Some formal systems of the logic programming

G.E. Mints
Institute of Cybernetics
Estonian Academy of Sciences
200108, Tallinn, USSR

1. We understand the logic programming in the broad sense of [T] as the nonprocedural programming in terms of logical specifications, so that the compilation of the resulting program (i.e. program synthesis) is performed (in essential part) by means of automatic proof search in suitable logical system. The most familiar (and for many people the only) example of logic programming is the Horn clause programming in the first order predicate calculus which forms the logical base of the PROLOG language. Unfortunately it is impossible to identify PROLOG with Horn clause programming in view of numerous nonlogical devices included in PROLOG to turn it into viable programming language. So the problem of exact formalization of PROLOG arises. We shall describe in section 2 such a formalization for the most prominent features of the control mechanism of PROLOG, viz. the ordering of the statements, failure mechanism, and cut predicate (/). A formalization of negation-as-failure similar to our (but simpler due to absence of other peculiar features of PROLOG) was independently found in [G].

In fact the propositional Horn clause logic is used in a number of program synthesis systems beyond PROLOG. The most developed of such system known to the author is PRIZ [MT]. In fact the logical base of PRIZ is also more powerful than propositional Horn clause logic, but in the other direction, than PROLOG: the planner (program synthesizer) of PRIZ is the complete procedure for the intuitionistic propositional calculus. The latter is known to be PSPACE complete, and one cannot expect good computational behaviour in the worst case. So a programming shortcut have been introduced to speed-up solution of some problems arising in practice. It turned out to be complete for the corresponding fragment of the modal logic $S_4$. Yet another shortcut proposed in [K] (but not implemented in PRIZ) turned out to be equivalent to the modal system $S_0.5$. 
Although there exists some logic interpretation of the negation-as-failure in Horn clause programming [Cl], it still has some undesirable features, and there were attempts to introduce more logical negation. In section 3 we shall show that the negation as inconsistency suggested in [G] leads to complete procedure for the suitable fragment of the so-called minimal predicate logic (cf. [Cu]). This logic is close to the intuitionistic one and coincides with it for the negative propositional queries. Judging from the description given in [G], the N-PROLOG introduced in [GR] is equivalent to the suitable fragment of the positive intuitionistic logic.

2. A system for a subset of Prolog including failure operation, ordering of clauses and control (/).

We present only main features. More complete description and proofs are in [M1],[M2].

Terms are constructed in a familiar way from individual variables with the help of functions. Atoms are expressions of the form \( P(t_1, \ldots, t_n) \), where \( P \) is \( n \)-ary predicate and \( t_1, \ldots, t_n \) are terms. Clauses are expressions of the form \( A \leftarrow B_1, \ldots, B_n \) and \( \neg B_1, \ldots, 
\) (abbreviated by \( B_1, \ldots, B_n \) where \( A, B_1, \ldots, B_n \) are atoms and \( n > 0 \). We describe first \(/\)-free case and then indicate necessary additions. The following is not sensitive to a kind of unification used, i.e. whether the most general unifier MGU(E,F) of two expressions E,F is found in conventional way or by matching used in PROLOG. The program works by transforming given list of goals. If goals \( A, X \) are to be proved, then the new list \((Z,X)s\) is formed, where \( A'<-Z \) is the clause to be used and \( s=MGU(A',A) \). The goals \( A, X \) will succeed with the substitution \( s \), which will be indicated by \( A,X+s \). A clause is suitable for the goal \( A \) if its head is unifiable with \( A \). It is understood that all clauses with the same head predicate are numbered consecutively and \( i:A<-Z \) means that \( A<-Z \) is the \( i \)-th clause in this enumeration.

Let us introduce derivable objects of our system together with interpretations.

\( Y+s : \) goal \( Y \) succeeds with the resulting substitution \( s \).

\( \neg Y : \) \( Y \) fails.

\( \neg j)Y : \) \( Y \) fails if the search is required to begin with \( j \)-th suitable rule

\( \neg i)X = \& (\neg j)X : \) \( X \) fails if search is required to begin with one of the first \((i-1)\) rules

Theorem 1. In the absence of / the goal \( X \) succeeds for a program \( P \) iff is derivable according to the following rules.