LOGIC PROGRAMMING IN ARTIFICIAL INTELLIGENCE

Robert A. Kowalski
Department of Computing
Imperial College of Science, Technology and Medicine
London SW7 2BZ, UK

1. INTRODUCTION

Logic programming originated in the field of artificial intelligence. It was artificial intelligence that provided both the theorem-proving research for its backward-reasoning execution strategy [42,47,68] and its first intended applications in natural language question-answering [14]. It also provided the controversy (see e.g. [32,81]). between the relative merits of procedural versus declarative representations of knowledge, which helped to motivate the procedural interpretation of Horn clauses, which is the basis of logic programming, even today.

In this short paper I will sketch some of the subsequent developments in logic programming, concentrating especially on extensions which have been developed to make logic programming more suitable for knowledge representation in artificial intelligence. I will focus particularly on developments in non-monotonic reasoning, abduction, and metareasoning. I will also argue that the unrestricted use of full first-order logic might not be necessary or useful for most applications.

2. THE SCOPE OF LOGIC PROGRAMMING FORM

Logic programming was originally restricted to sentences (also called *rules)* in *Horn clause* form:

A if B_1 and ... and B_n

with a single atomic conclusion A and zero or more atomic conditions B_i. All variables, $X_1, \ldots, X_m$, occurring in a rule are universally quantified in front of the rule:

for all $X_1, \ldots, X_m$ [A if B_1 and ... and B_n]

Today the notion of logic programming also includes rules whose conditions are arbitrary formulae of first-order logic. Lloyd and Topor [51,52] showed how to reduce such more general rules to *normal logic programming form*, where each condition is either an atomic formula or the negation of an atomic formula. Similarly the conditions B_i in *queries*

? B_1 and ... and B_n

to a logic program can be arbitrary formulae of first-order logic. Such queries can be reduced to a normal logic program together with a normal query, in which all conditions are atomic formulae or negations of atomic formulae.

The basis of logic programming is the interpretation of rules as procedures:

reduce problems of the form A
to subproblems of the form B_1, ..., B_n.

The significance of this procedural interpretation is two-fold: Not only can declarative sentences be executed as procedures, but procedures in problem-reduction form can be interpreted declaratively as statements of logic. In this way logic programming reconciles declarative and procedural representations of knowledge.

Today many artificial intelligence applications are explicitly represented in logic programming form, and many of these are implemented in a logic programming language such as Prolog. However, many other applications are represented de *facto* as logic programs, without explicit acknowledgement of the fact. The situation calculus and its application to the Yale shooting problem [31] are among the most interesting examples. The case of the Yale shooting problem is especially interesting, because as pointed out in [2,24,25] the use of negation by failure in logic programming solves the problem of non-monotonic reasoning which arises in the example. The procedural interpretation and operational semantics of negation by failure are discussed in section 4 below.

In addition to these artificial intelligence applications explicitly or implicitly formulated as logic programs, there are many others implemented in "if-then" languages [82], which approximate logic programming form. Most of these languages, many of which are in the EMYCIN family [79], have only the expressive power of variable-free (i.e. propositional) logic programs augmented with some form of "isa-hierarchy" for taxonomic reasoning, compensating for the propositional nature of the rules.

Other applications of logic, not restricted to those exclusively associated with artificial intelligence, such as those to do with the formal specification of programs, the implementation of database systems, and the formalisation of legislation, show a similar bias towards the use of logic programming form. Applications to legislation [44,45,49,70] have special significance for knowledge representation in artificial intelligence, because the law deals with every aspect of human affairs. They also have special significance for logic programming, because they provide a rich source of material for guiding the development of its extensions. It is interesting that the extensions needed for legislation do not seem to include disjunctions in the conclusions of rules or complex forms of quantification.

3. THE IF-AND-ONLY-IF FORM OF LOGIC PROGRAMS

It seems that many applications of logic in artificial intelligence which can not be reduced to normal logic programming form can be understood instead as expressing the only-if halves of if-and-only-if definitions.

Normal logic programs, on the other hand, can be understood as expressing the if-halves of definitions. The program

```
parent(X, Y) if mother(X, Y)
parent(X, Y) if father(X, Y)
mother(mary, jack)
mother(mary, jill)
father(john, jack)
father(john, jill)
```

for example, is the if-half of the if-and-only-if definitions

```
parent(X,Y) iff [mother(X,Y) or father(X,Y)]
mother(X,Y) iff [(X=mary and Y=jack) or
                 (X=mary and Y=jill)]
father(X,Y) iff [(X=john and Y=jack) or
                 (X=john and Y=jill)]
```

augmented with appropriate axioms of equality and inequality, such as

X=X
$s \neq t$, for every pair of distinct terms s,t.

It can be argued [43] that the if-and-only-if form expresses the semantics intended by the if-form.

It seems to be the case that reasoning with the if-halves of definitions can simulate reasoning with the only-if halves and vice-versa. Consider, for example, the assertion

```
parent(x, jack)
```

where x is a constant. Reasoning forward using the only-if half of the definitions we can derive first

```
mother(x, jack) or father(x, jack)
```

then

```
(x = mary and jack = jack) or
(x = mary and jack = jill) or
(x = john and jack = jack) or
(x = john and jack = jill).
```

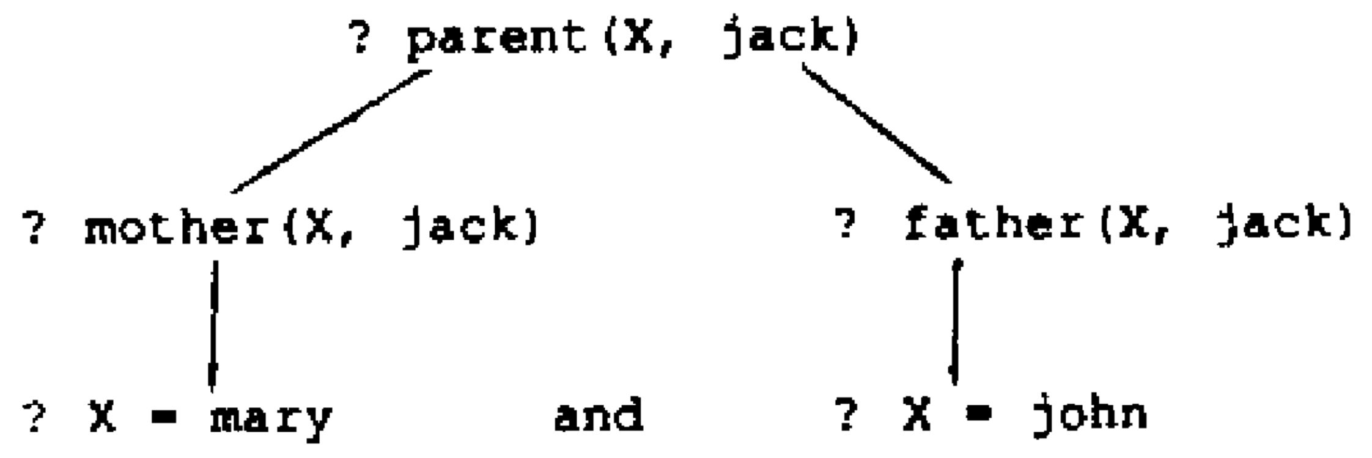
Simplifying this disjunction, we obtain the final conclusion:

```
x = mary or x = john.
```

This conclusion is dual (see e.g. [16]) to the one we obtain by reasoning backward, logic programming style, and deriving all answers to the query

```
? parent(X, jack)
```

where X is a variable:



Thus backward reasoning using the if-halves of the definitions simulates forward reasoning with the only-if halves. Moreover implicit reasoning with equality by means of unification simulates explicit reasoning with equality and inequality axioms.

The simulation of only-if halves of definitions by if-halves is also the basis for Clark's result [9] that negation by failure is a correct implementation of classical negation.

4. NON-MONOTONIC REASONING IN LOGIC PROGRAMMING

The execution of negative conditions in logic programs is performed by *negation by failure*:

not P holds if and only if
P fails to hold.

Clark [9] showed that finite failure to demonstrate P using the if-halves of definitions simulates proof of not P using the only-if halves. Reiter [66] showed that the if-and-only-if form (also called the *completion* or *Clark-completion*) of a logic program is "sometimes" equivalent to McCarthy's circumscription [54].

Despite the attractions of the completion as a semantics for negation by failure, it has a number of limitations, which have been addressed by subsequent investigators. Kunen [50] and Fitting [27] for example have proposed three-valued logic for the semantics of the completion. Apt, Blair and Walker [2], Przymuszynski [63,64] and van Gelder [76,77] have proposed extensions of the least fix point and minimal model semantics originally developed for Horn clauses by van Emden and Kowalski [75].

More recently Gelfond [28] showed that negation by failure in logic programming can be interpreted in Moore's autoepistemic logic [58]. A rule of the form

A if B_1 and ... B_n not C_1 and ... not C_m

where B_i and C_i are atomic formulae, can be interpreted as a sentence

A if B_1 and ... B_n and not LC_1 and ... not LC_m

where LC_i means C_i is believed, and therefore not LC_i means C_i is not believed. Gelfond and Lifschitz subsequently showed how to adapt the semantics of autoepistemic logic to obtain a direct *stable model semantics* [29] for normal logic programs.

Marek and Truszczynski [55] showed how negation by failure can be interpreted in Reiter's default logic [65]. A rule of the form given above is interpreted as a default rule

$$\frac{B1, \ldots, Bn: \mathbf{M} \text{ not } C1, \ldots, \mathbf{M} \text{ not } Cm}{A}$$

Eshghi and Kowalski [24] have shown that negation by failure can be interpreted as a case of abduction:

not C_1, ... , not C_m

are hypotheses which are assumed to hold, provided there is no evidence to the contrary. The abductive interpretation of negation by failure has been developed further by Kakas and Mancarella [37] and Dung [22].

5. ABDUCTION

Abduction has many applications in artificial intelligence, including fault diagnosis [8,62], image recognition [19], plan formation [23], plan recognition [8], temporal reasoning [71], and natural language understanding [8,74]. It can also be used for knowledge assimilation [38,39] and default reasoning [24,61].

Various strategies have been developed for generating abductive hypotheses. Most of these are based on resolution [19,26]. The ATMS approach developed by Reiter and deKleer [67] for propositional Horn clauses also uses a form of resolution, together with subsumption, called *consensus*. The use of subsumption guarantees that the hypotheses which are generated are minimal.

Similar strategies have also been developed for generating conditional answers within a logic programming "query-the-user" framework [69]. Instead of failing in a proof when a condition selected for execution fails to unify with the conclusion of any rule or can not be answered by the user, the condition is set aside as a hypothesis for the answer. Thus, for example, given the program

```
wobbly-wheel if broken-spokes
wobbly-wheel if flat-tyre
flat-tyre if punctured-tube
flat-tyre if leaky-valve
```

and the observed conclusion

```
wobbly-wheel
```

backward reasoning eventually reduces the conclusion to the hypotheses

```
broken-spokes
punctured-tube
leaky-valve
```

each of which is an alternative explanation of the observation.

In the general case such hypotheses have to be tested for compatibility with integrity constraints. In most applications the use of backward reasoning makes it unnecessary to test hypotheses for minimality.

Console, Theseidre Dupré and Torasso [15] have noted that abductive reasoning using the if-halves of if-and-only-if definitions can be replaced by deduction using the only-if halves. For example, using the only-if halves of the completion

```
wobbly-wheel iff [broken-spokes or flat-tyre]
flat-tyre iff [punctured-tube or leaky-valve].
```

forward reasoning from the observation

```
wobbly-wheel
```

generates the disjunction

```
broken-spokes or punctured-tube or leaky-valve.
```

Thus we see another example of a correspondence between if-halves and only-if halves of the if-and-only-if form of logic programs. Such examples give added support to the thesis that logic programming appropriately extended might provide a general basis for knowledge representation and reasoning in artificial intelligence.

6. "REAL" NEGATION

Recently, extensions of logic programming have been developed in which the conclusions of rules can be negations of atomic formulae. Gelfond and Lifschitz [29] in particular have shown how to extend the stable model semantics to allow logic programs to contain both "real" negation, "¬", as well as negation by failure, "not". Kowalski and Sadri [48] have adapted their semantics so that rules with negative conclusions are interpreted as exceptions to rules with positive conclusions.

Applied to the well-known example

```
 fly(X) if bird(X)
¬fly(X) if ostrich(X)
 bird(X) if ostrich(X)
 ostrich(tom)
```

the semantics gives the result

```
¬fly(tom)
```

but not the result

```
fly(tom).
```

Equivalent results can also be obtained by combining both forms of negation:

```
 fly(X)  if bird(X) and not ¬ fly(X)
¬fly(X)  if ostrich(X)
 bird(X) if ostrich(X)
 ostrich(tom).
```

Under this adaptation of the stable model semantics, both formulations are essentially equivalent to a conventional formulation in normal logic programming form

```
fly(X)  if bird(X) and not ab(X)
ab(X)   if ostrich(X)
bird(X) if ostrich(X)
ostrich(tom)
```

where the negative predicate ¬ fly(X) is renamed as a positive predicate ab(X).

Thus under this semantics, programs combining real negation and negation by failure can be transformed back into normal logic programs with only negation by failure. Nonetheless, empirical studies of the language of legislation [44] suggest that the extension of logic programming to include negation in the conclusion of rules is essential for naturalness of expression in practice.

7. METALOGIC PROGRAMMING

Metaprogramming, in which programs, databases, and "theories" in general are manipulated as data, is an important technique in logic programming methodology [5,43,73]. It is used for such applications as program transformation and verification, knowledge base management, and the implementation of expert system shells. It is commonly used to overcome the limitations of Prolog's simple execution strategy and to implement more sophisticated execution methods.

Metalogic programming is usually carried out with the aid of a one-argument predicate

```
solve(X),  which holds when the goal X
           can be solved,
```

or with a two-argument predicate

```
demo(X, Y),   which holds when the goal Y
              can be solved (or demonstrated)
              using the program, database,
              or theory X.
```

It is common to augment the metapredicate with extra arguments representing such entities as proof, uncertainty, or time.

The metapredicate can be implemented by reflection rules as in Weyhrauch's FOL [80] or Costantini and Lanzaroni's [18] Reflective Prolog. It can, and more usually is, implemented by means of a metainterpreter, as in the following case where the extra argument represents a time point:

```
demo(X, Y, T)      if   demo(X, Y ← Z, T)
                   and  demo(X, Z, T)
demo(X, Y ∧ Z, T) if   demo(X, Y, T)
                   and  demo(X, Z, T)
demo(example, mortal(X) ← human (X), T)
demo(example, human(socrates), T)
                   if 380 b.c. ≤ T
demo(example, human(turing), T)
                   if 1912 a.d. ≤ T
```

Here "←" and "∧" are infix function symbols naming "if" and "and". The constant symbol "example" names an object level theory.

The computational overheads of running the metainterpreter can be alleviated by "top-down" partial evaluation [74a] or "bottom-up" data-driven, transformation [17]. In the example above, the five rules can be replaced by three:

```
demo(example, mortal(X), T)
              if demo(example, human(X), T)
demo(example, human(socrates), T)
              if 380 b.c. ≤ T
demo(example, human(turing), T)
              if 1912 a.d. ≤ T
```

These rules can be simplified further. If we rename predicates, replacing

```
demo(example, mortal(X),T) by  mortal*(X,T)
demo(example, human(X),T)  by  human*(X,T)
```

we obtain the essentially equivalent object level rules:

```
mortal*(X, T) if human*(X, T)
human*(socrates, T) if 380 b.c. ≤ T
human*(turing, T) if 1912 a.d. ≤ T
```

Metalogic programming can also be used to implement more powerful object level reasoning, for example by means of such metarules as

```
demo(X, Y)  if  demo(X, Y ∨ Z)
            and demo(X, ¬ Z)
```

where "∨" and "¬" are function symbols naming disjunction and negation respectively. Such use of a restricted metalanguage to implement and reason about a more powerful object language is reminiscent of Hilbert's program to use a finitary metatheory to justify non-constructive mathematics (see e.g. [40]).

The proof predicate

```
demo(X, Y)
```

can also be used to represent belief:

```
X believes Y.
```

Such an interpretation of belief as provability in artificial intelligence has been advocated in different ways by Konolige [41] and Perils [603 - The use of the demo predicate within a logic programming framework to represent multi-agent knowledge and belief has been studied by Kowalski and Kim [46]-

The use of metalogic is essential also when logic programming is used to represent legislation. It is needed, for example, to represent situations where one statute refers to another, or where one provision refers to another provision of the same statute. It is needed also [45] to represent explicitly the executive agency's reasoning process.

Metalogic gives much of the power of higher order logic. Conversely, highereorder logic, as incorporated in Miller's X^Prolog [56] for example, can also be used for roetaprogramming.

8- TERMINQLOGTPAL REACHING

Beginning with the language KLONE [6], there has been much interest in recent years in languages and logics specifically designed for "terminological reasoning". In these logics, terms denote sets, and logical operators such as conjunction, disjunction, negation and quantification, which can be applied to terms, denote operations on sets- The important logical properties of such structured terms are whether a term is *satisfi&ble* (denotes a non-empty set) and whether one term subsumes another (denotes a set which includes the set denoted by the other}.

Ait Kasi and Nasr [1] have shown how to extend logic programming to include structured terms- Such extensions combine the advantages of logic programming with those of terminological reasoning. More recently, Biirkert [7] and Hohfeld and Smolka [33] have shown that such a combination of logic programming and terminological reasoning can be obtained by incorporating equations over structured terms as constraints within a constraint logic programming framework,

9, CONSTRAINT *hOGTC* PROGRAMMING

In constraint logic programming, the conditions of rules are partitioned into two kinds♦ One kind is executed normally by backward reasoning. The other kind is treated in a domain-specific manner as a constraint. Constraints are simplified and tested for satisfiability using algorithms specific to the given problem domain.

The first language incorporating an early notion of constraint logic programming was Colmerauer's Prolog II [13], since extended to Prolog 111 [121 ♦ The underlying theory was developed by Jaffar, Lassez, and Maher *[36]* * Some of the most successful applications have been implemented using CHIP [21], the language developed at ECRC.

Constraint logic programming has proved to be a fruitful paradigm for integrating logic programming with special-purpose problem-solving mechanisms for mathematical programming, functional programming, and finite domains. It may be that many of the special-purpose problem solving methods developed in artificial intelligence can usefully be integrated with logic programming in this way*

10. FpftMft^ METHODS

Logic *has* traditionally been used to formalise program specifications for program verification and synthesis. For such purposes, logic programming has the advantage over other programming approaches that programs and specifications are written in the same logical formalism- Moreover program execution, verification, and synthesis can all be performed using similar logical reasoning techniques. These characteristics of logic programming have motivated many investigations (e.g. [10,11,34,35]) into the problem of deriving efficient logic programs from more obviously correct, but inefficient, logical specifications. These studies are important for artificial intelligence, because they show how rigorous, formal methods can be applied to artificial intelligence applications.

1U LEfcRMIHG

it is not always possible or convenient to rigorously formulate a program specification, either before or after writing a program. In many such cases, however/ it is natural to specify the intended program informally by means of examples* The simple form of logic programs makes them especially amenable to such learning methods,

Ehud Shapiro's early work [72] was specifically oriented toward learning logic programs* Many other learning methods generate logic programs implicitly without explicitly acknowledging that fact. It has been shown, for example, that explanation-based learning can be viewed as partial evaluation of logic programs

Recent work (see e.g. Muggleton [59] and De Raedt [20]) has begun to show that non trivial logic programs can be generated automatically from examples+ Techniques which treat negative examples as exceptions to rules that generalise positive examples [41 seem to be especially appropriate for formulation as logic programs combining real negation with negation by failure.

12, CONCLUSION

I have argued that, to be more useful for knowledge representation and reasoning in artificial intelligence, normal logic programming needs to be extended to include such additional features as real negation, abduction, metareasoning, terminological reasoning and constraint solving. Other extensions such as temporal reasoning and uncertainty can be implemented conveniently by metalogic programming techniques. Disjunctive logic programming, in which conclusions of rules can be disjunctions, has also been developed, notably by Loveland [53] and Minker [57] and their colleagues, Characteristic applications for these systems remain to be identified.

The field of logic programming enjoys a healthy interaction between its theory and its practice. It also enjoys good connections with diverse areas of computing including database systems, formal methods and artificial intelligence, as well as with areas outside of computing including mathematics and legal reasoning. In my opinion, however, it is the links which logic programming retains with artificial intelligence and the emerging links with legal reasoning which will be most important for the development of logic programming in the future.

ACKKNOWLEDGEMENTA

This research was supported by ESPRIT Basic Research Project "Compulog" and by Fujitsu. I am grateful to Phan Minh Dung and Tony Kakas for helpful comments on a draft of this paper-

REFERENCES

[1] Ait Kaci, H. and Nasr, R. [1966]: LOGIN: "A logic programming language with built-in inheritance", Journal of Logic Programming, Vol. 3, pp. 185-215.

[2] Apt, K.R. and Bezem, M. [1990]: "Acyclic programs", Proc. of the Seventh International Conference on Logic Programming, MIT Press, pp. 579-597.

[3] Apt, K., Blair, H. and Walker, A. [1987]: "Towards a theory of declarative knowledge*'. In J. Minker, editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, C-A. pp. 89-142.

[4] Bain, M. and Muggleton, S. [1990]: "Non-monotonic Learning" Machine Intelligence 12, Oxford University Press.

[5} Bowen, K.A. and Kowalski, R.A. [1982]: "Amalgamating Language and Metalanguage in Logic Programming**, in Logic Programming (Clark, K.L. and Tarnlund, S.-A., editors), Academic Press, pp. 153-173.

[6] Brachmann, R.J. and Levesque, H.J. [1984]: "The tractability of subsumption in frame based description languages**, Proceedings of the Fourth National Conference of the AAAI, pp. 34-37.

[7] Burckert, JH.J.[1990]: "A resolution principle for clauses with constraints", Proc. 10th CADE, LNAI 449, pp. 178-192.

[8] Charniak, E. and McDermott, D. [1985]: "Introduction to Artificial Intelligence", Addison-Wesley,

[9] Clark, K.L. [1978]: "Negation by failure", in "Logic and databases", Gallaire, H. and Minker, J. [eds], Plenum Press, pp. 293-322.

[10] Clark, K.L. and Darlington, J. [1980]: "Algorithm classification through synthesis", Computer J. pp. 61-65.

[11] Clark, K.L. and Tarnlund, S.-A. [1977]; "A first- order theory of data and programs", In Proc IFIP 1977. Amsterdam: North-Holland, pp. 939-944.

[12] Colmerauer, A. [1990]: "An introduction to PROLOG III", Comm. ACM, pp. 70-90.

[13] Colmerauer, A., Kanoui, H. and Caneghem, M.V. [1983]: "Prolog, theoretical principles and current trends", Technology and Science of Informatics, Vol. 2, No. 4, pp. 255-292.

[14] Colmerauer, A., Kanoui, H., Pasero, R. and Roussel, P. [1973] "un systeme de communication homme-machine en Francais", Groupe d'Intelligence Artificielle, Univ. d'Aix Marseille II, Luminy, France.

[15] Console, L-, Theseider Dupre, D. and Torasso, P. [1990]: "A completion semantics for object-level abduction", Proc. AAAI Symposium on Automated Abduction, Stanford, March 1990.

[16] Copi, I.M. [1954]: "Symbolic Logic", The MacMillan Company, New York.

[17] Cosmodopoulos,Y., Sergot, M., and Southwick, R.W. [1991]: "A general data-driven transformation of meta-interpreters", Department of Computing, Imperial College, London.

[18] Costantini, S, and Lanzarone, G. A. [1989]: "A metalogic programming language", Proc. Sixth International Conference on Logic Programming, MIT press, pp,218-233.

[19] Cox, P.T. and Pietrzykowski, T. [1986] : "Causes for Events; Their Computation and Applications", in Proceedings CADE-86, pp 608-621.

[20] De Raedt, L. 11991} : Interactive concept-learning. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven,

[21] Dincbas, M., van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Berthier. F. [1988] "The constraint logic programming language CHIP", In Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88, pp. 693-702.

[22] Dung, P.M. [1991] "Negations as hypotheses: an abductive foundation for logic programming", Proc. ICLP-91, MIT Press.

[23] Eshghi, K. [1988]; "Abductive planning with event calculus", Proceedings of the 5th International Conference on Logic Programming, MIT press.

[24] Eshghi, K. and Kowalski, R.A. (1989] : "Abduction compared with negation by failure", proceedings of the Sixth International Logic Programming Conference, MIT Press, pp. 234-255.

[25] Evans, C [1989]: "Negation-as-failure as an approach to the Hanks and McDermott problem"/ Proc. Second International Symposium on Artificial Intelligence, Monterrey, Mexico.

126] Finger, J-J. and Geneaereth, M.R, (19851: "RESIDUE: A Deductive Approach to Design Synthesis", Stanford University Report No. CS-85-1035.

[27] Fitting, M. [1985]; "A Kripke-Kleene semantics for logic programs". Journal of Logic Programming, Vol- 2- No. 4. pp 295-312.

[28] Gelfond, M, [1987]: "On stratified autoepistemic theories", In Proceedings AAAI-87, American Association for Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, pp. 207-211.

[29] Gelfond, M. and Lifachitz, V. (1908]: "The stable model semantics for logic programs", Proceedings of the Fifth International Conference and Symposium on Logic Programming, Mil press pp. 1070-1080.

[30] Gelfond, M. and Lifschitz, V, [1990]: "Logic programs vith classical negation", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 579-597.

[31] Hanks, S. and McDermott, D. [1986]: "Default reasoning, non-monotonic logics, and the frame problem", Proc. AAAI, Morgan and Kaufman, pp, 328-333.

[32] Hewitt, C [1969]: "PLANNER; A language for proving theorems in robots*', In Proceedings of IJCAI-1 (Washington, D,C), pp. 295-301.

[33] Hofeld, M. and Smolka, G. [1988]: "Definite relations over constraint languages", LILOG Report 53, IWBS, IBM Deutschland, Stuttgart, Germany. To appear in the Journal of Logic Programming.

[34] Hogger, C.J. [1981]: "Derivation of logic programs", J. Ass. Comput. Mach. Vol. 28, pp. 372-322.

[35] Hogger, C.J. (1984]: Introduction to Logic Programming, Academic Press, London.

[36] Jaffar, J., Lassez, J. and Maher, M, [1987]: "Constraint Logic Programming" in Proc. of 14th ACM Symp. POPL.

[37] Kakaa, A.C. and Mancarella, P. [1990], "Generalised stable models: a semantics for abduction", Proceedings of ECAI 90, pp. 385-391.

[38] Kakas, A.C. and Mancarella, P. [1990J, "Database updates through abduction", Proceedings of VLDB 90.

[39] Kakaa, A-C. and Mancarella, P. [1990]: "Knowledge Assimilation and Abduction", Proceedings of the ECAI-1990 Workshop on Truth Maintenance Systems, Springer-Verlag, 1990.

[40] Kleene, S.C. [1952]; "Introduction to Metamathematics", D. van Nostrand Co., Princeton.

[41] Konolige, K. 11986}: "A Deduction Model of Belief", Pitman Research Notes in Artificial Intelligence.

[42] Kowalski, R.A. {1974]: "Predicate logic as programming language", In Proceedings of IFIP 1974 (Stockholm, Sweden). North-Holland, Amsterdam, pp. 569-574).

[43] Kowalski, R.A. [1979]: Logic for problem solving. New York: Elsevier

[44] Kowalski, R.A. [1989]: "The treatment of negation in logic programs for representing legislation", Proceedings of the Second International Conference on Artificial Intelligence and Law, pp. 11-15.

[45] Kowalski, R.A. [1991]: "Legislation as logic programs", Department of Computing, Imperial College, London.

{46] Kowalski, R.A. and Kim, J.S. 11991}; "A metalogic programming approach to multi-agent knowledge and belief", to appear in Artificial Intelligence and Mathematical Theory of Computation (V. Lifschitz, ed.) Academic Press-

[47] Kowalski, R.A. and Kuehner, D. [1971] "Linear resolution with selection function". Artif, Intell. Vol. 2, pp. 227-260.

[48] Kowalski, R.A. and Sadri, F. [1990}: "Logic programs with exceptions", Proceedings of the Seventh International Conference on Logic Programming, MIT Press, pp. 598-613.

[49] Kowalski, R A., Sergot, M.J. [1990]: "The use of logical models in legal problem solving", Ratio Juris, Vol. 3, No. 2, pp. 201-218.

[50] Kunen, K. [1987]: "Negation in logic programming". Journal of Logic Programing, Vol. 4, No. 4, pp. 289-308.

[51] Lloyd, J.W. and Topor, R.W. [1984]: "Making Prolog more expressive", Journal of Logic Programming, Vol. 3, No. 1, pp. 225-240.

[52] Lloyd J.W. [1987] *t* "Foundations of logic programming", second extended edition, Springer-Verlag.

[53] Loveland, D.W. [1987]: "Near-Horn Prolog" Logic Programmingt Proceedings of the Fourth International Conference, MIT Press, pp 456-469.

[54] McCarthy, J, [1980]: "Circumscription - a form of nonmonotonic reasoning*, Artificial Intelligence, Vol. 26, No. 3, pp. 89-116.

(55] Marek, W. and Trusiczynski, M. [1989]: "Stable semantics for logic programs and default theories", Proc. NACLP-89, MIT Press.

[56] Miller, D. and Nadathur, G. [1966]: "Higher-order logic programming", Proceedings of the Third International Conference on Logic Programming,. Springer-Verlag, pp. 448-462.

157] Minker, J. [1989]: "Toward a foundation of disjunctive logic programming" Logic Programming: Proceedings of the North American Conference, MIT Press, pp. 1215-1235.

[58] Moore, R.C. 11985]: "Semantical considerations on nonmonotonic logic in artificial intelligence", Artificial Intelligence, Vol, 25, pp. 75-94.

[59] Muggleton, S. [1991]: "Inductive logic programming", New Generation Computing, Vol. B, pp. 295-318.

[60] Perils, D. [1988]: "Language with Self-Reference II: Knowledge, Belief and Modality", Artificial Intelligence *Vol,* 34, pp. 179-212.

[61] Poole, D. [19B8]: "A logical framework for default reasoning", Artificial Intelligence Vol, 36, pp. 27-47,

162] Pople, H.E., Jr. [1973]: "On the mechanization of abductive logic" Proc. Third IJCAI, Stanford, Ca. pp 147-152.

[63] Przymuszynski, T. [1987]: "On the declarative semantics of stratified deductive databases and logic programs", in J. Minker, editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, CA., pp 193-216

[64] Przyntuszyrrski, T. [1989]: *"Non-monotonic* formalisms and logic programming", Proceedings of the Sixth International Logic Progamming Conference, MIT Press, pp. 655-674.

[65] Reiter, R. [1980] ; "A logic for default reasoning", Artificial Intelligence, Vol. 13, pp. 81-132.

[66] Reiter, R. [1982]: "Circumscription implies predicate completion (sometimes}", Proceedings of the National Conference on Artificial Intelligence, Pittsburgh Pa.

[67] Reiter, R. and deKleer J. [1987]: "Foundations of assumption-based truth maintenance systems: preliminary report", Proc. AAAI-87, Seattle, pp. 183-188.

[68] Robinson, J.A- [1565]: "A machine-oriented logic based on the resolution principle", J. ACM 12, Vol- 1, pp, 23-41.

[69] Sergot, M. [1982]: "A query-the-user facility for logic programming", In Proc. ECICS, Stresa, Italy (eds. P. Degano & E. Sandewall) Amsterdam: North-Holland, pp. 27-41.

[70] Sergot, M.J., Sadri, p., Kowalski, R.A., Kriwaczek, F., Hammond, P. and Cory, H.T. [1986]: "The British Nationality Act as a logic program", CACM, Vol. 29, No. 5, pp. 370-336.

[71] Shanahan, M. [19B9]: "Prediction is deduction but explanation is abduction", IJCAI 89.

[72] Shapiro. E.y. [1983]: Algorithmic Program Debugging. The MIT press.

[73] Sterling, L. and Shapiro, E.Y [1966]: "The Art of Prolog", MIT Press.

[74] stickel, M.E. [1968]: "A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation", Proc. International Computer Science Conference (Artificial Intelligence: Theory and Applications) (J,-L. Lassez and P.Y.L. Chin, editors), pp. 343-350.

[74a]Takeuchi, A. and Furukawa, K. [1986]: "Partial evaluation of Prolog programs and its application to metaprogranuning", Proc, IFIP, North Holland, pp. *415-420,*

[75] van Emden, M.H. and Kowalski, R.A. [1976]: "The semantics of predicate logic as a programming language", J. ACM, Vol. 23, No. 4 (Oct. 1976), pp. 733-742.

[76] van Gelder, A, [1987]: "Negation as failure using tight derivations for general logic programs". In J. Minker, editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, C.A., pp. 149-176.

[77] van Gelder, A., Rosa, K. and Schlipf. [1988]: "Unfounded sets and well-founded semantics for general logic programs". In Proceedings of the Symposium on Principles of Database Systems, ACM SIGACT-SIGMOD.

[78] van Harmelen, F. and Bundy, A. [1968] : *"Explanation* based generalization - partial evaluation", Artificial Intelligence, Vol. 36, pp. 401-412.

[79] van Melle, W.J. [1980]: System aids in constructing consultation programs, UMI Press, Ann Arbor, Mi.

[80] Weyhrauch, R.W. [1980]: "Prolegomena to a theory of mechanized formal reasoning", Artificial Intelligence Vol. 13, pp. 133-170.

[81] Winograd, T. [1972]: Understanding natural language, Academic Press, New York.

[82] Winston, P.H. [1984]: "Artificial Intelligence", second edition, Addison Wesley.