Editor's Foreword

This issue of Fundamenta Informaticae is devoted to the theory of logic programming. To put the papers which appear here in a proper perspective, let us discuss briefly the most relevant aspects of this subject.

Theory of Positive Programs

From the viewpoint of mathematical logic, logic programming is a proof theory and model theory of Horn clauses and some of their extensions. It started its life as an independent discipline around 1973 when Colmerauer and Kowalski realized that Horn clauses can be used as a programming language. A theoretical framework for this idea was published in Kowalski [1974].

A *logic program* consists of a finite set of *definite clauses* (i.e. Horn clauses with one positive literal) and is activated by a *goal* (i.e. a Horn clause with no positive literal). Definite clauses are written as

$$A_0 \leftarrow A_1, ..., A_n$$

where $n \geq 0$, and should be interpreted as

$$(A_1 \wedge ... \wedge A_n \rightarrow A_0),$$

whereas goals are written as

$$\leftarrow A_1, ..., A_n$$

where $n \geq 0$, and should be interpreted as the question

$$\exists(A_1 \wedge ... \wedge A_n)?$$

We shall also call logic programs *positive programs*, or simply *programs*.

The activation of a logic program leads to an attempt at refuting the goal. If it succeeds, it produces a constructive answer to the question expressed in the goal. This process generates a sequence of goals in which each consecutive goal is derived by computing a resolvent of the previous goal with one of the program clauses. To this purpose an atom is selected in each goal. This refutation process is called SLD-resolution (SLD stands for *s*election rule driven *l*inear resolution for *d*efinite clauses).

This interpretation of logic programs is called a *procedural semantics* which explains *how* a program computes, i.e. what is the computation mechanism which underlies the program execution. When in each goal the first atom from the left is selected and the search for a solution proceeds in a depth-first fashion, the programming language $PROLOG$ is obtained.

Another interpretation of logic programs is obtained by studying them from the model-theoretic point of view. With each logic program P the least Herbrand model M_P can be associated in a natural way. This model is the least fixpoint of a natural immediate consequence operator T_P introduced by Van Emden and Kowalski [1976]

which, given a set of ground atoms, produces the set of their direct consequences by means of the clauses which form the program P. Since T_P is continuous, $M_P = T_P \uparrow \omega$, where $T_P \uparrow \omega$ is the result of the iteration of T_P upward by ω steps.

The model M_P provides the meaning of a program P and allows us to explain what semantically follows from the program P. Thus it provides us with a *declarative semantics* in that it explains *what* the program computes without analyzing the underlying computational mechanism.

Completeness theorems link the above two semantics of logic programs and can be proved in a number of versions. The so-called *strong completeness theorem* shows that a goal semantically inconsistent with the program can be refuted by SLD-resolution for any given selection rule. However, this result does not imply completeness of $PROLOG$ which, for efficiency reasons, uses a depth-first search in the corresponding derivation tree (called an SLD-tree). These completeness results were proved by Hill [1974], Clark [1979], Apt and Van Emden [1982], and others.

To extend the expressiveness of logic programs, Colmerauer suggested to use negative literals in the goals and proposed a computational mechanism allowing us to resolve them. Clark [1978] formulated this mechanism as a *negation as failure* rule and investigated its semantics. The negation as failure rule allows us to infer $\neg A$ if for some selection rule a finite and failed SLD-tree with the goal $\leftarrow A$ can be constructed. To study the semantics of the negation as failure rule, the declarative interpretation of logic programs through the model M_P turns out to be inadequate since $\neg A$ holds in M_P iff $\leftarrow A$ cannot be refuted from P and the latter is in general a weaker statement than "$\neg A$ can be inferred using the negation as failure rule". Clark [1978] proposed therefore to interpret logic programs by means of a construction called the *completion of* P (written as $comp(P)$) which transforms the original set of *if*-clauses into a set of *iff*-statements. This construction introduces the equality predicate and therefore additionally so-called *free equality axioms* are needed here to ensure the appropriate interpretation of equality. These axioms enforce that non-unifiable terms are assigned to different elements in a model.

Jaffar, Lassez and Lloyd [1983] proved completeness of the negation as failure rule with respect to the completion of the program. Thus the negation as failure rule allows us to prove exactly those negative literals which are semantically implied by the completion of the program. Earlier Apt and Van Emden [1982] characterized the negation as failure rule in terms of the complement of $T_P \downarrow \omega$, the result of the iteration of T_P downward by ω steps. They also noted that the pre-fixpoints of the operator T_P are exactly all Herbrand models of the program P whereas the fixpoints of T_P are exactly all Herbrand models of $comp(P)$. Putting these and some other related results together one gets an interesting duality between the concepts of SLD-refutation and the negation as failure rule.

Theory of General Programs

By general programs we mean here an extension of positive programs in which negative literals are allowed in the bodies of the clauses. An extension of the above results to the case of general programs presented a number of difficulties which only

recently have been resolved. First of all, the completion of a general program can be inconsistent (just take $P = \{A \leftarrow \neg A\}$; then $A \leftrightarrow \neg A$ is in $comp(P)$). Secondly, the appropriate extension of SLD-resolution introduced by Clark [1978] and called $SLDNF$-resolution (for SLD-resolution with the negation as failure rule), has to be properly defined in order to avoid circularity and reduction to undefined cases.

One line of research proposed by Apt, Blair and Walker [1988] and Van Gelder [1988] resulted in the isolation of a natural subclass of general programs called *stratified programs* whose completion is consistent. Stratified programs are general programs in which recursion "through" negation is disallowed. They form a simple generalization of a class of database queries introduced by Chandra and Harel [1985].

Stratified programs have a natural semantics associated with them in the form of a minimal Herbrand model. This model is built iteratively using smallest Herbrand models and forms a natural generalization of the model M_P. Przymusinski [1988] found an interesting characterization of this model as the unique minimal Herbrand model in a specific partial ordering $<$ on the models of P and termed this model a *perfect model.* In this partial ordering $N < M$ if N is obtained from M by minimizing some of the relations defined "lower" in P even at the cost of extending some of the relations defined "higher" in P. With this model chosen as a declarative semantics for a stratified program P, Przymusinski [1988] proved a completeness result which refers to a non-effective form of resolution, called SLS-resolution, in which a negative literal $\neg A$ is considered proved if A cannot be proved.

Another line of research, based on the use of 3-valued logic, was advocated by Mycroft [1984], Fitting [1985] and Kunen [1987]. In the 3-valued semantics, for any ground atom, the value **true** corresponds to being provable, the value **false** to being refuted and the value **u** (for *u*ndefined) to divergence. With an appropriate interpretation of the logical connectives $comp(P)$ is always consistent in the 3-valued semantics.

Kunen [1987] showed completeness of $SLDNF$-resolution with respect to this interpretation of $comp(P)$. No restriction to stratified programs is needed here, though (similarly as in the case of the SLS-resolution) one needs to ensure that only ground negative literals are resolved. The proof relies on a natural generalization by Fitting [1986] of the operator T_P to its 3-valued counterpart Φ_P. Φ_P is monotonic (though not continuous) and for every formula ϕ, ϕ has value **true** in all 3-valued models of $comp(P)$ iff ϕ has value **true** in $\Phi_P \uparrow n$ for some finite n. This is a crucial lemma since it allows us to carry out the completeness proof by an induction set up in a suitable way.

We thus see that two natural views of stratified programs arise here, depending on what declarative semantics one chooses.

Short Summary of this Issue

The first four papers deal with general programs.

Apt and Blair study in their paper the recursion-theoretic complexity of perfect models. They show among others that for a stratified program with n strata its perfect model is Σ^0_n and that for each $n > 0$ there exists a stratified program with n

strata whose perfect model is Σ_n^0 complete. Using these results they obtain a sir characterization of the recursion-theoretic complexity of a number of formal dealing with non-monotonic reasoning.

Fitting and Jacobs study 3-valued semantics of stratified programs. One of t is obtained by considering the operator Φ_P discussed in the previous section. Sir is monotonic, by the Knaster-Tarski theorem it has a least fixpoint. It turns out for stratified programs this least fixpoint coincides with the outcome of an iter construction in which for each successive stratum the least and greatest fixp of the T_P operator associated with the current stratum are used. Consequently latter construction is independent on the stratification of the considered progr

Kunen in his paper extends his completeness result discussed in the pre section by including in the language a number of *PROLOG* features. The exten discussed include the numeric and term comparison operators. The proper han of these constructs requires an appropriate modification of the completion *comp* Finally he proves that any general program computing transitive closure either negative literals (i.e. is not positive) or uses function symbols.

Przymusinski and Przymusinska study in their paper a natural extension o iterative construction which in the case of stratified programs leads to the pe model. This construction takes dynamically into account which literals are "ir vant". In this construction the decomposition into strata is performed dynami rather than statically. The semantics obtained by this process is applicable larger class of general programs than the stratified ones.

One of the most striking aspects of the theory of logic programming is tha a positive program P its immediate consequence operator T_P is upward contin but does not need to be downward continuous. This fact has been first notice Clark, Andreka and Nemeti (see Apt and Van Emden [1982]). However, exan of such programs seemed artificial and difficult to construct. Bagai, Bezem and Emden provide in their paper a number of natural positive programs whose ir diate consequence operator is not downward continuous and clarify this proble relating it to certain concepts of graph theory.

I hope that this special issue will raise the reader's interest in the theo logic programming. Those wishing to get a systematic treatment of the subjec advised to consult Lloyd [1987] or Apt [1988].

Krzysztof R. Apt

References

1. Apt, K.R., Introduction to Logic Programming, Report CS-R8826, Cent Mathematics and Computer Science, Amsterdam, to appear in: *Handbo Theoretical Computer Science* (J. van Leeuwen, ed.), North Holland, Am dam, 1988.

2. Apt, K.R., Blair, H. and Walker, A., Towards a Theory of Declarative Kı edge, in: *Foundations of Deductive Databases and Logic Programmin*

Minker, ed.), Morgan Kaufmann, Los Altos, 1988.

3. Apt, K.R. and van Emden,M.H., Contributions to the Theory of Logic Programming, *JACM, vol. 29, No. 3,* 1982, pp. 841-862.

4. Clark, K.L., Negation as Failure, in: *Logic and Databases*, (H. Gallaire and J. Minker, Eds.), Plenum Press, New York, 1978, pp. 293-322.

5. Clark, K.L., Predicate Logic as a Computation Formalism, *Research Report* DOC 79/59, Department of Computing, Imperial College, 1979.

6. Chandra, A. and Harel, D., Horn Clause Queries and Generalizations, *Journal of Logic Programming*, vol. 2, no. 1, 1985, pp. 1-15.

7. van Emden, M.H. and Kowalski, R.A., The Semantics of Predicate Logic as a Programming Language, *JACM*, vol. 23, no. 4, 1976, pp. 733-742.

8. Fitting, M., A Kripke-Kleene Semantics for General Logic Programs, *Journal of Logic Programming*, vol. 2 , no. 4, 1985, pp. 295-312.

9. van Gelder, A., Negation as Failure Using Tight Derivations for General Logic Programs, in: *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, 1988.

10. Hill, R., LUSH-Resolution and its Completeness, *DCL Memo 78*, Department of Artificial Intelligence, University of Edinburgh, 1974.

11. Jaffar, J., Lassez, J.-L. and Lloyd, J.W., Completeness of the Negation as a Failure Rule, in: *IJCAI-83*, Karlsruhe, 1983, pp. 500-506.

12. Kowalski, R.A., Predicate Logic as a Programming Language, in: Proc. *IFIP 74*, 1974, pp. 569-574.

13. Kunen, K., Signed Data Dependencies in Logic Programs, *Technical Report*, No. 719, Department of Computer Science, University of Wisconsin, to appear in *Journal of Logic Programming.*

14. Lloyd, J.W., Foundations of Logic Programming, Second Edition, Springer Verlag, 1987.

15. Mycroft, A., Logic Programs and Many-valued Logic, in: *Proc. of Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science 166, Springer-Verlag, Berlin, 1984, pp. 274-286.

16. Przymusinski, T., On the Semantics of Stratified Deductive Databases, in: *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, 1988.