Artificial Intelligence 175 (2011) 416–427

From systems to logic in the early development of nonmonotonic reasoning

Erik Sandewall

Department of Computer and Information Science, Linköping University, Linköping, Sweden

ARTICLE INFO

Article history:
Available online 8 April 2010

Keywords:
Nonmonotonic reasoning
Frame problem
Truth maintenance
Defeasible inheritance history of AI

ABSTRACT

This note describes how the notion of nonmonotonic reasoning emerged in Artificial Intelligence from the mid-1960's to 1980. It gives particular attention to the interplay between three kinds of activities: design of high-level programming systems for AI, design of truth-maintenance systems, and the development of nonmonotonic logics. This was not merely a development from logic to implementation; in several cases there was a development from a system design to a corresponding logic. The article concludes with some reflections on the roles and relationships between logicist theory and system design in AI, and in particular in Knowledge Representation.

1. John McCarthy and nonmonotonic reasoning

Nonmonotonic reasoning has emerged as one of the most central concepts in artificial intelligence. It is nowadays an accepted notion in formal logic, and several other disciplines address knowledge representation problems where nonmonotonic reasoning would seem to have a natural role. John McCarthy is of course the founding father that both initiated this development and gave it a large part of its structure and direction.

The purpose of the present article is to describe the beginnings of nonmonotonic reasoning (nonmon) in Artificial Intelligence, that is, the initial development that preceded and led up to the special issue of the Artificial Intelligence journal in 1980 whereby nonmonotonic reasoning was established as a topic of its own, and where circumscription (which is arguably the major approach to nonmonotonic reasoning today) was first presented in a journal article. I participated in the development of the field during those early years, and the present article is based partly on my recollections of my own activities as well as those of fellow researchers, and partly on communication with these during the preparation of this article.

Defeasible reasoning has also been studied in philosophical logic during about the same time as in artificial intelligence. Although there has been some interaction between the fields more recently in this respect, that was not the case during the early years. The present article will therefore be strictly restricted to the early development within A.I. research.

Some of the early publications on nonmon are quite well-known, in particular the article “Some philosophical problems from the standpoint of Artificial Intelligence” by John McCarthy and Patrick Hayes in the 1968 Machine Intelligence workshop [23]. Other work during those formative years included the introduction of the *thnot* operator in Carl Hewitt's `Planner` system [13,14], and the first proposal for a default rule for the frame problem, in my article at the 1971 Machine Intelligence workshop [35].

These early articles are manifestations of the search process where several researchers tried different mechanisms for obtaining nonmonotonic reasoning and different ways of understanding the knowledge representation problems where it seems to be required. One important aspect of that search process is that logic-based approaches and computationally based approaches were tried concurrently and with frequent interactions, and *truth maintenance* became an important issue

E-mail address: erisa@ida.liu.se.

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.04.013

from the systems-oriented point of view. This early connection between logic and systems is less visible today, and in fact the logic-based work dominates among contemporary research contributions. It may however be worthwhile not to let the systems aspect fall into oblivion since truth maintenance may well reemerge as an important issue, in particular in cognitive robotic systems.

In spite of the diversity of approaches that were tried, there is actually a single starting-point for this early work, namely, John McCarthy's paper on *Programs with Common Sense*. This short article was presented at the Teddington Conference on the Mechanization of Thought Processes in December 1958 and it was circulated and reprinted repeatedly, in particular as part of a book [22]. It is important as a starting-point for two reasons. This paper is recognized as the initial proposal for the use of formal logic as the formal and theoretical framework for artificial intelligence systems. However, it also makes the proposal for a *deliberating* software system, that is, a system that works continuously at interpreting its inputs and analyzing its possible futures and its future actions. McCarthy's proposal was to organize such a system in such a way that it continuously draws conclusions from carefully selected subsets of its available knowledge, including conclusions that specify actions to be performed by the system. In this way a deliberating system differs from a *reactive* system that merely responds to queries or requests, and which idles when there is no such input. McCarthy used the term *the advice taker* for the proposed first version of such a software system, with restricted capabilities.

The logicist aspect of the Advice-taker paper has had a strong influence on what we know today as research in knowledge representation. The deliberating-system aspect is also important since continuous "forward" reasoning from known or believed facts leads to the need for truth maintenance in any system where default conclusions are permitted. It is therefore fundamental for cognitive robotics, and more generally for every cogitating system that is set to operate autonomously in some environment.

2. Ambiguity logic

The Advice-taker proposal, as well as the ongoing discussion at the Stanford AI Lab in the mid-to-late 1960's, inspired my own work on a representational language called "ambiguity logic," and on its implementation as a programming system called Lisp A. This language belonged to the same category of so-called "programming languages for AI research" [1] as `Planner` and `QA4`. It was more or less concurrent with `Planner` but independent of it, and it preceded `QA3` and `QA4` in time. Several aspects of nonmonotonicity started in the context of `Lisp A`, so therefore it will be described in some detail here.

The 'logic' part of this work started from a suggestion in another one of McCarthy's early articles [18], to the effect that it would be useful for representation purposes to have "ambiguous" functions, that is, functions with more than one value. An obvious example is the function "the wife of" in a multicultural context. In ambiguity logic, one would represent such a function in more conventional terms, as follows. Consider an application domain containing some kind of identifiable objects where it is natural to have functions from object to object, but for some arguments the function has no value, and for some other arguments it may have several values. Terms in ambiguity logic are interpreted as sets of such objects. There are no expressions for individual objects, only for sets of them, but instead there is a special predicate `*` of one argument that is true iff the argument is a singleton set, having exactly one member. The subset and subset-equal relations are used as usual.

Furthermore, for each ambiguously valued "function" f on the object level that the application offers, one introduces a corresponding function `F` on the set level, where `(F A)` is the set of all objects that are a value of the function f applied to some member of `A`. Set-level functions of more than one argument are defined similarly. Obviously every such function `F` is monotonic with respect to the subset relation in each of its arguments. The basic concept in ambiguity logic was therefore the subset relation on sets of objects, rather than the use of predicates as they occur in predicate logic. There is an evident similarity with modern description-language representations.

Predicates on the object level are considered as functions whose value is one of the two objects `T` or `F`; consequently their counterparts on the set level have four possible values.

In addition to the monotonic functions, including predicates, that are constructed from object-level counterparts, it is convenient to also have some special functions that are not monotonic. The subset relation itself should only be considered to have the value `{T}` or `{F}` and is not monotonic, and the same holds for the `*` predicate in its single argument. Quantification can be expressed using the subset relation: an object-level predicate p is true for all members of a set `A` iff `(P A)` $\subseteq$ `{T}`, and it is true for some member of the set iff `{T}` $\subseteq$ `(P A)`. The generalization to predicates of several arguments is trivial. Implication can be handled in a similar fashion.

Expressions of the form $\{f(x) \mid x \in A\}$ had a counterpart in ambiguity logic where one could write a function as

```
(rho (x)(F x))
```

If `R` is this function, then `(R A)` denotes the union of all `(F` A_i`)` for all singleton subsets A_i of `A`. The reason for expressing this with a lambda-expression-like construct was that one can then write recursive definitions very conveniently.

One use of the `rho` operator was for writing quantified expressions. For example, if `P` was a predicate and the statements

```
((rho (x)(P x)) A) ⊆ {T}
```

```
C ⊆ A
(* C)
```

had been asserted, so that in particular `C` was known to be a singleton subset of `A`, then the instantiation of the "quantified" statement was obtained as follows and using the monotonicity of the rho-expression

```
(P C) = ((rho (x)(P x)) C) ⊆ ((rho (x)(P x)) A) ⊆ {T}
```

showing that `(P C)` is either the empty set or `{T}`, so that it is not false.

This notation was called ambiguity logic, although in retrospect it would have been more appropriate to call it a calculus. Anyway, it was a way of characterizing sets of objects that have certain relationships, so it was primarily for knowledge representation and not for programming.

3. Nonmonotonicity in ambiguity logic

Nonmonotonic inference was introduced in this design in order to deal with another issue which was described as follows in the ambiguity logic report [31]:

"*… incertitude … may arise in several ways*:
(A) *We know that the full and precise statement should be*

$$\textit{let t be a u in } p(t) \wedge q(t) \wedge \neg v(t) \supset r(t)$$

where $v(t)$ *is a relation which indicates some exceptional circumstance* (*like* "*the ceiling is falling down*") *that we can usually ignore.*
(B) *There are not one but many exceptional circumstances, and they cannot all be enumerated. —*
We introduce a Boolean function unless, *which is to be used instead of* not *in cases where the argument is usually false. Thus the statement shall go*

$$\textit{let t bea u in } p(t) \wedge q(t) \wedge \textit{unless}\big(v(t)\big) \supset r(t)\text{"}$$

In the (B) case the predicate v was used to represent the disjunction of all possible exceptional conditions, and each exception was represented by a separate axiom where v was used in the consequent.

Notice that the emphasis was not on the restricted frame problem of assuring persistence of fluents that are not affected by an action; it was on the harder qualification problem. This was the issue that McCarthy emphasized in his early work on this topic.

The *let* expression that was used in this quotation is a modification of Landin's *let* operator for use with rho-expressions. An expression `((rho (x)(P x)) A)` could be written more legibly as `let x be a A in (P x)`.

4. Lisp A and truth maintenance

Ambiguity logic was implemented in `Lisp` as a program called `Lisp A` that maintained the subset relationships between terms in this calculus and answered queries concerning those relationships. The basic approach in Lisp A was inspired by another early paper in the McCarthy tradition. His early proposal for a deliberating system, as described in the Advice-taker memo, had been carried forward through the proposal by Lombardi and Raphael for an *incremental computing system* [17]. They defined three requirements for an "incremental computer", the first one being that

The extent to which an expression is evaluated is controlled by the currently-available information context. The result of the evaluation is a new expression, open to accommodate new increments of pertinent information by simply evaluating again with a new information.

This can be taken as a recipe for partial evaluation, but in `Lisp A` it was realized in a way that was closer to the vision of the advice taker. The input to the Lisp A system was a stream of assertions and queries. The basic cycle of the system was to accumulate assertions to its knowledge base in the form of terms that were related by the ⊆ relation, to administrate the transitivity of ⊆, and to simplify `rho` expressions when appropriate. Each assertion of a subsumption relation could trigger new forward inferences using a first-come, first-served strategy with several "lanes" with different priority. There were handles for procedural attachment, so that the assertion of `(P C)` in the above example could produce computational side-effects. The `rho` operator therefore provided a means of assertion-driven invocation of new assertions and queries, and of invoking attached procedures written directly in `Lisp`.

The use of the `unless` operator in this approach created the need for truth-maintenance, although that particular term had not been introduced yet. There was in fact also another reason for needing it, namely, the possibility that expressions in the input stream could require previously stated relationships to be retracted. The implementation therefore kept track of the justifications of each proposition that had been asserted in it, so that a previously asserted relationship could be retracted by a request from the user, or because all its justifications had been retracted.

The implementation of the `unless` operator in `Lisp A` was described as follows in the quotation above. (The function `ambeval` was the main evaluator in `Lisp A`.)

> *Ambeval needs the following operators to handle the function* unless:
> 1. *An operator which checks* `t` *in each expression* `unless(t)` *and, if* `t` *does not have any* `BECAUSE` *property, gives* `unless(t)` *the property* `NIL` *under the attribute* `BECAUSE` (*which means that* `unless(t)` *is considered true with no reason*); —

The `BECAUSE` property contained the justifications for a proposition as a list of other propositions whose conjunction implied the proposition at hand. The property-value `NIL` was distinct from an absence of value, and represented that the proposition had been accepted by default. (A more complete justification structure on disjunctive normal form was provided using another property.)

In summary, `Lisp A` was an early example of a truth maintenance system, where nonmonotonic behavior was introduced, using an operator called *unless*, in order to handle what later came to be called the qualification problem. The articles about this work do not at any point mention the "frame problem" for reasoning about actions, and the representation is motivated with an instance of the qualification problem.

Ambiguity logic and its `Lisp A` implementation were developed during the first eight months of 1967. Detailed reports can be found in a departmental report from the University of Uppsala, Sweden [31] and in an article about `Lisp A` at the 1968 Spring Joint Computer Conference [33]. This approach was used in a proposal for representing vector diagrams in the modelling of bubble-chamber pictures [34]. The conciseness of the formalism was illustrated by showing hos GPS-style search could be implemented in a few lines of `Lisp A` definitions [32].

5. The `Planner` system

Carl Hewitt's `Planner` system was a concurrent approach to "programming languages for AI" which also developed nonmonotonic concepts at an early stage. The direct sources for that work are in Hewitt's papers at IJCAI 1971 [13] and a sequence of successive revisions of the `Planner` system memo during the preceding years, converging in Hewitt's Ph.D. thesis at MIT in 1972 [14]. (The earlier article on Planner at IJCAI 1969 [12] describes Planner as a high-level programming language based on a pattern matcher, and does not address the topic of the present article.) The following is a description of the `Planner` approach from the abstract of those memos and the thesis:

> `Planner` *is a formalism for proving theorems and manipulating models in a robot. The language is built out of a number of problem-solving primitives together with a hierarchical multiprocess backtrack control structure. Statements can be asserted and perhaps later withdrawn as the state of the world changes. — The deductive system of* `Planner` *is subordinate to the hierarchical control structure in order to make the language efficient. —*

Therefore, `Planner` also had an assertion/retraction capability supporting forward and backward inference rules. It differed from `Lisp A` since its backtracking mechanism was integrated with the programming language and since it provided depth-first search through strands of reasoning, whereas `Lisp A` had a first-come, first-serve queueing system for tasks that was "above" the level of the language implementing it, and justification tags on propositions to support retraction. Also, `Planner` used a representation along the lines of first-order predicate calculus, to mention only the most important differences.

Procedures were invoked implicitly in `Planner`, using patterns that specified the 'goals', that is, specifying what the procedures were supposed to accomplish.

The *thnot* operator is not mentioned in the 1968 version of the `Planner` memo, but appears in the 1970 revision as follows:

> `(THNOT x)` *is an abbreviation for* `(thcond (x (fail))(t t))`. *Thus* `(thnot ())` *is* `t`, `(thnot t)` *is* `()`, *and* `(thnot (fail))` *is* `t`. *The function* `thnot` *is due to T. Winograd.*

It is not mentioned in the IJCAI 1971 article [13] about `Planner`, and in fact Hewitt was always reserved about the nonmonotonic logic project. His view was at that time[1]:

> *We knew about the frame problem from contemporary ongoing work at SRI on Strips with its ADD and DELETE lists. But we thought that Planner was superior because of the ability to write pattern-directed* (*conditional recursive*) *plans for goals and assertions that could more flexibly do additions and deletions to the global data base to try to keep things consistent.*

[1] Carl Hewitt, e-mail communication, 2008.

I thought that THCOND was more fundamental than THNOT. THNOT was obvious in the context of the other Planner "TH" variants of Lisp primitives and could be trivially defined in terms of THCOND. On the other hand, THCOND did not fit into the clause-based logic that was in favor at the time. Thus Prolog duplicated THNOT but ignored THCOND.

Hewitt later revised his design in favor of his Scientific Community Metaphor that avoided the use of deletion and of backtracking.

However, in spite of Hewitt's lack of interest in the *thnot* operator, a number of other authors were inspired by it and noticed its relevance for nonmonotonic reasoning. Drew McDermott and Jon Doyle write, in [25]:

In PLANNER ..., *a programming language based on a negationless calculus, the* `thnot` *primitive formed the basis of non-monotonic reasoning.* `thnot`, *as a goal, succeeded only if its argument failed, and failed otherwise. Thus if the argument to* `thnot` *was a formula to be proved, the* `thnot` *would succeed only if the attempt to prove the embedded formula failed. In addition to the nonmonotonic primitive* `thnot`, PLANNER *employed antecedent and erasing procedures to update the data base of statements of beliefs when new deductions were made or actions were taken. Unfortunately, it was up to the user of these procedures to make sure that there were no circular dependencies or mutual proofs between beliefs. ...*

In summary, although there were a number of important differences between `Lisp A` and `Planner`, they also had important notions in common, including:

- Invocation of rules according to assertions and requests.
- Assertions, retraction of assertions and their consequences.
- Negation by failure.

6. The STRIPS system

The `STRIPS` system [10] was developed by Nils Nilsson and Richard Fikes at SRI around the same time as `Planner`. It was a system for the planning of action sequences in robots which was combined with other subsystems, in particular for the execution of the constructed plans, in the construction of the `Shakey` autonomous robot system.

`STRIPS` was based on rules specifying the additions and deletions in the model of the robot's environment that are associated with each of a number of robot actions, so the nonmonotonicity in its behavior was not formulated in terms of logic. In spite of this it has played an important role in the development of nonmonotonic logics for the purpose of contrast: the use of logic for reasoning about actions requires a way of doing within logic what STRIPS does outside of formal logic, so STRIPS has been repeatedly used as a challenge.

In this context it is also interesting to note that Fikes implemented a truth-maintenance-like feature that kept track of the support for derived elements of the state model after each state change [9].

7. The procedural-declarative controversy, nonmonotonicity, and the frame problem

During these early years, the need for nonmonotonic reasoning was mostly mentioned in the context of the qualification problem, the frame problem, and for reasoning about knowledge and belief. The first published article where this was laid out was the classical "Some philosophical problems from the standpoint of Artificial Intelligence" by McCarthy and Hayes [23], which was published in 1969, but most of the ideas in this paper had been circulated by its authors during the preceding years.

The period around 1970–1975 was the time of the big "procedural-declarative controversy" in AI, where the proceduralists, such as Marvin Minsky and Carl Hewitt, argued against the use of logic which had been proposed by John McCarthy in his original "advice taker" paper as well as in the McCarthy–Hayes article. The objections were of several kinds, including insufficient expressiveness, inefficient implementation, and lack of agreement with how intelligence works in humans. With respect to expressiveness, one major point was that standard logic is monotonic, in the sense that the set of theorems from a set of axioms increases monotonically as the axiom set is extended. Common-sense reasoning and other common types of human thinking are not like that, it was argued, in particular because of how we can make inference on the basis of defaults.

The McCarthy–Hayes paper does not give any direct answers to those objections, but it does introduce some novel ideas that extend the traditional notions of logic. With respect to nonmonotonicity, there is a proposal for "formal literatures" where lines in a proof are allowed to refer in various ways to the entire sequence of earlier lines, and not only use them as the basis for applying inference rules. There is a new quasi-predicate, *consistent* p, which specifies that a particular formula p, is consistent with the previous lines of the proof, and another one, *normally* p, for specifying what is normally the case. Finally there is a rule that allows one to draw a tentative conclusion, written *probably* p, if both the first two predicates apply to p.

The paper discusses an example where this mechanism is applied to representing what is now called qualification. The example concerns what is required in order to place a telephone call in the normal ways, and what are some exceptional circumstances where placing the call may fail. The paper also describes the limited frame problem (persistence of fluents that are not implicated in an action) and suggests in one sentence that the "formal literature" technique might be useful for solving this problem as well, but without going into any details:

Many of the problems that give rise to the introduction of frames might be handled in a similar way.

8. The frame problem paper at Machine Intelligence 7

The McCarthy–Hayes article had put the frame problem on the agenda, and it was also addressed in an article by Bertram Raphael that was presented at a workshop in 1970 [27]. From a logicist point of view, the natural next step was to find a representation for this nonmonotonic reasoning problem that was more in line with standard logic than either the procedural approach of `Planner`, the functional approach of `Lisp A`, or the formal-literature proposal of McCarthy and Hayes.

The *unless* operator from `Lisp A` could easily be adapted to fill this need, and at the Machine Intelligence 7 workshop in 1971 I proposed to extend first-order logic with the use of rules of the form

```
A,  Unless B   =>   C
```

meaning that if `A` had been proved and `B` *could not* be proved (from the set of axioms and rules at hand), then `C` followed. The article [35] is noncommittal as to whether an expression `Unless B` should be considered as a well-formed formula that can be embedded inside larger formulas, or whether a rule such as the one above ought to be considered as a special-purpose inference rule. The pros and cons of those alternatives was considered as a topic of further investigation.

There are two major points in the MI7 article, besides the proposal for rules with an `Unless` operator in itself. First, the article observes that the use of rules of this kind results in the possibility of multiple extensions, as exemplified by the three axioms or rules

```
A,  Unless B  =>  C
A,  Unless C  =>  B
A
```

This is of course a major issue for nonmonotonic logic. The paper includes a brief discussion of how to handle this problem, and in particular a suggestion to handle it by imposing a priority order on the default rules.

The other major point in this article was a discussion of the frame problem and a proposal for how to express it using a nonmonotonic 'frame rule'. The proposal was as follows. Given that one wishes to use a predicate `IS` where `IS(o,p,s)` means "the object `o` has the property `p` in situation `s`", introduce a second predicate `ENDS` and a 'frame rule' of the form

```
 IS(o,p,s), Unless ENDS(o,p,Succ(s,a))  =>  IS(o,p,Succ(s,a))
```

where `Succ(s,a)` is the successor situation that results from the situation `s` by performing the action `a`. When the article proposes using this rule for the frame problem, that term is taken to include more than plain persistence of fluents. It points out the usefulness of rules of the form

```
ENDS(o,p,s) -> ENDS(o',p',s)
```

in order to represent what is today known as ramification. The article says:

This approach to the frame problem gains its strength (*as compared to, for example, STRIPS*) *from the fact that one can make deductions to any depth using the predicate ENDS.*

Examples included "If *x* ends being alive, then *x* ends being a friend of *y*"; "If *x* supports *y* and *x* moves to *l*, then *y* moves to *l*", and "If *x* moves, then *x* ends being where it was".

It is regrettable that this article did not contain any discussion of how the proposed approach relates to the proposal by McCarthy and Hayes involving the three operators *consistent*, *normally*, and *probably*, nor even a reference to their MI4 paper. The proposal in the article represents a simplification that reduces the expressivity of the formalism. In particular, the *normally* operator allows one to distinguish between hard conclusions and conclusions that have been obtained by default. The new proposal using `unless` removed that possibility.

An interesting indication of the utility of the original operators proposed by McCarthy and Hayes occurred in the late 1980's in the context of the Prometheus project.[2] The road-traffic experts that participated in this project reported that many traffic accidents arise in situations where a driver "guesses" that the current situation shall be interpreted in a par-

[2] Prometheus was a European cooperation project on information technology in automobiles.

ticular way, and the guess is plausible but happens to be wrong. This is a situation that could maybe be characterized as $normally(\phi) \wedge \neg\phi$ in McCarthy–Hayes terms. It would be useful to be able to reason about such accident-prone situations. The NML3 logic defined by Doherty and Łukaszewicz [4] was a proposal for how to fill that need.

9. Prolog and negation by failure

The Prolog programming language was developed by Alain Colmerauer of the University of Aix-Marseille in cooperation with Robert Kowalski of the Imperial College in London. Colmerauer's immediate interest was in developing Prolog as a software tool for honoring database queries expressed in natural language.[3] Kowalski was interested in the same topic for two reasons: in order to show that Planner's procedural representation of knowledge could be obtained by means of resolution logic, and to demonstrate that logic is useful as a high-level programming language. Much of the basic language design was made during Kowalski's visit to Marseille in the summer of 1971. However, the introduction of negation by failure was made by Colmerauer during Kowalski's visit the following summer, as a way of handling problems that came up in the course of developing the natural-language system.

The initial development of nonmonotonicity in Prolog differed in an important respect from the approaches that have been described above: it was only seen as a way of cutting off a branch of search (hence its alternative name, the 'cut'). The initially intended applications had a reactive character and did not suggest the use of a deliberating system.

The following years saw the successful use of Prolog for a number of applications, such as Warren's WARPLAN system for planning [40] and Kanoui's system for symbolic integration [15], besides the natural-language system of Colmerauer and his colleagues.

An important step forward was taken when Keith Clark presented a declarative semantics for negation as failure [2]. This provided Prolog with a formal strength that complemented its proven practical usability and contributed to the rapid growth of its enthusiastic user community and the resulting literature.

10. The debate about nonmonotonicity

The idea of nonmonotonic logic and nonmonotonic reasoning was put forward by my MI7 paper and by the widespread interest in Planner's *thnot* operator, and it led to a considerable discussion during the following years, sometimes with fairly heated arguments. The 1980 special issue of *Artificial Intelligence* contains some of the discussion and references to earlier discussion.

At the 1975 IJCAI (held in Tbilisi, Georgia, which at that time was a republic within the USSR), there was an article by Ivan Kramosil [16] who argued that it was impossible to assign a meaningful semantics to nonmonotonic formalisms, effectively because they lack the extension property. Later developments have of course shown that he was wrong.

11. The development of truth maintenance systems

Hewitt's Planner system was an attempt to address the problem of *control of reasoning* in a profound way by tightly integrating the reasoning machinery (propositions, methods for drawing and retracting conclusions, and methods for controlling those methods) with a programming language. This project was in line with Marvin Minsky's critique of the use of logic in AI, and in particular his remark that many of the valued properties of logic (such as its monotonicity) makes it unsuitable for controlling reasoning. The question of how to control reasoning processes, even in deliberating systems, was therefore an important topic of interest in the MIT AI Laboratory during the post-Planner period in the 1970's.

In hindsight it is easy to see how this would lead to the work on truth maintenance systems, and from there to proposals for a nonmonotonic logic. However, some preceding work on constraint propagation in other contexts is also part of the history.

Constraint propagation at MIT had started with David Waltz's work on interpreting visual scenes, and the same techniques had been used soon thereafter by Gerry Sussman and Richard Stallman in the EL electronic circuit analysis system [39]. Sussman, Terry Winograd, and Eugene Charniak had previously implemented a subset of Planner as a system called Micro-Planner [38] which had been used by Winograd as a software tool for the SHRDLU system [41]. One of the goals of EL was that it should provide explanations of its results, and it can therefore be seen as one of the earliest expert systems. In order to be able to provide those explanations it employed a scheme for making assumptions and for changing them, although each change of assumptions required an exhaustive check of the validity of conclusions throughout the system.

Sussman and Stallman then generalized their approach and developed the Antecedent Reasoning System, ARS, a system for forward propagation of constraints [37]. One of the new features in ARS was *dependency-directed backtracking*, whereby the system could undo the effects in a particular thread of reasoning when that thread had led to an inconsistency. Jon Doyle, then a graduate student at MIT, joined the ARS project and contributed to the design of the backtracking system.

[3] http://alain.colmerauer.free.fr/curriculum.html.

Similar problems also arose in other projects at MIT at that time. Drew McDermott had developed a planning system, called NASL [24] which is believed to be the first planning system that controlled itself by declarative means. Both the individual actions in the plans and the supporting declarative information were expressed in first-order logic, with minor extensions. Default reasoning was used in an on-demand basis: when the system failed to obtain an answer for a query, it invoked rules containing an “unless” condition in order to obtain additional assumptions whereby an answer to the query could be found. NASL was used as the basis for McDermott’s electronic circuit design system.

The work on declarative control of reasoning continued in a joint project with Gerry Sussman, Drew McDermott, Jon Doyle, Johan de Kleer and Guy Steele which resulted in a system called AMORD, for “A Miracle of Rare Device” [3], a phrase of literary origin.

As one of the project members, Jon Doyle decided to work on combining these two lines of work, EL/ARS and NASL/AMORD, which led to his development of the Truth Maintenance System, a term which was at first the name of his particular system, but which has since then become generic. This work started in 1976 and was reported in [7,8]. The main problem was, of course, how to organize the justification information for formulas in the database. The solution was to express the justification as an expression using the AND and OR operators to combine elements of the form (IN N_i) or (OUT N_i), intended to mean that N_i is “in” the database in the sense of the presently believed and supported propositions, or “outside” the database in the opposite case.

In addition, each proposition in the store was labelled with its IN or OUT status. Doyle identified the requirements that one must impose on this structure, namely, the IN or OUT labelling of each proposition must be consistent with its justification expression, and there must not be any circular justifications. The update algorithm that maintains these requirements was an important part of the work, and there was work on making them as efficient as possible.

One of the consequences of this architecture is that there may be several correct assignments of IN/OUT status and justifications for a given initial set of propositions, which is analogous to the existence of multiple extensions in nonmonotonic logics. This was observed and taken both as a natural consequence of the design, and as a reasonable state of affairs in an AI system. The identification of the grounded stability condition is an important result of Doyle’s work.

The overriding problem was however how to control search and reasoning in a systematic way; Doyle showed how to do this using nonmonotonic justifications.

12. From truth maintenance systems to nonmonotonic logic

The stage had now been set for proceeding from truth maintenance systems to the formulation of a corresponding logic, and McDermott and Doyle addressed this problem jointly. This resulted in the definition of Nonmonotonic Logic I [25], which is related to Doyle’s previous work on TMS in the sense that extensions were defined in a way that makes them equivalent to fixedpoints of the assumption closure operator in the TMS. The article about Nonmonotonic Logic I is actually the only one in the 1980 special issue of the Artificial Intelligence Journal that discusses the relationship of the logic in question to truth-maintenance systems (or to any kind of software system, in fact) to any depth.

Both McDermott and Doyle continued to work on this topic during the following years, for example in the direction of supporting the inference of new default rules.

13. Default logic

The development of default logic by Ray Reiter can be traced back to his 1978 article on this topic [28], where he proposes that a number of significant concepts in computer science and in AI can be unified through the concept of a *default*. In particular he mentions the *thnot* operator in Planner, my MI7 paper, the use of negation by failure in Prolog, and the use of defaults in programming languages and software systems. This appears to be the first systematic discussion of the range of potential uses for reasoning by default, i.e., for nonmonotonic reasoning.

An additional important point in this article is that it discusses inheritance of properties in taxonomical structures, where higher-level nodes can specify defaults that can be overridden by assignments to lower-level nodes. This marks the beginning of representing defeasible inheritance in terms of nonmonotonic logic, which is now one of the major applications for the latter.

As a uniform framework for representing these various kinds of defaults, Reiter proposes a logic that represents nonmonotonic rules as specialized inference rules, along the lines of one of the two alternatives for my MI7 paper. However, with respect to the possibility of multiple extensions, his position is clear and negative: “*Default theories exhibiting such behavior are clearly unacceptable*”.

This position has been relaxed in his major article two years later where default logic is introduced in a systematic way [29]. Here Reiter introduces default logic in a more general form, but he also identifies restrictions on the logic whereby the existence and uniqueness of extensions are guaranteed. Furthermore he writes, concerning normal defaults:

In fact I know of no naturally occurring default which cannot be represented in this form.

Reiter's perspective on truth maintenance is similarly restrictive. His original paper on default logic in 1978 devotes one column in the proceedings article to the problem of automatic update, but the article in 1980 contains results that guarantee that 'belief revision' will not be required.

14. Expressing the frame rules in default logic

Reiter's preference for normal defaults may be the reason for a curious development with respect to the formulation of the frame rule in default logic. It was expressed as follows in my MI7 paper:

```
IS(o,p,s), Unless ENDS(o,p,Succ(s,a))  =>  IS(o,p,Succ(s,a))
```

In his 1980 paper Reiter quoted this proposal but rewrote it as, in comparable notation,

```
IS(o,p,s), Unless not IS(o,p,Succ(s,a))  =>  IS(o,p,Succ(s,a))
```

which says that if it is consistent for the object `o` to retain the property `p` after the action `a` has been performed, then it will do so. In this way he obtained a normal default rule, for which he had proved a number of useful properties, and a rule that corresponded directly to STRIPS; he writes:

Intuitively this default schema formalizes the so-called 'STRIPS assumption': Every action (*state change*) *is assumed to leave every assumption unaffected unless it is possible to deduce otherwise.*

This formulation was generally accepted, and it was used e.g. by Hanks and McDermott in their Yale Shooting Problem article at the AAAI 1986 conference [11]. The following year, Paul Morris in [26] proposed an alternative formulation in default logic where the YSP anomaly did not arise. He argued that the use of normal defaults was the reason for the problem, and proposed using essentially the following nonnormal default rule

```
IS(o,p,s), Unless AB(o,p,e,s)  =>  IS(o,p,Succ(s,e))
```

which is equivalent to the original proposal in the MI7 paper. This provides a case for the utility of nonnormal default rules.

15. Circumscription and minimization of models

The standard early reference to the circumscription method is McCarthy's article in the 1980 special issue of the Artificial Intelligence Journal [20], which may lead the casual reader to think that this powerful concept came about in the abstract way that it is often presented. There is however a background story even in this case.

McCarthy liked to use simple examples of commonsense reasoning which he characterized as his 'drosophila': simple examples that are useful to research although they lack practical significance. One of these drosophila was the old problem of three missionaries and three cannibals that are supposed to cross a river using one boat. Already during the 1960's he had made a point of the closed-world assumptions that are needed in order for a puzzle like this one to make sense. One must be able to assume that there is no bridge across the river, there is not a hole in the bottom of the boat, and so on, based only on the fact that the problem statement does not mention any of these.

This example is used in the first published account of circumscription, which appeared at the IJCAI Conference in 1977 [19]. After briefly discussing the closure problem in the example just described, McCarthy writes:

The intuitive idea of circumscription *is as follows*: *We know some objects in a given class and we have some ways of generating more. We jump to the conclusion that this gives all the objects in the class. Thus we* circumscribe *the class to the objects we know how to generate*.

McCarthy then describes how this operation can be expressed in the framework of first-order logic using a predicate of one argument, and how it can be expressed in set theory. He furthermore observes that there is a semantic way of looking at circumscription, simply as a minimization operation on models.

The subsequent journal article [20] describes circumscription in more detail, but the object-oriented way of looking at it is still present. McCarthy writes:

Circumscription is one candidate for accomplishing [*the closure*]. *It will allow us to conjecture that no relevant objects exist in certain categories except those whose existence follows from the statement of the problem and the common sense knowledge*.

Later on in the article McCarthy proposes that the ontology should allow the ramification of all those *things* that can invalidate the proposed solution of the problem, not only the 'bridge' but also the 'leakiness' and even the 'lack of oars'.

There is a significant step from only minimizing a domain or a predicate of one argument, to the minimization of other predicates. This step is taken in the last sections of the same article where circumscription is used for specifying the closure of a transitive relation and, interestingly, for an early variant of the abnormality predicate which was later introduced in [21]. The 1980 variant of abnormality was called *prevents* and was used for the axiom

$$\forall x \forall y \forall s.\big(\forall z.\neg \mathit{prevents}\big(z, \mathit{move}(x,y), s\big) \supset \mathit{on}\big(x, y, \mathit{result}\big(\mathit{move}(x,y), s\big)\big)\big)$$

together with several axioms where *prevents* occurs in the consequent.

Although most of the subsequent work on circumscription has used the syntactical formulation using an axiom schema or a second-order axiom, there has also been some following to the semantic way of viewing it that McCarthy mentioned in his first article. This was natural: defeasible reasoning was often understood in terms of preferences of one kind or another. For example, it served as the basis for Shoham's proposal for a "semantical approach" to nonmonotonic logics [36] where he attempts to formulate a general framework that may subsume the nonmonotonic logics that were being considered at the time. However it may be more accurate to see Shoham's proposal as a generalization of the semantic view of circumscription that also subsumes modal nonmonotonic logics such as the nonmonotonic logic of McDermott and Doyle [25], whereas default logic does not quite fit the mold.

In summary, when reading the accounts of circumscription that were published in 1977 and 1980, one is struck by how well they represent issues and methods that are still valid today.

16. Circumscription and negation-by-failure

Since the 1980 special issue of the Artificial Intelligence Journal is commonly viewed as the main starting-point for nonmonotonic reasoning research in A.I., it is particularly regrettable that the state of the art of Prolog-related research was not represented there. Keith Clark's work on the semantics of negation by failure, which has already been mentioned, had been published two years earlier. This omission was however compensated to some degree in 1982 with the publication of Reiter's article about the relation between circumscription and predicate completion [30].

The immediate impact of the original circumscription articles was moderate, maybe because the circumscription approach appeared to be so different from all the previous ones. It was only with the publication of additional articles, by McCarthy himself and by Vladimir Lifschitz in the mid-1980's that circumscription began to be understood and used. It is also at this time that the emphasis on predicates in general becomes more pronounced and the focus on minimizing domains begins to recede.

17. Winners and non-winners in the development after 1980

Circumscription, default logic and logic programming are the dominating approaches to nonmonotonic reasoning today, although there is also some work that uses e.g. the semantic approach proposed by Shoham. Each of these has its family of applications and of implemented systems. In a contemporary perspective it may be natural see them as examples of a standard pattern that begins with the development of theory, and that continues with applications and with software implementation.

It is striking, however, that the particular systems aspect that was important when nonmonotonic logic emerged has now more or less left the research scene, namely, truth-maintenance systems, and that those proposals for nonmonotonic logic whose development had been intimately connected to truth-maintenance are also not very much present. In hindsight this is actually a bit strange. Two of the winners were represented in the 1980 special issue by a highly theoretical paper (default logic) or by a paper that was based on a few very simple examples of commonsense reasoning (circumscription). The third winner (prolog) was not even represented there, but its strength at the time laid in having a combination of important factors: a clean semantics, efficient implementations, a number of demonstrated applications, and an enthusiastic user community.

The contributions on Nonmonotonic Logic I by McDermott and Doyle would a priori seem to have occupied the centerground: they were founded in a suite of successful systems that were arguably more advanced than the Prolog-based systems at the time in the sense that they addressed the needs of deliberating systems and not merely reactive systems, and the integration between logic and system was more profound. The research community in the MIT AI Laboratory was strong and vibrant, and McDermott having moved to Yale had added one more immediate constituency.

So why did not this approach win widespread acceptance? One reason may be that their logic continued to evolve and to change, and that the McDermott–Doyle cooperation ebbed out. Default logic and prolog remained stable, by comparison, and circumscription evolved with more continuity. This meant that those who were interested in the continued work of McDermott and of Doyle tended to focus on the logic as such, and the lack of visible applications may have led others to lose interest.

Another reason may have been that the combination of nonmonotonic reasoning and truth maintenance resulted in complex and resource-consuming systems that could not run effectively in the computer systems at that time. Logic programming, in particular, had the advantage of efficient implementation.

Yet another reason can maybe be extracted from McDermott's analysis of "the PROLOG phenomenon" after a visit to Europe [25]:

Americans and Europeans have reacted differently to the problems of PLANNER-like languages. Americans tended to see the PLANNER interpreter as a problem solver, with PLANNER programs as data. [*Systems in this tradition*] *have served as vehicles for studying control regimes that are fundamentally different from PLANNER's backtracking.*
The attitude [*of the Europeans*] *towards backtracking has been simply that it is a programmer's duty to remember that his programs will be executed backward as well as forward, that his programs must correct bad guesses as well as exploit good ones.* (...) *The logical next step was to freeze the interpreter design and make it as efficient as possible. The result is a programming language, not a problem solver or a theorem prover*; *it doesn't compete with NOAH, but with LISP.* (...)

The proposed intercontinental divide aside, one may argue that these two perspectives on actual systems can also be applied to the nonmonotonic logics themselves. In a view that corresponds to the "American" perspective according to McDermott, the nonmonotonic logic is supposed to be a correct logic of commonsense reasoning; if provided with commonsense propositions it shall be able to produce commonsense conclusions without further aid. In a programming-language-like view, on the other hand, a nonmonotonic logic is seen as a machinery that has to be "programmed" by designing the set of axioms appropriately, and if a given set of axioms results in unintended conclusions then one should just modify the set of axioms until the intended results are obtained.

The Hanks–McDermott article with the Yale Shooting Problem [11] expresses disillusionment with both perspectives. It observes that unintended conclusion sets are obtained when standard approaches to nonmonotonic logic at the time are combined with a few spontaneously written axioms for a very simple example. At the same time, the authors argue that crafting the premises or the logic itself until it works as intended, is not a realistic enterprise. McDermott's doubts about the viability and relevance of nonmonotonic logic must have contributed to weakening the interest in the approach that he had pioneered.

A related, important question is why truth maintenance is not an active research topic at present, especially since its basic underlying principle is far from dead. Automatic update of derived facts following changes in their premises is in widespread use, in particular in spreadsheet software. It is also clear that deliberating autonomous agents are going to need this kind of update, not merely for elementary data but also as an integral part of their knowledge representation systems.

A possible explanation for this state of affairs may be that the time has not yet come. Cognitive robotics systems that are capable of sophisticated reasoning while operating in the real world are arguably the most difficult task that artificial intelligence addresses, and there are many other problems that must be solved before one can make use of both nonmonotonic reasoning and truth maintenance.

If these are the explanations then one should expect that the logicist aspect and the software-system aspect of nonmonotonic reasoning will be reintegrated, maybe in ways that resemble what was done in the period when nonmonotonic reasoning techniques were first developed.

18. Conclusions

The scenario that I have now described for how nonmonotonic reasoning evolved in artificial intelligence, is in fact an instance of a more general issue concerning the relations between systems and theory in AI. Given that the overriding goal of Artificial Intelligence is to design *intelligent systems* and the goal of Knowledge Representation is to provide the "knowledge" aspect of those systems, is it then best to focus the theory part of KR on those issues that evolve from working with systems, or is it best to go from intuitions and drosophila, to knowledge representations and logic, and from there to actual systems? The early history of nonmonotonic reasoning may provide arguments for both of those positions.

Acknowledgements

In the preparation of the present article I have invited as many as possible of the researchers that participated in the early development of nonmonotonic reasoning to give their comments, as well as several others who could provide missing pieces of information. Their contributions have been very helpful, and I want particularly to thank John McCarthy, Jon Doyle, Cordell Green, Carl Hewitt, Bob Kowalski, Jérôme Lang, Hector Levesque, Vladimir Lifschitz, Drew McDermott, Nils Nilsson, Bert Raphael and Terry Winograd for their help in these respects. A more extensive piece of background was provided by Jon Doyle whose memoir manuscripts [5,6] contain information about the development of truth-maintenance systems and related logics at MIT.

References

[1] Daniel G. Bobrow, Bertram Raphael, New programming languages for artificial intelligence research, ACM Computing Surveys 6 (1974) 153–174.
[2] Keith Clark, Negation as failure, in: Hervé Gallaire, Jack Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 293–322.
[3] Johan de Kleer, Jon Doyle, Guy L. Steele Jr., Gerald Jay Sussman, AMORD, explicit control of reasoning, in: Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages, Association for Computing Machinery, 1977, pp. 116–125.
[4] Patrick Doherty, Witold Łukaszewicz, NML3 – a non-monotonic logic with explicit defaults, Journal of Applied Non-Classical Logics 2 (1–2) (1992) 9–48.
[5] Jon Doyle, The luxury of eccentricity, Unpublished memoir drafted August 3, 2000 and revised in 2002, 2004.
[6] Jon Doyle, Winning by default, Unpublished memoir drafted August 3, 2000 and revised in 2002, 2004.

[7] Jon Doyle, Truth maintenance systems for problem solving, Technical Report TR-417, MIT AI Laboratory, 1978.
[8] Jon Doyle, A truth maintenance system, Artificial Intelligence 12 (2) (1979) 231–272.
[9] Richard E. Fikes, Deductive retrieval mechanisms for state description models, Technical Note 106, SRI Artificial Intelligence Center, 1975.
[10] Richard E. Fikes, Nils J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971) 189–208.
[11] Steve Hanks, Drew McDermott, Default reasoning, non-monotonic logics, and the frame problem, in: Proceedings of the AAAI National Conference, 1986, pp. 328–333.
[12] Carl Hewitt, PLANNER: A language for proving theorems in robots, in: Proc. International Joint Conference on Artificial Intelligence, 1969, pp. 295–301.
[13] Carl Hewitt, Procedural embedding of knowledge in PLANNER, in: Proc. International Joint Conference on Artificial Intelligence, 1971, pp. 167–184.
[14] Carl Hewitt, Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot, Technical Report A.I. Memo 251, MIT Project MAC, 1972.
[15] H. Kanoui, Some aspects of symbolic integration via predicate logic programming, SIGSAM Bulletin 10 (1976) 29–42.
[16] Ivan Kramosil, A note on deduction rules with negative premises, in: Proc. International Joint Conference on Artificial Intelligence, 1975, pp. 53–56.
[17] L.A. Lombardi, B. Raphael, Lisp as the language for an incremental computer, in: E.C. Berkeley, D.G. Bobrow (Eds.), The Programming Language Lisp: Its Operation and Applications, MIT Press, 1966.
[18] John McCarthy, A basis for a mathematical theory of computation, in: P. Braffort, D. Hirschberg (Eds.), Computer Programming and Formal Systems, North-Holland Publishing Company, 1967.
[19] John McCarthy, Epistemological problems of artificial intelligence, in: Proc. International Joint Conference on Artificial Intelligence, 1977, pp. 1038–1044.
[20] John McCarthy, Circumscription – a form of non-monotonic reasoning, Artificial Intelligence 13 (1980) 27–39, 171–172.
[21] John McCarthy, Applications of circumscription to formalizing common-sense knowledge, Artificial Intelligence 28 (1986) 89–116.
[22] John McCarthy, in: V. Lifschitz (Ed.), Formalization of Common Sense, Papers by John McCarthy, Ablex, 1990.
[23] John McCarthy, Patrick Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: Machine Intelligence, vol. 4, Edinburgh University Press, 1969, pp. 463–502.
[24] Drew McDermott, A deductive model of control of a problem solver, ACM SIGART Bulletin (1977) 2–7.
[25] Drew McDermott, Jon Doyle, Non-monotonic logic I, Artificial Intelligence 13 (1980) 41–72.
[26] Paul Morris, Curing anomalous extensions, in: Proc. National Conference on Artificial Intelligence, 1987, pp. 437–442.
[27] Bertram Raphael, The frame problem in problem-solving systems, in: N.V. Findler, B. Meltzer (Eds.), Artificial Intelligence and Heuristic Programming, Edinburgh University Press, 1971.
[28] Ray Reiter, On reasoning by default, in: Proc. Second Symposium on Theoretical Issues in Natural Language Processing, 1978.
[29] Ray Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[30] Raymond Reiter, Circumscription implies predicate completion (sometimes), in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1982, pp. 418–420.
[31] Erik Sandewall, Ambiguity logic as the basis for an incremental computer, Technical Report Nr. 9, Uppsala University, Department of Computer Sciences, September 1967.
[32] Erik Sandewall, The GPS expressed in LISP A, Technical Note Nr. 2, Uppsala University, Department of Computer Sciences, October 1967.
[33] Erik Sandewall, LISP A, A LISP-like system for incremental computing, in: Proc. Spring Joint Computer Conference, 1968, pp. 375–384.
[34] Erik Sandewall, Use of ambiguity logic in the picture description language, Technical Report 52, Stanford Linear Accelerator Center, GSG Group, December 1968.
[35] Erik Sandewall, An approach to the frame problem, and its implementation, in: Machine Intelligence, vol. 7, Edinburgh University Press, 1972, pp. 195–204.
[36] Yoav Shoham, A semantical approach to nonmonotonic logics, in: Proc. International Joint Conference on Artificial Intelligence, 1987, pp. 388–392.
[37] Richard M. Stallman, Gerald J. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, Artificial Intelligence 9 (1977) 135–196.
[38] Gerald Sussman, Terry Winograd, Micro-planner reference manual, Technical Report A.I. Memo 203, MIT Project MAC, 1970.
[39] Gerald Jay Sussman, Richard M. Stallman, Heuristic techniques in computer-aided circuit analysis, IEEE Transactions on Circuits and Systems, CAS 22 (11) (November 1975).
[40] David Warren, WARPLAN: A system for generating plans, Technical Report 76, Department of Computational Logic, University of Edinburgh, 1974.
[41] Terry Winograd, Procedures as a representation for data in a computer program for understanding natural language, Cognitive Psychology 3 (1) (1972) (entire issue).