", first published in [41].

[†]Centro de Inteligência Artificial (CENTRIA), Departamento de Informática, Universidade Nova de Lisboa, P-2829-516 Caparica, Portugal. email: lmp@di.fct.unl.pt, URL: http://centria.di.fct.unl.pt/~lmp/.

Our motivation is to entice the philosophically inclined reader, as well as the *LP* veteran, to the new territories recently landmarked in the logical continent by ongoing research in *AI* through LP, by providing a *personal* bird's eye view of the region. The technical parts of the exposition aim at an intuitive capturing of the general issues by rather simple examples, and of how they are tackled and overcome. Detailed formal treatment is deferred to the abundant references.

The structure of the chapter is as follows: The next section recapitulates and emphasizes the unique rôle of logic as a human knowledge representation and reasoning tool. The section that follows prolongs the one before to stress the benefit of mechanizing logic on the computer. Subsequently, the different encroachments of *AI* on logic are listed, and the last one, a framework whose fresh import for indexphilosophy philosophy is underscored, is selected for further analysis. In the sequel, some of the LP tools bearing on the proposed framework are brought out. Afterwards, the use of such tools for diagnosing a logical theory is outlined. In the section following that a dynamical logic framework is sketched, and then elaborated upon by summarily describing the *MENTAL* project. Finally, there are Concluding Remarks and, at the end, a challenge for philosophers.

2 Evolution and Reasoning

Evolution has provided humans with symbolic thought, and symbolic language communication abilities.

Objective common knowledge requires thought to follow abstract, content independent rules of reasoning and argumentation, which must not be entirely subjective, on pain of making precise communication and collective rational endeavour impossible. Such rules have become ingrained in human thought, and hold an enormous joint survival value[19, 22, 42].

In human cognitive evolution, both mimetic knowledge (such as that inherent in reality simulating maps and models), and imitation knowledge (such as that present in ritual observation, or in artifact reproduction), were essential first steps towards socially situated, joint rule following behaviour, required by, say, hunting plans. Subsequently, throughout human cultures, abstract rule following social games emerged.

Game rules encapsulate concrete situation defining patterns, and concrete situation-action-situation causal sequencing, which mirrors causality-obeying physical reality. From games, further abstraction ensued, and there finally emerged the notions of situation-defining concepts, of general rules of thought and their chaining, and of legitimate argument and counter-argument moves. Together they compose a cognitive meta-game[26].

The pervasiveness of informal logic for capturing knowledge and for reasoning, a veritable *lingua franca* across human languages and cultures, rests on its ability to actually foster rational understanding and common objectivity. Crucially, objective knowledge evolution dynamics itself, whether individual or plural, follows ratiocination patterns and laws.

Moreover, and more recently, the very same rules of reasoning can and are employed to reason about reasoning. Though some reasoning methods are well known, some are still unconscious but, like the rules of grammar, can be discovered through research. What is more, new reasoning methods can and have been invented and perfected throughout human history. Examples of these are transfinite induction, *reductio ad absurdum* (proof by contradicition), recursion, abduction, and contradicition removal, to name but a few.

New purported reasoning methods may be disputed, just like any specific train of reasoning can. But reasoning can only be disputed by further reasoning, if any consensus is to be found! (cf.[38]). Some argue that scientific and philosophical discussion is necessarily tantamount to a culture sensitive, and culturally relative, persuasive informal *ad hoc* argumentation, allied to anything goes rethoric[25]. They ignore that argumentation is just another a form of reasoning which has itself been made the subject of logical formalization, and are oblivious to that rethoric may be fine for preachers, but is not conducive to the two-sided communication required to reach common agreement in the all rigorous scientific praxis that lead to cumulative knowledge.

3 Logic and the Computer

Logic, we have seen, has arisen as the content independent formulation of the Laws of Thought. Predicates express whatever conceptual relations we may wish, about the external and internal worlds. These predicate building blocks can then be combined into formulas, by means of quantifiers and logical connectives, and can be manipulated according to inference rules characterizing divers forms of reasoning.

Logic is capable of articulating an extensional behaviouristic view of predicates, seen as black boxes or input/output relations, simultaneously with an intensional view of predicates, seen as functionally decomposable into other predicates, like black boxes wired inside black boxes[37].

Since the language of logic is symbolic and its rules content-independent, its workings can be specified by general, abstract, rule following procedures. These, in turn, can be programmed on the computer. Through their mechanization, logical theories and reasoning forms attain, for the first time, an *in vitro* existence, an ability and availability for repeatable execution, independently of any human mind hardware support[40].

The two views above, extensional and intensional, are reconciled in the computer by insisting that the declarative semantics, the "what" is to be computed, be equivalent to the procedural computational semantics, the "how" it is computed. This is the cornerstone of the logic programming paradigm, of which more later [1].

4 Philosophy and AI

Philosophy has been a cradle for rational thought throughout human intellectual history. Its study of language fostered the refinement of logic as an abstraction of natural language and as a tool of linguistic inquiry. The investigations on the foundations of mathematics too assigned to logic a meta-instrumental rôle that would deploy it as the general purpose symbol manipulating tool *par excellence*. Computer science as well has adopted logic as its general foundational tool, while *AI* has made viable the proposition of turning it into a *bona fide* computer programming language.

At the same time, *AI* has developed logic beyond the confines of monotonic cumulativity, typical of the precise, complete, endurable, condensed, and closed mathematical domains, in order to open it up to the non-monotonic real world domain of imprecise, incomplete, contradictory, arguable, revisable, distributed, and evolving knowledge. In short, *AI* has added dynamics to erstwhile statics.

In *AI*, as with any science, common logic has an important rôle in it. Indeed, any science, implicitly or explicitly, presses logic and reasoning into its service, to build up arguments, counter-arguments, and to settle disputes. But a greater rôle can be expected of *AI* in logic!

- First, *AI* means to mechanize logic, it being such a core tool for so many rational activities.

[1] Henceforth, Science and Technology, with the support of Computer Science and most especially of AI, will more and more see automated, in symbiosis with Man, their own processes of creation, in positive retroaction!

- Second, AI intends to make explicit, and well-defined, the unconscious logic we use, and put it to objective test by automating it on the computer.

- Third, AI employs logic as a generic language of communication, of knowledge and of procedures, between humans and computers, and among computers themselves.

- Fourth, in AI, even when procedures and devices are not implemented using logic directly, logic can play the rôle of a precise specification language for their requirements, and of a formalism with which to study their semantical properties.

- Fifth, AI has contributed significantly to the phrasing and examination of the problem of identifying limits to symbolic reasoning embodied in computers, and whether these limits apply to humans as well.

- Sixth and final rôle, AI has helped researchers explore new reasoning issues and methods, and to combine disparate reasoning modalities into a uniform unified framework, so as to deal with incomplete, imprecise, contradictory, and changing information.

This last rôle, in contrast with its predecessor, much pampered by philosophers, has hardly caught their attention yet. The latter-day epigenics of logic it announces, and its computational scaffolding, point to an innovative rôle for the logic enterprise. To wit, that of supporting, justifying, and mechanizing the dynamics of the abstract epistemic and argumentative processes subjacent to scientific endeavour itself. And this, I believe, is good news for philosophers of science.

Consequently, it is to that that we now turn, in the hope that philosophers will be who shall help it mature into a full-blown computational epistemology.

5 A Logical Tool of AI

Classical logic has been developed to study well-defined, consistent, and unchanging mathematical objects. It thereby acquired a static character. AI needs to deal with knowledge in flux, and less than perfect conditions, by means of more dynamic forms of logic. Much of this has been the focus of research in logic programming, a field of AI which uses logic directly as a programming language[2], and provides specific implementation methods and efficient working systems to do so[3]. logic programming is moreover much used as a staple implementation vehicle for other AI approaches to logic.

What are the issues and requirements of the dynamics of logic which AI, namely via logic programming, has contributed to solve?[4]

5.1 Logic Programming

There is a close connection between logical implication and physical causality. After all, logic permits us to model the goings on in the world. This is even more so in the case of Logic Programming, where all true conclusions must be supported, "caused", by some premises, and where implication is unidirectional, i.e. not contrapositive: "causes" do not run backwards. So-called Horn clause notation is used to express this directionality. This ability for logic programming to model actions by means of updates is brought out in the section on updating.

To produce a result or conclusion C we need a conjunction of enough positive conditions P to sustain it, conjoined by ',' to the absence or negation of a conjunction $\neg N$ of all the negative conditions that would prevent it, given P:

$$C \leftarrow P, \neg N$$

If there is more than one way to obtain C, then we have several rules of this form:

$$C \leftarrow P_i, \neg N_i$$

If our information about each rule, and about the whole set of rules for C, is complete, then we would have:

$$C \longleftrightarrow \bigvee_i P_i, \neg N_i$$

But what if we don't? First, we might not have enough information about each $\neg N_i$. We might conceivably know about each required P_i, without at least one of which C can never be achieved. But regarding the $\neg N_i$, i.e. all the conditions which, if present, would prevent us from concluding C in the presence of each P_i, that's more difficult.

[2] Cf. [28] for the foundations and [7] for a recent state-of-the art.

[3] For the most advanced, incorporating more recent theoretical developments, see the XSB system at: //www.cs.sunysb.edu/~sbprolog/xsb-page.html/.

[4] For a detailed technical treatment of the ensuing issues cf. our book[7], and references therein, and the more recent pointers in this text.

5.2 Open worlds

Too many things can go wrong in an open non-mathematical world, some of which we don't even suspect. For all we know, some bomb might go off that destroys the whole physical setup we're trying to model logically (at a safe distance). Must we model that too? In the real world, any setting is overly complex enoughtoo complex already for us to define it exhaustively each time. We have to allow for unforeseen exceptions to occur, based on new incoming information. Thus, instead of having to make sure or prove that some condition N_i is not present, we can assume it is not, on condition that we are prepared to accept subsequent information to the contrary. I.e. we may assume a more general rule than warranted, but must henceforth be prepared to deal with arising exceptions.

Take for example, with the obvious reading, this piece of knowledge, again in Horn clause form:

$$faithful(H, K) \leftarrow married(H, K), \neg lover(H, L)$$

We don't normally have explicit information about who is or is not the lover of whom, though that kind of information may arrive unexpectedly, especially of presidents[5]. Thus, we can code our knowledge on presumption that if someone H is married to K, and there is no information or basis to conclude $lover(H, L)$, then H is faithful to K:

$$faithful(H, K) \leftarrow married(H, K), not\ lover(H, L)$$

This is expressible via so-called default default negation *not* P, which may be read as "P is not provable". That is, we no longer require proof that $\neg lover(H, L)$ for any L, given some H, but will assume so unless we can establish $lover(H, L)$ for some L given H. In other words, if I have no evidence to conclude $lover(H, L)$ for some L given H , I can assume it false for all L given H.

5.3 Defeasible Assumptions

default negation was introduced by AI researchers via Logic Programming, and may be read, interchangeably, as "P is not provable", or "the falsity of P is assumable", or "the falsity of P is abducible", or "there is no evidence for P", or "there is no argument for P". Default negation allows us to deal with

[5]At the time of writing this example I was in the USA, and the American president was being insistently harassed about a purported love affair.

lack of information, a common situation in the real world. It introduces non-monotonicity into knowledge representation. If later we're informed about a pair of lovers, we must then go back on our previous conclusion about whatever faithfulness we had presumed. Indeed, conclusions might not be solid because the rules leading to them may be . Legal texts, regulations, and courts employ this form of negation abundantly, as they perforce deal with open worlds, though it had not before been formalized in logic till AI too discovered its need of it.

5.4 Closed World Assumption

Mark that *not* should grant positive and negative information equal standing. That is, we should be able to write:

$$\neg faithful(H,K) \leftarrow married(H,K), not\ \neg lover(H,L)$$

to model instead a world where people are unfaithful by default or custom, and where it is required to explicitly prove that someone does not take any lover before we conclude that person not unfaithful. The issue arises because often, particularly in data bases and knowledge bases, the (CWA) in enforced. That is, the CWA obtains when we presume the database or knowledge base to have all the pertinent positive information, so that only what is explicitly stated, or explicitly derivable from it, is true; otherwise, it is presumed false. But how could it, in general, have all the pertinent information in a changing world?

Information is normally expressed positively, by dint of mental and linguistic economics. So, by CWA, the absent, non-explicitly obtainable information is usually the negation of positive information. Which means, when no information is available about lovers, that $\neg lover(H,L)$ is true by CWA, whereas $lover(H,L)$ is not. This asymmetry is undesirable, inasmuch predicate names are purely conventional and we could, as justifiably, have come up with positive predicate $non - lover(H,L)$ and its negation $\neg non - lover(H,L)$ to model our knowledge. Though in general P is considered the positive predicate if its extension greater than its negative counterpart, we might not know beforehand which is the greater one. A logical language should be indifferent with respect to such knowledge representation accidents.

5.5 Explicit Negation

In addition, when we have no factual or derivable, either positive or negative, information we'd like to be able to say that both are false epistemically, i.e. from the "knowledge we possess" point of view. Accordingly, the excluded middle postulate is unacceptable because some predication or other and its negation may be false simultaneously.

The CWA and the symmetry and epistemologic requisites, can be reconciled by reading '$\neg$' above not as classical negation, which complies with the excluded middle postulate stating that any predication is either true or false, but as yet a new form of negation, dubbed in logic programming "explicit negation"[10] (which ignores the excluded middle provision). Now we can state the CWA for just those predicates P or $\neg Q$ we wish, simply by writing:

$$\neg P \leftarrow not\ P \quad \text{or} \quad Q \leftarrow not\ \neg Q$$

Alternatively, the use of a *not* P or of a *not* $\neg Q$ assumption may be made at just those predicate occurrences requiring them, like we did before.

5.6 Revising Assumptions

Let us next examine the need for revising assumptions and for introducing a third truth-value, call it "undefined", into our framework.

When we combine the viewpoints of the two above worlds we become confused:

$$faithful(H, K) \leftarrow married(H, K), not\ \neg lover(H, L)$$

$$\neg faithful(H, K) \leftarrow married(H, K), not\ lover(H, L)$$

Assuming $married(H, K)$ for some H and K, it now appears that both $faithful(H, K)$ and $\neg faithful(H, K)$ are contradictorily true. Because we have no evidence for $lover(H, L)$ nor $\neg lover(H, L)$, there simply is no information about them, we make two assumptions about their falsity. But when an assumption leads to contradicition one should retract it. Yes, it is the venerable principle of *reductio ad absurdum, or* " reasoning by contradicition". In our case, two assumptions led to the contradicition. Which shall we retract? They are on equal footing!

5.7 Undefinedness

Given no other eventually preferential information, we retract both because we cannot justly decide between them. That is, we assume neither $lover(H, L)$ nor $\neg lover(H, L)$ false. Since neither is provably true also, we make each *undefined*. I.e. we introduce a third truth-value to better characterize this lack of information about some lovers' situation, thereby making $faithful(H, K)$ and $\neg faithful(H, K)$ undefined too. This imposition of undefinedness can be achieved simply, by adding to our knowledge:

$$\neg lover(H, L) \leftarrow not\ lover(H, L)$$

$$lover(H, L) \leftarrow not\ \neg lover(H, L)$$

Given no other information, we cannot prove either of $lover(H, L)$ or $\neg lover(H, L)$ true, or false. Any attempt to do so runs into a self-referential circle involving default negation. However, once we do hypothesize the one true the other perforce becomes false, and vice-versa. These two possible situations are not thus ruled out. But the safest, skeptical, third option is to take no side in this marital dispute, and abstain from believing either.

Indeed, the well-founded semantics of logic programs (WFS) assigns to the literals in the above two clauses the truth value *undefined* in its knowledge skeptical well-founded model, and allows also for the other two, non truth-minimal, more credulous models.

But why can we not simply add:

$$lover(H, L)\ \vee \neg lover(H, L)\ \ ?$$

Because it will not do. We would still not be able to prove either $lover(H, L)$ or $\neg lover(H, L)$ definitely true, and so the contradicition would ensue all the same![6]

When dealing with non-provability one really needs a third truth-value to express our epistemic inability to come up with information. Even assuming the world is ontologically 2-valued, our access to information coded about it might be 3-valued in general. Besides, the world may very well not be 2-valued, or, in any case, not capturable in a 2-valued way. Can we say of some particle/wave that it is either here or not here? Is it really either here or not here, though we cannot say it? The inherent dispute apparently cannot be settled experimentally.

[6] For a survey on paraconsistent *LP* semantics cf. [14].

In any case we are in wont of a third logical value for other reasons. As we build up our real world imperfect knowledge base, we may very well create, unwittingly and unawares, circular dependencies as above. For example, the Legislator may well enact conflicting, circular, laws. Still, we want to be able to carry on reasoning, whether or not such circularities legitimally express what they model. And when they do not, we want to detect and function with them rather than throwing away the baby with the bath water.

But undefinedness may also be legitimally pressed into service to express common knowlege, in its variously credulous and skeptical readings. Why, even yesterday I read in a book[12] of the old sailor proverb that "He weathers the storms he cannot avoid, and avoids the storms he cannot weather", i.e.:

$$weather \leftarrow storm,\ not\ avoid$$

$$avoid \leftarrow storm,\ not\ weather$$

What will happen to an experienced sailor in a storm? Will he weather it or avoid it? Three outcomes are possible, including being undecided about the outcome. For us, it means we must weather circular knowledge, not avoid it in the calm waters of the artificial mathematical paradises...

5.8 Expressiveness

The introduction of this 3-valued semantics into logic programming is an important innovation of AI, with a number of consequences. Namely, we have seen, it allows us to be skeptical and suppose no more than is warranted.

Another innovation, to recapitulate, is that the negation $\neg$ used above, called explicit negation, is 3-valued too, and so differs from classical negation. Explicit negation, we've seen, does not conform to the principle of excluded middle. Predications and their negations can be both false ("the chair is angry", "the chair is not angry"). In a knowledge base, we may definitely not be able to prove either something or its (explicit) negation, so that both become false by default.

Additionally, the introduction of explicit negation, and in particular its combination with default negation, provides for more expressive knowledge representation. Consider for instance the injunction

$$\neg cross \leftarrow train$$

versus the alternative injunction

$$\neg cross \leftarrow not\ \neg train$$

The latter is clearly a safer option. According to it you must actually prove that a train is not coming before $\neg cross$ becomes false, whereas with the first option $\neg cross$ becomes false solely on the grounds that you fail to prove a train is coming!

Yet another innovation is that the implicational arrow $\leftarrow$, when combined with default or explicit negation, is not material implication. Contrapositives are not countenanced. Consider:

$$\begin{array}{lcl} A & \leftarrow & B \\ & & \neg A \end{array}$$

$\neg B$ does not follow, for there is no presumption of the contrapositive $\neg B \leftarrow \neg A$. Whenever contrapositives are desired they must be explicitly added. Instead, $\leftarrow$ should be seen as expressing an inference rule, which can be used procedurally "bottom-up" to conclude A given B, or "top-down", like an invoked procedure, to try and prove the body B in order to prove the head A. Facts are just procedures with empty body. This inferential reading of clauses as rules turns $\leftarrow$ into a directional operator, which makes it rather appropriate to model not only inference but causality as well.

Indeed, the semantics of logic programs emphasizes exactly this, with their insistence on admitting only minimal models. Minimal models are those for which every positive or explicitly negated literal *true* in the model is supported on some rule whose head or conclusion is the literal, and whose body or set of premises is in turn supported. Thus facts, since they have an empy body, are automatically supported.

Additionally, any positive or any explicitly negated literal is *false* just in case all rules for it have a false body. Consequently, if there are no rules whatsoever for a literal it is automatically false.

All other positive or explicitly negated literals are *undefined* (this can happen only if they are involved in unresolved self-referential loops through default negation). Finally, a default negated literal, *not* P, is *true/ false/ undefined* just in case P is, respectively, *false/ true/ undefined.*

5.9 Theory Diagnosis

Many examples exist of the use of default negation, explicit negation, contradicition removal, and 3-valuedness, made possible by research in Logic

Programming, namely in the areas of abduction, argumentation, belief revision, knowledge updating, learning, and diagnosis[7, 24]. Let us illustrate the latter, i.e. the diagnosis or debugging of a generic knowledge base. Its connections with the other aforementioned areas allow us to brush on them too.

Suppose we have the following rules, expressing a logic program that allows us to compute predicate C on the basis of predicates P and N:

$$C(X) \leftarrow P(X), \neg N(X)$$

$$\begin{array}{ll} P(a) & \neg N(a) \\ P(b) & \\ P(c) & \neg N(c) \end{array}$$

Let us allow for the chance that the rule for C is conceivably wrong. i.e. that it may have exceptions, by writing it thus:

$$C(X) \leftarrow P(X), \neg N(X),\ \mathit{not\ exception}(C(X))$$

If nothing else is stated about the *exception* predicate, the conclusions of the logic program remain unchanged, so the program may contain these rules from the start, for any predicate we wish.

If we next learn that $C(a)$ is false, we can salvage the rule for C, since it is no longer valid for all $X = a$, simply by adding to our knowledge:

$$\mathit{exception}(C(a))$$

The rule for C will continue to work for the remaining cases, though more exceptions might subsequently be accumulated.

This method takes care of rules that are unsound, i.e. produce wrong results.

What about the other problematic case, that of incompleteness, i.e. when results that would be correct are missing? Suppose we learn that $C(b)$ should true, but which our rules rule out? Well, it is enough to introduce for every set of rules for a predicate a catch-all rule. In this case:

$$\begin{array}{rcl} C(X) & \leftarrow & \mathit{missing}(C(X)) \\ P(X) & \leftarrow & \mathit{missing}(P(X)) \\ \neg N(X) & \leftarrow & \mathit{missing}(\neg N(X)) \end{array}$$

If nothing else is stated about the *missing* predicate, the conclusions of the logic program remain unchanged, so the program may contain these rules from the start. Now, to make $C(b)$ true, it now suffices now to abduce, or adopt, one of two hypotheses: that $missing(C(b))$ or that $missing(\neg N(b))$ is true.

This method takes care of rules that are incomplete, i.e. fail to produce results.

We have achieved generality. By making the two predicates, $exception(\_)$ and $missing(\_)$, abducible ones, we can diagnose and debug a knowledge base expressed in logic.

abduction is a well-known reasoning process by which one can adopt hypotheses to the effect that some predicate instances (or their negations) are true in order to prove some conclusion. It's reasoning from goals to requirements, and has been extensively studied in the logic programming context[7]. Of course, the assumptions or hypotheses so adopted may later prove erroneous, if and when they lead to contradicition. We have then to make provision for the application of a contradicition removal process based on revising assumptions, possibly adopting of some other assumptions instead. This too has been extensively studied in AI and in the logic programming setting, and as a result automated reasoning systems have been made available which do that work for us[1, 13] and have been applied to real domains[23].

5.10 Updating

Last but not least, work in logic programming has concerned itself with the updating of a knowledge base by another one. This notion of knowledge updating, as opposed to that of simple fact updating, opens up another dimension to the dynamics of logic, in contradistinction to the statics of the logic of old. Given an existing knowledge base, containing facts and rules, and some new knowledge with facts and rules as well, possibly contradicting the previous knowledge when added to it, what is the resulting updated knowledge base? Can this process be iterated?

Most of the work conducted so far in the field of logic programming has focused on representing *static* knowledge, i.e., knowledge that does not evolve with time. This is a serious drawback when dealing with *dynamic knowledge bases* in which not only the *extensional* part (the set of facts) changes dynamically but so does the *intensional* part (the set of rules). Recent work has shown how this can be achieved with generality[2, 9]. Application areas in point are when the two knowledge bases consist in pieces

[7] For an overview, cf.[27]. For an implementation, cf. [11].

of legislation, or regulations, or safety procedures, or rules of robot conduct.

The introduction of dynamic program updates extends program updates to ordered sets of logic programs (or modules). When this order is interpreted as a time order, dynamic program updates describe the evolution of a logic program which undergoes a sequence of modifications. This opens up the possibility of incremental design and evolution of logic programs, leading to the paradigm of *dynamic logic programming* (*DLP*). This new paradigm significantly facilitates modularization of logic programming, and thus modularization of non-monotonic reasoning as a whole.

Specifically, suppose that one is given a set of logic program modules, each describing a different state of our knowledge of the world. Different states may represent different time points or different sets of priorities or perhaps even different viewpoints. The rôle of the dynamic program update is to employ the mutual relationships existing between different modules to precisely determine, at any given module composition stage, the declarative as well as the procedural semantics of the combined program resulting from the modules[4]. However, ordered sets of program modules need not necessarily be seen as just a temporal evolution of a logic program. Different modules can also represent different sets of priorities, or viewpoints of different agents. In the case of priorities, a dynamic program update specifies the exact meaning of the *"union"* of the modules, subject to the given priorities.

In order to represent negative information, *generalized logic programs* are employed in updating. These allow default negation not only in rule bodies but also in their heads. This is needed, in particular, in order to specify that some atoms should became false, i.e., should be deleted. However, such updates are far more expressive than a mere insertion and deletion of facts. The updates can be specified by means of arbitrary program rules, and thus the updates themselves are logic programs. Consequently, the approach empowers one to update one generalized logic program P (the initial program) by another generalized logic program U (the updating program), obtaining as a result a new, updated logic program $P \oplus U$. As a consequence, program modules may contain mutually contradictory as well as overlapping information.

The common intuition behind the update of a model of the world has been based on what is referred to as the commonsense law of inertia, i.e. things do not change unless they are expressly made to do so. Suppose, for example, that we have a model in which "sunshine" is true and "rain" is false; if later we receive the information that the sun is no longer shining, we conclude that "sunshine" is false, due to the update, and that "rain" is still false by inertia.

Suppose now that our vision of the world is described instead by a logic program and we want to update it. Is updating each of its models enough? Is all the information borne by a logic program contained within its set of models? The answer to both these questions is negative. A logic program encodes more than the set of its individual models. It encodes the relationships between the elements of a model, which are lost if we envisage updates simply on a model by model basis, as proposed by revision programs. In fact, while the semantics of the resulting knowledge base after an update indeed represents the intended meaning when just the extensional part of the original knowledge base (the set of facts) is being updated, it leads to strongly *counter-intuitive* results when also the intensional part of the database (the set of rules) undergoes change, as the following example shows.

Consider the logic program P :

$$sleep \leftarrow not\ tv\_on$$

$$tv\_on \leftarrow$$

$$watch\_tv \leftarrow tv\_on$$

Clearly $M = \{tv\_on, watch\_tv\}$ is its only stable model. Suppose now that the update U states that there is a power failure, and that if there is a power failure then the TV is no longer on, as represented by the logic program U:

$$not\ tv\_on \leftarrow power\_failure$$

$$power\_failure \leftarrow$$

With simple model updating, we would have $M_U = \{power\_failure, watch\_tv\}$ as the only update of M by U. This is because $power\_failure$ needs to be added to the model and its addition forces us to make tv_on false. As a result, even though there is a power failure we are still watching TV.

However, by inspecting the initial program and the updating rules, we are likely to conclude that since $watch\_tv$ was true only because tv_on was true, the removal of tv_on should make $watch\_tv$ false by default. Moreover, one would expect $sleep$ to become true as well. Consequently, the intended model of the update of P by U is the model $M_{U'} = \{power\_failure, sleep\}$. Suppose now that another update U_2 follows, described by the logic program:

$$not\ power\_failure \leftarrow$$

stating that the power is back up again. We should now expect the TV to be on again. Since power was restored, i.e. $power\_failure$ is false, the rule $not\ tv\_on \leftarrow power\_failure$ of U should have no effect, and the truth value of tv_on should be obtained by inertia from the rule $tv\_on \leftarrow$ of the original program P.

This example illustrates that, when updating knowledge bases, it is not sufficient to just consider the truth values of literals figuring in the heads of its rules, because the truth value of their rule bodies may also be affected by the updates of other literals. In other words, it suggests that the *principle of inertia* should be applied not just to the individual literals in an interpretation but rather to the *entire rules of the knowledge base.*

Newton's first law, also known as the law of inertia, states that: *"every body remains at rest or moves with constant velocity in a straight line, unless it is compelled to change that state by an unbalanced force acting upon it"* (adapted from [39]). One often tends to interpret this law in a commonsensical way, as things keeping as they are unless some kind of force is applied to them. This is true but it doesn't exhaust the meaning of the law. It is the result of all applied forces that governs the outcome. Take a body to which several forces are applied, and which is in a state of equilibrium due to those forces canceling out. Later, one of those forces is removed and the body starts to move.

The same kind of behaviour presents itself when updating programs. Before obtaining the truth value, by inertia, of those elements not directly affected by the update program, one should verify whether the truth of such elements is not indirectly affected by the updating of other elements. That is, the body of a rule may act as a force that is sustaining the truth of its head, but this force may be withdrawn if and when the body becomes false. Accordingly, the correct way to view program updating, and in particular the rôle of inertia, is to say that it is the rules of the initial program that carry over to the updated program, due to inertia (instead of the truth of literals) just in case they are not overruled by the updating program. This should be so because rules encode more information than just their models.

This approach was first adopted in [34], where the authors presented a program transformation which, given an initial program and an update program, produces an updated program obeying rule inertia. The stance of the dynamic logic programming approach is precisely that of ensuring that any added update rules are indeed in force, and that previous program rules are still in force (by inertia) only in so far as possible, i.e. they are kept for as long as they do not conflict with any newly added ones.

To represent negative information in logic programs and their updates *DLP* allows for the presence of default negation in rule heads. It is worth noting why, in the update setting, generalizing the language to allow default negation in rule heads is more adequate than introducing explicit negation in programs (both in heads and bodies). Suppose we are given a rule stating that A is true whenever some condition $Cond$ is met. This is naturally represented by the rule $A \leftarrow Cond$. Now suppose we want to say, as an update, that A should no longer be the case (i.e. should be deleted or retracted), if some condition $Cond'$ is met. How to represent this new knowledge?

By using extended logic programming (with explicit negation) this could be indicated by $\neg A \leftarrow Cond'$. But this rule says more than we want to. It states that A is false upon $Cond'$, and we just want to go as far as to say that the truth of A is only to be deleted in such a case. All is wont said is that if $Cond'$ is true then *not* A should be the case, i.e. *not* $A \leftarrow Cond'$. The difference between explicit and default negation is fundamental whenever the information about some atom A cannot be assumed to be complete, as we have seen before. Under these circumstances, the former means that there is evidence for A being false, while the latter means that there is no evidence for A being true. In the deletion example, we desire the latter case.

In other words, a *not* A head means A is deleted if the body holds. Deleting A means that A is no longer true, not necessarily that it is false. When the CWA is adopted as well, then this deletion causes A to be false. In the updates setting, the CWA must be explicitly encoded from the start, by making all *not* A literals false in the initial program being updated. That is, the two concepts, deletion and CWA, are orthogonal and must be separately incorporated. In the *Stable Models* and *Well-Founded* semantics of single generalized programs, the CWA is adopted *ab initio*, and default negation in the heads is conflated with non-provability because there is no updating, and thus no deletion. Note that in updates the head *nots* cannot be moved freely into the body, to obtain simple denials, for two reasons: first, this manoeuvre, although allowed in a 2-valued semantics does not carry over to a 3-valued one; second and more importantly, there is inescapable pragmatic information in specifying exactly which *not* literal figures in the head, namely the one being deleted when the body holds true, and thus it is not indifferent that any other (positive) body literal in the denial could be moved to the head.

In [3], a Language of Updates (*LUPS)* has been proposed, expressly designed for specifying sequences of updates to logic programs. Even though the main motivation behind the introduction of *DLP* was to represent the evolution of knowledge in time, the relationship between the different states can encode other aspects of a system of knowledge, as pointed out above. In fact, since their introduction, *DLP* and *LUPS* have been employed to represent several distinct aspects, namely as a way to:

- represent and reason about the evolution of knowledge in time [4, 6];
- combine rules learned by agents [32];
- reason about updates of agents' beliefs [20];
- model agent interaction [43, 44];
- model and reason about actions [5, 6];
- resolve inconsistencies in metaphorical reasoning [36];
- combine updates and preferences [8].

The common aspect among these applications of *DLP* is that the states associated with the given set of theories only encode one of the several possible representational dimensions (e.g. time, hierarchies, domains,...). This is so because *DLP* is only defined for linear sequences of states. For example, *DLP* can be used to model the relationship of a hierarchically related group of agents, and *DLP* can be used to model the evolution of a single agent over time. But *DLP*, as it stands, cannot deal with both settings at once, and model the evolution of such a group of agents over time.

An instance of such a multi-dimensional scenario can be found in legal reasoning, where the legislative agency is divided conforming to a hierarchy of power, governed by the principle *Lex Superior* (Lex Superior Derogat Legi Inferiori) according to which the rule issued by a higher hierarchical authority overrides the one issued by a lower one, and the evolution of law in time is governed by the principle *Lex Posterior* (Lex Posterior Derogat Legi Priori) according to which the rule enacted at a later point in time overrides the earlier one.

In effect, knowledge updating is not to be simply construed as taking place in the time dimension alone. 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* has been extended to allow a more general structure of states.

Multi-dimensional Dynamic logic programming (*MDLP*) [35] generalizes *DLP* to allow for collections of states represented by arbitrary acyclic digraphs, not just sequences of states. *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, *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, *MDLP* provides a "societal" viewpoint in logic programming, important in these web and agent days, for combining knowledge in general.

updating inevitably raises issues about revising and prefering, and some work is emerging on the articulation of these distinct but highly complementary aspects. And learning is usefully seen as successive approximate change, as opposed to exact change, and combining the results of learning by multiple agents, multiple strategies, or multiple data sets, inevitably poses problems within the province of updating. Finally, goal directed belief revision can be fruitfully construed as abductive updating.

Thus, not only do the aforementioned topics combine naturally together – and so require precise, formal, means and tools to do so – , but their combination results in turn in a nascent complex architectural basis and component for logic programming rational agents, which can update one another and common, structured, updatable blackboard agents. It can be surmised, consequently, that the fostering of this meshing of topics within the Logic Programming community is all of opportune, seeding, and fruitful. Indeed, application areas such as software development, multi-strategy learning, abductive belief revision, model-based diagnosis, agent architecture, and others, are being successfully pursued while employing precisely this outlook.

6 A Dynamical Logic 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.

The above sketch of a dynamical logical framework is being made concrete within the context of our *MENTAL* agents project[8], at the Centro de Inteligência Artificial, Universidade Nova de Lisboa.

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.

The aim of the *MENTAL* project is that of establishing, on a sound theoretical basis, the design of an overall 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 [33]. 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-

[8]Portuguese research funding programme *PRAXIS XXI*, project *MENTAL* (2/2.1/TIT/1593/95), http://centria.di.fct.unl.pt/~ica/mental/.

level layer that ensures internal task distribution and belief revision, sub-agent 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[9].

The use of logic programming for the overall endeavour is justified on the basis 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.

7 Concluding Remarks

How can the general reasoning issues explored here be approached, within a uniform framework, except in the context of logic? I submit indeed that logic is the only conceivable venue for the core of such an enterprise. And must we not do it? Must we not mechanize reasoning procedures so that

[9]This new approach is reported in [30, 31].

robots and computers can perform, for us or with us, the sometimes tedious sometimes overly complex work involved in them? Surely, only so persisting will permit the grappling of the evermore demanding reasoning problems that face us.

I hope to have convinced you that *AI*, most especially through logic programming, will continue to accomplish a good deal in identifying, formalizing, and implementing the Laws of Thought. Most notably, *AI* has taken on the challenge of opening up logic to the dynamics of knowledge in flux. And in so doing, it has been progressively meeting our expections and requirements. Continuing this work is essential if we are to cope, albeit with the aid of computers, with the challenges of evermore accumulated and distributed knowledge, in a changing world.

A wealth of investment in logic programming research has accumulated in the form of published results and built systems, over the more than 25 years since its inception. Some of these could not be brought to full fruition at the time because of technological costs and funding limitations, or more simply because of the lack in availability of cheap and efficient requisite equipment, nowadays having become affordable and performant. So, though the theory and implementational know-how were there at the time, their full fruition was then not viable.

However, to-day, younger researchers are not well aware of such past results and of their potential, some from the early eighties, even when the implementational technology may have evolved to allow them to be put to use[10]. And all the more so because the complexity of present systems makes such immediately available "theory technologies" all the more desirable. On the other hand, the cost of memory and the appearance on the scene of new "implementation technologies", such as (theory-driven) tabling, make the prior investment in theory not only realizable now but affordable and profitable as well.

Conversely, we should not stop or refrain from investing now for the future, for the "theory technologies" of today, in addressing real felt problems, will be the providers of tomorrow's implementational and applicational reapers.

For instance, were it not for the impressive development in Logic Programming semantics in the past 12 years or so, and its extensions to non-monotonic and other forms of reasoning, we would not have today the impressive, sound, and efficient system implementations of such semantics (be

[10]Illustrative case examples in point are those of "intelligent backtracking" and of "declarative debugging".

it the stable or well-founded varieties or their hybrid combination), which have been opening up a whole new gamut of application areas as well as a spate of sophisticated reasoning abilities over them. This was made possible only through successive and prolonged efforts at theoretical generalization and synthesis, and by way of the combined integration of theory, procedure development, and practical implementation.

The lesson, then, for further investing in this still emerging information technology, is that it must be continual: both for the fruition of past investment as for the seeding of timely future delectation.

The logic programming paradigm provides a well-defined, general, integrative, encompassing, and rigorous framework for systematically studying computation, be it syntax, semantics, procedures, or attending implementations, environments, tools, and standards. *LP* approaches problems, and provides solutions, at a sufficient level of abstraction so that they generalize from problem domain to problem domain. This is afforded by the nature of its very foundation in logic, both in substance and method, and constitutes one of its major assets.

Indeed, computational reasoning abilities such as assuming by default, abducing, revising beliefs, removing contradicitions, updating, belief revision, learning, constraint handling, etc. etc., by dint of their generality and abstract characterization, once developed can readily be adopted by, and integrated into, distinct topical application areas.

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 in previous sections. Forthcoming rational agents, to be realistic, will require an admixture of any number of them to carry 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?

Moreover, the issue is not just that of conceptual integration but that of conceptual cross-fertilization too. How can one employ learning in the service of belief revision? How does one remove contradictions from updates? How can learnt preferences be combined with preference updates to guide belief revision in a rational agent? How may fuzzy preferences be revised in the light of ongoing abduced assumptions and constraints? How can probabilistic and multi-valued logic programs be combined?[11]

Participation by philosophers in this venture can take the form of partaking in the development of the conceptual underpinnings of the required computer implemented logic tools, and of employing such tools for the computer functional rational reconstruction of historical epistemological argument[12], and of scientific theory evolution duplication, by automated learning. Evolving taxonomic classification of natural kinds might be a good starting point.

8 Acknowledgements

This work was mostly supported by PRAXIS XXI project MENTAL (2/2.1 /TIT/1593/95), and by NATO scholarships while the author was on sabbatical leaves at the Department of Computer Science, University of California, Riverside. Abundant thanks are due to the several co-authors of joint work mentioned.

References

[1] José Júlio Alferes, Carlos V. Damásio and Luís Moniz Pereira, "A logic programming system for non-monotonic reasoning". Journal of Automated Reasoning, 14:93-147, 1995.

[2] José Júlio Alferes, João A. Leite, Luís Moniz Pereira, Halina Przymusinska and Teodor C. Przymusinski, "Dynamic logic programming", Int. Conf. on Knowledge Representation and Reasoning (KR'98), Trento, Italy. Stuart Shapiro (ed.), Morgan Kaufmann, 1998.

[3] José Júlio Alferes, Luís Moniz Pereira, H. Przymusinska and T. C. Przymusinski, "LUPS - a language for updating logic programs". In M. Gelfond, N. Leone and G. Pfeifer (eds.), Procs. of the 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning, El Paso, Texas USA, pages 162-176, LNAI 1730, Springer, 1999. An extended updated version has been accepted by the AI journal as "LUPS - a language for updating logic programs".

[4] José Júlio Alferes, João A. Leite, Luís Moniz Pereira, Halina Przymusinska and Teodor C. Przymusinski, "Dynamic Updates of Non-Monotonic

[11] Cf. [15, 16, 17, 18].
[12] As can be found for instance in [29].

Knowledge Bases", The Journal of Logic Programming 45(1-3): 43-70, September/October 2000.

[5] José Júlio Alferes, João A. Leite, Luís Moniz Pereira, Paulo Quaresma, "Planning as Abductive Updating". In D. Kitchin (ed.), Procs. AISB'00 Symposium on AI Planning and Intelligent Agents, pages 1-8, AISB, Birmingham, England, April 2000.

[6] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, Teodor C. Przymusinski and Paulo Quaresma, "Dynamic Knowledge Representation and its Applications". In Stefano Cerri and Danail Dochev (eds.), Procs. of The 9th Int. Conf. on Artificial Intelligence - Methodology, Systems, Applications (AIMSA'00), Varna, Bulgaria, LNAI 1904, Springer, pages 1-10, 2000.

[7] José Júlio Alferes and Luís Moniz Pereira, "Reasoning with logic programming", Lecture Notes in Artificial Intelligence, vol. 1111, Springer, 1996.

[8] José Júlio Alferes and Luís Moniz Pereira, "Updates plus Preferences". In M. O. Aciego, I. P. de Guzmán, G. Brewka and L. M. Pereira (eds.), Logics in AI, Procs. JELIA'00, Málaga, LNAI 1919, pages 345-360. Springer, 2000.

[9] José Júlio Alferes and Luís Moniz Pereira, "Logic programming updating: a guided tour". In "Essays in honour of Robert Kowalski", Fariba Sadri and Antonis Kakas (eds.), Springer, forthcoming 2001.

[10] José Júlio Alferes, Luís Moniz Pereira and Teodor C. Przymusinski, "'Classical' Negation in Nonmonotonic Reasoning and Logic Programming", Journal of Automated Reasoning, 20:107-142, 1998.

[11] José Júlio Alferes, Luís Moniz Pereira and Terrance L. Swift, "Well-founded Abduction via Tabled Dual Programs". In D. De Schreye (ed.), Procs. 16th Int. Conf. on Logic Programming, Las Cruces, New Mexico, pages 426-440, MIT Press, 1999.

[12] John Barth, "On with the story", page 55, Back Bay Books, Little Brown and Company, 1996

[13] Carlos V. Damásio, Wolfgang Nejdl, Luís Moniz Pereira and Michael Schroeder, "Model-based Diagnosis Preferences and Strategies Representation with Logic Meta-Programming". In K. Apt and F. Turini

(eds.), "Meta-Logics and Logic Programming", pp. 267-308, MIT Press, 1995.

[14] Carlos V. Damásio and Luís Moniz Pereira, "A Survey on Paraconsistent Semantics for Extended Logic Programas". In D.M. Gabbay and Ph. Smets (eds.), "Handbook of Defeasible Reasoning and Uncertainty Management Systems", vol. 2, pp 241-320, Kluwer Academic Publishers, 1998.

[15] Carlos V. Damásio and Luís Moniz Pereira, "Hybrid probabilistic Logic Programs as Residuated Logic Programs". In M. O. Aciego, I. P. de Guzmán, G. Brewka and L. M. Pereira (eds.), Logics in AI, Procs. JELIA'00, Málaga, LNAI 1919, pages 57-72, Springer 2000.

[16] Carlos V. Damásio and Luís Moniz Pereira, "Monotonic and Residuated Logic Programs". In Procs. 6th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01), S. Benferhat and Ph. Besnard, Springer, LNAI 2001.

[17] Carlos V. Damásio and Luís Moniz Pereira, "Antitonic Logic Programs". In Procs. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'01),T. Eiter and M. Truszczynski (eds), Springer LNAI 2001.

[18] Carlos V. Damásio, Luís Moniz Pereira and Terrance L. Swift, "Coherent Well-founded Annotated Logic Programs". In M. Gelfond, N. Leone and G. Pfeifer (eds.), Procs. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), El Paso, Texas USA, pages 262-276, LNAI-1730, Springer, 1999.

[19] Terrence Deacon, "The Symbolic Species", W.W. Norton, 1997.

[20] Pierangelo Dell'Acqua, Luís Moniz Pereira, "Updating Agents". In S. Rochefort, F. Sadri and F. Toni (eds.), Procs. ICLP'99 Workshop on Multi-Agent Systems in Logic (MASL'99), Las Cruces, New Mexico, 1999.

[21] Pierangelo Dell'Acqua, Luís Moniz Pereira, "Preferring and Updating in Abductive Multi-Agent Systems". In Procs. Workshop on "Engineering Societies in the Agent's World 2001" (ESAW'01), Andrea Omicini et al. (eds), Prague.

[22] Merlin Donald, "Origins of the Human Mind", Harvard University Press, 1991.

[23] Peter Froehlich, Carlos V. Damásio, Wolfgang Nejdl, Luís Moniz Pereira, Michael Schroeder, "Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks". In Procs. of The 12th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems IEA/AIE-99. Cairo, Egypt. LNAI, Springer 1999.

[24] Joseph Gartner, Terrance L. Swift, Allen Tien, Carlos V. Damásio, Luís Moniz Pereira, "Psychiatric Diagnosis from the Viewpoint of Computational Logic". In J. Lloyd et al. (eds.), Procs. of First Int. Conf. on Computational Logic (CL 2000), London, UK, pages 1362-1376, LNAI 1861, Springer, July 2000.

[25] Paul R. Gross and Norman Levitt, "Higher Superstition", The Johns Hopkins University Press, 1994.

[26] John Holland, "Emergence", Addison-Wesley, 1998.

[27] Anthony C. Kakas, Robert A. Kowalski and Francesca Toni, "Abductive Logic Programming", Journal of Logic and Computation, 2:719-770, 1993.

[28] Robert A. Kowalski, "Logic for Problem Solving", North-Holland, 1979.

[29] Imre Lakatos, "Proofs and Refutations: the Logic of Mathematical Discovery", Worrall, J. and Zahar, E.G. (eds), Cambridge University Press, 1976.

[30] E. Lamma, F. Riguzzi, L. M. Pereira, "Belief Revision by Lamarckian Evolution". In Procs. 1st European Workshop on Evolutionary Learning (EvoLEARN2001), E. J. W. Boers et al (Eds.) "Applications of Evolutionary Computing", Springer-Verlag, LNCS 2037, pp. 404-416, 2001.

[31] Evelina Lamma, Luís Moniz Pereira and Fabrizio Riguzzi, "Logic Aided Lamarckian Evolution". In P. Brazdil and R. Michalski (eds.), Procs. Multi-Strategy Learning Workshop (MSL'00), Guimarães, Portugal, published by LIAAC - Porto University, June 2000.

[32] Evelina Lamma, Fabrizio Riguzzi and Luís Moniz Pereira, "Strategies in Combined Learning via Logic Programs". Machine Learning 38(1/2): 63-87, January 2000.

[33] João A. Leite, José Júlio Alferes, Luís Moniz Pereira, "Minerva - A Dynamic Logic Programming Agent Architecture". In Proc. Agent Theories, Architectures, and Languages (ATAL'01), John-Jules Meyer (ed.), Springer LNAI, 2001.

[34] João A. Leite and Luís Moniz Pereira, "Generalizing updates: from models to programs". In ILPS'97 Workshop on Logic Programming and Knowledge Representation, LPKR'97, Port Jefferson 1997.

[35] João A. Leite, José Júlio Alferes and Luís Moniz Pereira, "Multi-dimensional Dynamic Logic Programming". In F. Sadri and K. Satoh (eds.), Procs. CL-2000 Workshop on Computational Logic in Multi-Agent Systems (CLIMA'00), London, England, July 2000. An extended updated version appears as "Multi-dimensional Dynamic Knowledge Representation", Procs. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'01), T. Eiter and M. Truszczynski (eds), Springer LNAI 2001.

[36] João A. Leite, Francisco C. Pereira, Amílcar Cardoso and Luís Moniz Pereira, "Metaphorical Mapping consistency via Dynamic Logic Programming". In G. Wiggins (ed.), Procs. AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science, pages 41-50, AISB, Birmingham, England, April 2000.

[37] Luís Monteiro and Luís Moniz Pereira, "Aspectos Cibernéticos da Epistemologia". In "Novas Perspectivas das Ciênciasdo Homem", pp. 13-27, Biblioteca de Ciências Humanas 4, Editorial Presença, Lisboa, 1970.

[38] Thomas Nagel, "The Last Word", Oxford University Press, 1997.

[39] Isaaco Newtono, "Philosophiæ Naturalis Principia Mathematica". Editio tertia & aucta emendata. Apud Guil & Joh. Innys, Regiæ Societatis typographos, 1726. Original quotation: *"Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus illud a viribus impressis cogitur statum suum mutare."*

[40] Luís Moniz Pereira, "Inteligência Artificial: mito e ciência". In Revista Colóquio-Ciências", 3:1-13, October 1988, Fundação Calouste Gulbenkian, Lisboa.

[41] Luís Moniz Pereira, "The Logical Impingement of Artificial Intelligence", Grazer Philosophische Studien (Internationale Zeitschrift Für

Analytische Philosophie), Amsterdam/Atlanta, vol. 56, pp. 183-204, 1999.

[42] Steven Pinker, "How the mind works", W.W.Norton, 1997.

[43] Paulo Quaresma and Luís Moniz Pereira, "Modelling Agent Interaction in Logic Programming", In O. Barenstein (ed.), Procs. of the 11th Int. Conf. on Applications of Prolog (INAP'98), Tokyo, Japan, September 1998.

[44] Paulo Quaresma and Irene P. Rodrigues, "A collaborative legal information retrieval system using dynamic Logic Programming". In Procs. 7th Int. Conf.on AI and Law (ICAIL-99), pages 190-191. ACM SIGART, ACM Press 1999.