How to Handle Negation in Prolog: Several Logical Approaches

Can BAŞKENT

February 26, 2008

Abstract

Present paper surveys several formal and axiomatic methods to deal with the problem of negation in PROLOG. Furthermore, a brief philosophical account is also mentioned.

Contents

1 Prologue (for Prolog) .......................... 2
   1.1 Introduction and Motivation ............... 2
   1.2 The Problem .................................. 3

2 Four Valued Logic for Prolog .................. 3
   2.1 Negation in Four Valued Logic .......... 4
   2.2 Semantics .................................. 4
   2.3 Sketch of the Solution ..................... 6

3 Extending Programs with Classical Negation .... 9
   3.1 Classical Negation in Programs .......... 9

4 Extending the Models for Negation .......... 10
   4.1 Sketch of the Ideas ......................... 11
   4.2 Construction of the Model ................. 11
   4.3 Intuitionistic Logic for Prolog .......... 12

5 Some Philosophical Considerations .......... 13

6 Discussion and Conclusion ................. 14

References .................................. 15
1 Prologue (for Prolog)

In this paper, I will survey several logical and mathematical approaches to handle negation, specifically in the programming language Prolog.

Although Prolog belongs to the programming paradigm, called logical, logical paradigm has several pitfalls which do not exist in sound and complete logical systems. The most familiar and perhaps the most investigated drawback of Prolog is the handling of negation.

In spite of the fact that it is generally considered a knowledge representation tool, logic programming does not allow us to deal directly with incomplete information which might possibly arise in the context of knowledge representation. The reason for that is the fact that a classical consistent theory divides the set of sentences into three disjoint categories: provable, refutable and undecidable sentences. However, a logic program partitions the set of ground queries into two parts. First group is the queries which are answered yes, the second group is those which are answered no due to the closed world assumption [5]. The closed world assumption assumes that each ground atom which does not follow from the facts declared in the program is false. This mismatch between logical programming and the logic itself is the root of the problem.

The most common yet unsound implementation of negation is called negation as failure which can be formalized as follows.

\[
\text{not(Goal)} :- \text{Goal}, !, \text{fail}.
\]
\[
\text{not(Goal)}.
\]

The negation as failure is well discussed issue in the literature. Thus, we will refer the reader to [10] for further discussion of the issue. It is also immediate to observe that the semantics of negation as failure employs a non-standart operator \(!\). Thus, negation as failure requires an extended syntax. However, in order not to deviate from the present focus, we will not discuss the cut operator.

1.1 Introduction and Motivation

Since Frege, the notion of equality has always been a center of argumentation especially when the extension of meaning is in question [4].

However, once the notion of possible worlds were started be discussed in philosophical circles, prominently by S. Kripke, D. Lewis and R. Stalnaker, non-equality also gained significant importance.

Let me illustrate the ontological problem with a simple example. In our world, which is defined and restricted by Peano Axioms, \(2 \times 2 = 4\) is a validity. However, following the discussions of the aforementioned philosophers, one can think of a world in which \(2 \times 2 = 5\) where 5 is not 4. Therefore, the existence of negated sentences (in this case \(\neg(2 \times 2 = 4)\)) presents a challenging question: how can they exist?

\[\text{We have to underline the point that the numeral 5 and the numeral 4 denote the different numbers. Therefore, the problem is not a denotation problem but rather an ontological problem.}\]
In a similar fashion, in epistemology, the very same issue can be addressed from a knowledge perspective. How can we claim that we know $2 \times 2 = 4$? Do we need to know each instances $x$ where $2 \times 2 \neq x$. Well, I can recursively enumerate the integers $x$ such that $2 \times 2 \neq x$. The set of integers are easy to enumerate. What about then reals, complex or algebraic, transcendental numbers? So, how can I ensure that I enumerated all cases? In other words what does it mean to know $\varphi$? Do I have to know what I know? Do I have to know what I do not know?

Therefore, the closed world assumption presupposed a challenging position: an $S5$ model for knowledge.

Starting from these philosophical and logical motivations we will discuss the problem of negation in Prolog. In order to present a formal system to deal with negation in Prolog, several approaches have been suggested. We will start with the one put forward by Andrews in [1] where he extended the codomain of truth functions. Then, we will consider the approach presented by Gelfond & Lifschitz where they extended the logic programs so that they include the classical negation $\neg$ together with negation as failure $\text{not}$. Finally, I will briefly mention a rather technical and involved paper by Harland [6] in which he considers a model theoretic model for logic programming.

1.2 The Problem

The reason why the negation is problematic in Prolog is the fact that “it is not possible to express negative information with pure Horn clauses” as Prolog’s resolution works by utilizing Horn clauses [10]. In other words, it has been remarked that “a logic program is a set of definite Horn clauses” [11].

One of the immediate yet ad hoc solution to this problem, as we implied above, is to extend the formal system in such a way that the negated atoms can be allowed in the queries [10]. In other words, the formal language is then extended by allowing the negated propositional variables (i.e. $\neg p$). Obviously, the cost we have to pay is an increase in the complexity.

2 Four Valued Logic for Prolog

In formal sciences, one of the most used methods to deal with anomalies is to extend the logical system so that the newly obtained system will encompass the problematic point and thus will eliminate the aspect which caused the problem. Russell Paradox is maybe one of the most familiar example. We leave it to the reader to verify that Russell Paradox realizes our previous claim.

This was also the approach that Andrews followed in his paper [1]. We will first illustrate the problem and then interpret Andrews’ results.

\footnote{The very same issues were discussed by Hintikka in [7]. We can briefly mention it. To know what is known corresponds to $K\varphi \rightarrow KK\varphi$ axiom which characterizes the transitive frames. On the other hand, to know what is unknown corresponds to $\neg K\varphi \rightarrow KK\neg K\varphi$ axiom which characterizes the symmetric frames. Correspondence results can be found in [2].}
2.1 Negation in Four Valued Logic

What Andrews introduced was to consider a particular form of negation, and then to modify the logic in such a way that the considered form of negation works well in that restricted logic. He proceeded as follows.

He started by discussing a specific form of negation called “insist-on-ground negation as failure” (GNAF) [1]. The first distinguished aspect of that method is the fact that it computes a goal \( \neg \varphi \) only if \( \varphi \) is ground, and terminates immediately if \( \varphi \) is not ground\(^3\).

The approach that Andrews suggested was to utilize a four valued logic to deal with GNAF. The truth values of the extended logic were \( T \) for true, \( F \) for false, \( U \) for undefined and \( N \) for “floundering on negation”. Moreover, there defined a total order on the degree of computational priority of the given truth values. This total and linear order will be handy when the semantics of existential quantifier will be defined.

The imposed order is as follows: \( F \leq U \leq T \leq N \)

We also need a collection of auxiliary constants which will be denoted by \( S \). They will behave as “logical stand-ins” for unbound variables, so that they will provide a logical interpretation for groundness condition - which is the indispensable condition for this construction. The way these constants work can be summarized as follows. Had a query flounders, then these special constants let some ground instance of that query flounder, too\(^4\).

The formal language we will use is the first order logic with equality.

\[
(s = t) \mid p(t_1, \ldots, t_n) \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \exists x \varphi \mid \neg \varphi
\]

where \( s \) and \( t \) are used for terms whereas \( p \) is used for predicates. We will use \( x, y \) for metavariables which stand for the variables.

2.2 Semantics

We will now list the axioms given in [1].

1. \[
\frac{\theta: B, C, \alpha \Rightarrow \sigma}{\theta: B \land C, \alpha \Rightarrow \sigma}
\]

2. \[
\frac{\theta: B \land C, \alpha \Rightarrow \sigma}{\theta: B, \alpha \Rightarrow \sigma}
\]

3. \[
\frac{\theta': \alpha \Rightarrow \sigma}{\theta: s = t, \alpha \Rightarrow \sigma}
\]

4. \[
\frac{\theta: A(t_1, \ldots, t_n), \alpha \Rightarrow \sigma}{\theta: p(t_1, \ldots, t_n), \alpha \Rightarrow \sigma}
\]

where \( p(x_1, \ldots, x_n) \leftrightarrow \) is in complement(\( P \)).

5. \[
\frac{\theta: B \Rightarrow T, \alpha \Rightarrow \sigma}{\theta: \neg B, \alpha \Rightarrow F}
\]

where \( B \theta \) does not contain free variables of special constants.

6. \[
\frac{\theta: \neg B, \alpha \Rightarrow \sigma}{\theta: B, \alpha \Rightarrow N}
\]

where \( B \theta \) contains free variables or special constants.

\(^{\text{3It was remarked that GNAF has some other forms in which the call to } \neg \varphi \text{ is delayed until } \varphi \text{ is ground.}}\)

\(^{\text{4However, note that, it was remarked that the floundering can be avoided by requiring that the variables inside the negations be input-mode.}}\)
7. \[ \theta: x = T \]
8. \[ \frac{\theta: B[x := x'], \alpha \Rightarrow \sigma}{\theta: \exists x. B, \alpha \Rightarrow \sigma} \] where \( x' \) does not appear below the line.
9. \[ \frac{\theta: B, \alpha \Rightarrow F, \beta: C, \alpha \Rightarrow \sigma}{\theta: B \lor C, \alpha \Rightarrow \sigma} \]
10. \[ \theta: s = t, \alpha \Rightarrow F \] where \( s \) and \( t \) have no unifiers.
11. \[ \theta: p(t_1, \ldots, t_n), \alpha \Rightarrow U \] where \( p \) is not defined in complement(\( P \)).
12. \[ \frac{\theta: B, \alpha \Rightarrow \sigma}{\theta: \neg B, \alpha \Rightarrow \sigma} \] where \( B \theta \) does not contain free variables or special constants.
13. \[ \theta: B \Rightarrow \sigma \] where \( B \theta \) does not contain free variables of special constants, and \( \sigma \) is \( U \) or \( N \).

Note that \( T \) means that the computation returns at least one answer substitution; \( F \) means that the computation fails finitely; \( U \) means that the computation has floundered upon trying to call a non-existent predicate; and finally \( N \) means that the computation has floundered on negation on a non-ground subgoal [1].

Let us now elaborate more on these new truth values. As it is known extensively due to the Boolean algebraic characterization of the underlined logic, \( \varphi \land \psi \) is equivalent to \( \psi \land \varphi \). However, things do not go that smooth in applications due to the left-to-right reading in the operational semantics. Let me briefly explain the illustration given in [1]. Let \( v \) be the valuation function. Then, \( v(\varphi \land \psi) = T \) only when \( v(\varphi) = v(\psi) = T \). Let \( \varphi_1 \) be the term \( 2 = 2 \), \( \psi_1 \) be \( 2 = 3 \), \( \varphi_2 \) be \( 2 = 2 \lor \text{Loop}(3) \) and \( \psi_2 \) be \( 2 = 3 \) where \( \text{Loop}(x) \) is a predicate that goes into infinite loop for any input. Obviously, \( v(\varphi_1) = v(\varphi_2) = T \), and \( v(\psi_1) = v(\psi_2) = F \). The crucial observation hence is as follows. We have \( v(\varphi_1 \land \psi_1) = F \) whereas \( v(\varphi_2 \land \psi_2) = U \). Although the formulae \( \varphi_1 \) and \( \varphi_2 \) have the same truth value after Boolean algebra; they yield different truth values when they appear as the first conjunct of a conjunction. Thus, Andrews remarks, the truth function \( v \) is not compositional. Also, observe that in this case the distribution property of the underlined logic would not be of much help. We leave it to the reader to verify this claim.

Before going into the very involved account that Andrews suggests, let us investigate the problem a bit further. Consider now the following congruence
\[ \varphi \land (\psi \lor \chi) \equiv (\varphi \land \psi) \lor (\varphi \land \chi) \]
The following is the parse trees of both sides of the equivalence.\(^5\)

\(^5\)The very same tree can also be obtained by using game theoretical semantics for propositional logic. See [8] for the theoretical aspects of \( \forall \exists \) Ehrenfeucht - Fraïssé games. Therefore, although they are equivalent formulae, the games they correspond are different. From cognitive science and AI aspects, this is expected.
Thus, it is now easy to realize that equivalent formulae may be parsed by following a different order. Considering the left-to-right reading order, this comes as no surprise.

2.3 Sketch of the Solution

Let us now see the proposed solution of Andrews. However, first note that as his account for the problem is rather involved, we will briefly sketch his suggestion and refer the reader to the original paper for the minute details.

The solution will be constructed as follows. First, as Andrews put it, “a valuation on a subclass of goal formulas which characterizes their behavior with respect to the empty program $\emptyset$” will be given [1]. The reason for that is the fact that, the compositionality of the valuation function works only with the subset of the set of formulae. We start with the empty set $\emptyset$ as $\emptyset$ is a natural candidate for the base step for the recursive construction of the (extensions of) logical formulae. Second, a congruence relation will be given which will establish the relation between all the formulae and the designated subclass with respect to $\emptyset$. In other words, the base case of the bisimulation will be given. Third, and last, characterization of the congruence relation will be given for the rest of the formulae for a specific program $P$, so that the desired bisimulation will properly be constructed (in the meta level).

In the first stage, Andrews defined a subclass of formulae which he called $O$ formulae after “outer disjunction”. In order to define $O$ formulae, we first need another class which he called $N$ formulae after “negated disjunction”. The recursive definition of the aforementioned classes of formulae was given as follows.

$$N ::= (t_1 = t_2) \mid p(t_1, \ldots, t_n) \mid N \land N \mid \exists xN \mid \neg O$$

$$O ::= N \mid O \lor O$$

Therefore, $N$ formulae are the $O$ formulae whose top-level connective is not a disjunction. The $O$ formulae, on the other hand, are the goal formulae in which all interior disjunctions are the immediate subformulae of either negations of other disjunctions. The reason why we define the $O$ formulae like this will be clear once the semantics is given. Before proceeding into the inductive definition of the semantics, we will define a total order on the truth values for $O$ formulae which goes as follows.

$$F \leq_o U \leq_o T \leq_o N$$
Since we have an ordering, now we can define the maximum or minimum of the truth values. Before giving the semantics, recall that a formula is called ordinary if it does not contain any special constant. Now, the semantics of O formulae goes as follows.

1. \( v(t = t) = T \)
2. \( v(s = t) = F \) when \( s \) is not \( t \).
3. \( v(p(t_1, \ldots, t_n)) = U \)
4. \( v(\varphi \land \psi) = \begin{cases} v(\psi), & \text{if } v(\varphi) = T \\ v(\varphi), & \text{otherwise} \end{cases} \)
5. \( v(\varphi \lor \psi) = \begin{cases} v(\psi), & \text{if } v(\varphi) = F \\ v(\varphi), & \text{otherwise} \end{cases} \)
6. \( v(\exists x \varphi) = \max(\{\varphi(x = t) : t \text{ ground}\}) \)
7. \( v(\neg \varphi) = \begin{cases} F, & \text{if } \varphi \text{ is ordinary, and } v(\varphi) = T \\ U, & \text{if } \varphi \text{ is ordinary, and } v(\varphi) = U \\ T, & \text{if } \varphi \text{ is ordinary, and } v(\varphi) = F \\ N, & \text{otherwise} \end{cases} \)

Clearly, one can manipulate the ordering \( \leq_o \) in a different way and redefine the semantics of existential and negation formulae. Note that an existential formula is assigned to the truth value \( N \) if and only if at least one of its instances has the truth values \( N \).

For the logical considerations, note that it has been shown that the given semantics is sound and complete with respect to the valuation \( v \) whenever the goal is an \( O \) formula and the program is empty. For the inductive proof of the aforementioned propositions, the interested reader is referred to the original paper [1].

As the second step, now we define a congruence relation \( \equiv \) as follows.

- \( (\varphi \lor \psi) \land \chi \equiv (\varphi \land \chi) \lor (\psi \land \chi) \)
- \( \varphi \land (\psi \lor \chi) \equiv (\varphi \land \psi) \lor (\varphi \land \chi) \) when \( \varphi \) is an \( N \) formula.
- \( \exists x(\varphi \lor \psi) \equiv (\exists x \varphi) \lor (\exists x \psi) \)

The main theorem of the congruence \( \equiv \) is given as follows.

**Theorem 2.1.** \( \equiv \) preserves operational outcome on goal stacks.

**Proof.** We will very briefly sketch the proof. The proof goes on the total number of unfolding in the given formula. For the complex formulae whose unfolding number is greater than or equal to 2, the axioms given in the previous section will be used to unravel the formula so that the induction step can be properly applied. For the technical details refer to [1].
Therefore, the unfolding operation and algorithm of a given $O$ formula always terminate. The reason for that is the fact that we are in the first order logic and hence the formulae can only have countably many conjuncts or disjuncts. Moreover, for the existential formulae, we already set a total order, hence even without referring to the axiom of choice, we can pick the proper value for existential formula. So, for each form of formula, this procedure terminates.

Now, we can extend the discussion to any goal formulae. The extended valuation function $v'$ was defined as follows.

$$
v'(\varphi) = \begin{cases} v(\varphi), & \text{if } \varphi \text{ is an } O \text{ formula} \\ v'(\varphi') & \text{if } \varphi \equiv P \varphi' \end{cases}
$$

One can also observe that $v'$ precisely characterizes the outcome of general goals with respect to $\emptyset$ [1].

Previously, we extended the truth function. Now, we will extend the congruence $\cong$ in order to associate it with a program $P$. The fold/unfold congruence $\cong_P$ associated with a program $P$ is the least congruence relation such that for every predicate definition $p(t_1, \ldots, t_n) = \varphi$, we have

$$p(t_1, \ldots, t_n) \equiv_P \varphi(x_1 := t_1, \ldots, x_n := t_n)$$

So, $\cong_P$ is the extension of $\cong$ from the empty program to an arbitrary program $P$. Therefore, we have a similar preservation result, but this time for program $P$. The result goes as follows.

**Theorem 2.2.** $\cong_P$ preserves operational outcome with respect to $P$.

**Proof.** The proof is similar to the proof of Theorem 2.1, and hence skipped. □

The one of the original constructions introduced in the original paper was to define a truth valuation $v_P$ associated with a program $P$ in such a way that $v_P$ can stand for the denotation of the program $P$. Since now we have a program $P$, not the empty program $\emptyset$, we do not want the truth value $U$ appear so often. Thus, by introducing yet another ordering $\leq_k$, we suppress $U$. Hence, for the truth values $T,F,U$ and $N$, we define $\leq_k$ as follows. $U \leq_k F, U \leq_k N, U \leq_k T$. Now, we will explicitly suppress $U$ in the truth valuation function. So, we will have a new truth function $v_P$ associated with $P$. We define it as follows.

$$v_P(\varphi) = \max_k(\{v'(\varphi') : \varphi \equiv_P \varphi'\})$$

where $\max_k(S)$ is the maximum over the ordering $\leq_k$ of the truth values in the set $S$. 6

Henceforth, $v_P$ precisely characterizes the outcome of all ground goals with respect to any program $P$. The proof can be found in [1].

\[\text{Observe that } v_P \text{ function is not total recursive. Because, the search for a defined truth value (i.e. } \not\equiv U \text{) might be an infinite search.} \]
3 Extending Programs with Classical Negation

Yet another method to deal with the problems that negation as failure present, is to extend the logic programs in such a way that they will include the classical negation, denoted by $\neg$. Therefore the programs will include negative information explicitly. As a result, there will be two types of queries which do not succeed. One type does not succeed since it fails and the other does not succeed since its negation succeeds [5]. Note that, notation-wise, we will put $\neg$ for classical negation and $\text{not}$ for negation as failure.

The extended logic program, therefore, will have the following scheme for a rule:

$$L_0 \leftarrow L_1, \ldots, L_m, \text{not} L_{m+1}, \ldots, \text{not} L_n \quad (1)$$

where $n \geq m \geq 0$ and each $L_i$ for $0 \leq i \leq n$ is a literal. We define literal as a formula of the form $A$ or $\neg A$ for an atom $A$. Recall that $\neg A$ means the negation of $A$ in the classical logical sense, not in the sense of negation as failure.

Therefore, for a ground query $A$, the program will return $\text{yes}$ if the answer set contains $A$; and will return $\text{no}$ if the answer set contains $\neg A$ or $\text{unknown}$ if the answer set contains neither $A$ nor $\neg A$. In other words, the $\text{no}$ answer will explicitly mean the presence of negative information in the program [5]. It is thus easy to observe that extended programs will not correspond to the positive Horn clauses. Furthermore, we have to be careful when defining the answer set. The answer set will be defined as the set of all ground literals which can be generated by using either the rules of the program or by utilizing classical logic. Then, what happens if the set of ground literals has both $A$ and $\neg A$? Recall that for inconsistent set $X$ we have, $X \vdash \varphi$ for any formula $\varphi$. So, we conclude that if it were the case, the answer set will be the whole set of literals. Thus, we derive that every contradictory program has exactly one answer set - which is the set of all literals.

3.1 Classical Negation in Programs

The distinction between $\neg \varphi$ and $\text{not} \varphi$ is essential when the closed world assumption is not applicable to $\varphi$ for some reason. One can come up with numerous practical examples, so we refer the interested reader to the original paper for precise examples [5].

As the usual (i.e. general) programs are specific cases of extended programs, the immediate question that may be raised is the following: “What is the relation between extended programs and general programs in terms of answer sets?”. Let us now investigate it a bit further.

Let $P$ be an extended program. For each predicate $p$ occurring in $P$, let $p'$ be the positive form of the negative literal $\neg p$. We will denote the positive form of a literal $L$ by $L^+$. For any set of literals $S$, then, $S^+$ is the set of positive forms of the members of $S$. Moreover, the program $P^+$ stands for the general program which is obtained from $P$ by replacing each rule of the form given in Equation 1
by the following rule

\[ L_0^+ \leftarrow L_1^+, \ldots, L_m^+, notL_{m+1}^+, \ldots, notL_n^+ \]  

(2)

In this setting it was observed that, a consistent answer set \( S \) is the answer set of the program \( P \) if and only if \( S^+ \) is an answer set for the program \( P^+ \). The proof of this proposition can be found in the original paper [5].

The important result, that the above statement yields is the following fact.

**Proposition 3.1.** For a consistent set \( S \) of literals, if \( S^+ \) is the only answer set of the program \( P^+ \), then \( S \) is the only answer set of the program \( P \).

In other words, under some conditions (such as stability), one can reduce the extended programs to the general programs in such a way that \( not \) is well-handled.

It is easy to see that the last proposition would not hold if we drop the consistency assumption. Let \( P \) be the following program.

\[
\begin{align*}
\alpha &\leftarrow not\neg\alpha \\
\beta &\leftarrow \alpha \\
\neg\beta &\leftarrow \alpha
\end{align*}
\]

This program has no answer set, and the answer set of \( P^+ \) is the positive form of the inconsistent set \( \{\alpha, \beta, \neg\beta\} \).

4 Extending the Models for Negation

The last approach we will consider is the modal logical approach that Harland suggested. It is loosely based on the possible world semantics of Kripke.

Our Kripke frame can be thought of an inverted cone with the empty program \( \emptyset \) as the root. The frame will then be reflexive and transitive as the each extension \( v \) of a state \( w \), will be accessible from \( w \). The accessibility relation, thus, can be defined as follows.

\[ P \cup \{A\} \text{ is reachable from } P \text{ iff } A \text{ is not completely defined in } P. \]

Therefore the aim is to form a structure in which some states are accessible whereas some are not. In order to illustrate this, let us reproduce the example given in the original paper. Assume that the program \( \text{append} \) is completely defined in the program \( P \). Consider the larger program \( Q = P \cup \{\text{append}([1, 2], [3, 5], [9, 10])\} \). So, we want \( Q \) not to be accessible from \( P \) [6].

As the mathematical tools which were used in the original paper are rather involved, we will sketch the idea and refer the reader to the original paper for the technical and logical details. However, we will conclude our discussion of Hartland’s paper by a short note on intuitionistic logic.
4.1 Sketch of the Ideas

Recall that the interpretation function which is familiar from model theory is a monotonic function mapping definite formulae to sets of closed atoms. In other words, interpretation function explicitly identifies which atoms are true at which point. Then, what about undetermined atoms (such as loops)?

Therefore, the original paper, at this stage adopts a three-valued logic with truth values \textit{true}, \textit{false}, and \textit{undefined}. Recall that this logic has an intuitionistic sense as it does not adopt the law of excluded middle. In other words, there are formulae which are neither true nor false.

Hence, it seems reasonable to extend the interpretation function to cover the \textit{false} cases. Therefore, the new interpretation function \( I \) will be a 2-valued function where \( \text{left}(I) \) is for \textit{true} values, and \( \text{right}(I) \) is for \textit{false} values. Since we do not have the law of excluded middle, there might be some states which are not returned by \( I \) either as \textit{true} or \textit{false}. Those states are clearly the \textit{undefined} states.

The truth of the interpretation is the stage where we have to be careful. The semantics is as follows for the interpretation function \( I \), and a program \( P \).

\[
\begin{align*}
I, P \models A & \iff A \in \text{left}(I) \\
I, P \models \neg A & \iff A \in \text{right}(I) \\
I, P \models \varphi \land \psi & \iff I, P \models \varphi \text{ and } I, P \models \psi \\
I, P \models \varphi \lor \psi & \iff I, P \models \varphi \text{ or } I, P \models \psi \\
I, P \models \exists x \varphi & \iff I, P \models \varphi[x/t] \text{ for some } t \\
I, \langle D, N \rangle \models (D' \rightarrow \varphi) & \iff I, \langle D \land D', N \rangle \models G \text{ and } \langle D \land D', N \rangle \geq \langle D, N \rangle \\
& \text{(i.e. } \langle D \land D', N \rangle \text{ is reachable from } \langle D, N \rangle) \\
\end{align*}
\]

where the syntax for \( D \) and \( \varphi \) is given as follows

\[
\begin{align*}
D & ::= A \mid \forall x D \mid D \land D \mid \varphi \rightarrow A \\
\varphi & ::= A \mid \exists x \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid D \rightarrow \varphi
\end{align*}
\]

In this setting, \( \varphi \) represents the goals which are not allowed to contain universal quantifiers. There are several reasons for that. Let us now briefly mention that.

First is the Halting Problem. We may not be able to consider the each atoms in the space. Second, we do not adopt closed world assumption. Thus, we will be forced to check whether each atom satisfies the desired properties.

The proof theoretical notion \( \vdash \) can easily be defined in a similar fashion. We will not go into the details but refer the reader to the original paper [6].

4.2 Construction of the Model

One of the very important theorems in model theory is Lindenbaum’s Lemma [2]. Lindenbaum’s lemma states that, every consistent set can be extended to a maximally consistent set. The important aspect of this lemma is the fact that the proof gives a constructive method to build a maximally consistent set. Let us briefly reproduce the famous proof here.

Assume that \( S \) is the given consistent set. Enumerate all the formulae in the language as \( \varphi_0, \varphi_1, \ldots \). Define inductively,
Thus, $\Sigma$ is the maximally consistent extension of $S$. We leave it to the reader to verify the facts that $\Sigma$ is an extension of $S$, and $\Sigma$ is both maximal and consistent.

The method for constructing a model given in [6] is essentially the method used in building the maximally consistent set step by step. Therefore, there will be a level, where a formula fails to be true and a formula succeeds to be true. Then, considering all formulae which are true and false, we will end up with a large model.

Obviously, whilst constructing aforementioned model, one should be careful about the computational restrictions that Prolog poses. One such restriction is the non-computability of point infinity. How can we introduce the point infinity to the Prolog? The answer is to define it as a constant which does not have any predecessor. This relates to the issue of handling the infinite loops. Therefore, the constant which stands for the point infinity and the truth value undefined can be identified.

To conclude, the model we obtain for Prolog is constructed from the root by adding up the states by maintaining consistency which is essentially the same idea used in the proof of Lindenbaum’s Lemma.

What about then implementing these ideas from Lindenbaum’s Lemma in a real programming language. However, the important aspect of the constructive proof of the lemma is the denumerability of the formulae of the language. Therefore, once the enumeration is set, the construction of the maximally consistent sets is easy. Although the logics should have countable number of variables and hence countable number of formulae and strings, it does not seem feasible and efficient to employ the actual Lindenbaum’s Lemma in a real programming paradigm. The underlying reason for that is the computational cost.

### 4.3 Intuitionistic Logic for Prolog

There exists a close correlation between the closed world assumption and intuitionistic logic. The problems related to the closed world assumption can be interpreted from an intuitionistic logical point of view. Recall that in intuitionistic logic $\varphi \lor \neg \varphi$ is not a theorem as Brouwer put it. Therefore, if a query does not succeed, it does not necessarily entail that the query will fail. Hence, without the closed world assumption, one can suggest that the logic of prolog is rather intuitionistic, and we force it to behave classically by dictating closed world assumption. However, from the perspective of programming languages, this seems inevitable considering the vast majority of practical implementations.
5 Some Philosophical Considerations

The handling of negation in “negation as failure approach” requires a massive assumption, namely “the closed world assumption”. However, in most AI cases, this assumption is far from being satisfied. The underlying reason for that, in scientific and even in practical cases, is a variance of the problem of induction.

The modern discussions of the negation can be traced back to Hume [9]. He put in his masterwork which he published in the exact same age of the present author:

In every system of morality, which I have hitherto met with, I have always remark’d, that the author proceeds for some time in the ordinary ways of reasoning, and establishes the being of a God, or makes observations concerning human affairs; when all of a sudden I am surpriz’d to find, that instead of the usual copulations of propositions, is, and is not, I meet with no proposition that is not connected with an ought, or an ought not. This change is imperceptible; but is however, of the last consequence. For as this ought, or ought not, expresses some new relation or affirmation, ’tis necessary that it shou’d be observ’d and explain’d; and at the same time that a reason should be given; for what seems altogether inconceivable, how this new relation can be a deduction from others, which are entirely different from it. [9]

The so-called **ought-is** problem is usually illustrated by the following famous example.

> Until today, I have observed that the sun raised from west. Therefore, I will observe that the sun will raise from west tomorrow morning, too.

The reflexivity of the knowledge (i.e., $K\varphi \rightarrow \varphi$) is an indispensable property of knowledge models. Therefore, if one wants to approach to Hume with this property as a major tool, the immediate conclusion she will draw will be the fact that “sun raises from west” is not a knowledge. Because, we cannot verify that it will hold in each temporal instance of the cosmos. Therefore, in a sophisticated sense, we cannot know neither “sun raises from west” nor “sun does not raise from west”.

In a similar fashion, Duhem’s paradox is a modern version of the very same problem. Although Duhem’s problem is about the philosophy and methodology of natural science, the exact same intuition can be used to reason about the negation as failure. He put

> ... the experiment does not tell us where the error lies. [3, p. 187]

He also added:

> The mathematical symbols used in theory have meaning only under the very definite conditions. (...) Theory is forbidden to make use of
Hence, it is not difficult to observe that the handling of negation is not a current problem, but a problem which kept philosophers and logicians busy throughout the ages. As our primer focus in the present paper is not philosophical aspects of negation, we will not investigate the issue further.\footnote{However, one can wonder, how can Prolog can handle the change of facts in the declaration of the program. For instance, how one can model if the fact \texttt{:- mortal(Socrates)} changes to \texttt{:- immortal(Socrates)} or to \texttt{:- mortal(Aristotle)}. This issue has both logical and philosophical significance, too. Goodman’s Paradox and model updates are the main research topics in this subject.}

\section{Discussion and Conclusion}

In this paper, we considered three approaches to the negation problem in logic programming. The reason we chose these approaches was their logical strength. In other words, the mathematical tools employed in these approaches represent the general approaches to deal with anomalies in formal languages. It is, thus our unfounded intuition that, most other approaches to the negation problem would be reduced one or another instance of either of these approaches.

The immediate observation was the fact that all of these methods had some roots in the formal treatment of (modal) logics. Extending the modal language and signature, constructing a new (modal) model from the old one are the familiar methods in logic.

Another observation is the fact that, the problem of negation is not peculiar only to Prolog. Therefore, the results mentioned in the present paper can easily be adopted to most other logical programming languages which adopts closed world assumption.

However, the research in this very subject can still be carried out. The several approaches to do research in this topic might include:

\begin{itemize}
    \item Restricted closed world assumption: Restrict the closed world assumption to a particular set of propositions such as atomic ones.
    \item Modal negation: This approach can best be described by the motto: “negated somewhere else”. In other words, we need to separate set of syntactical domains. One for positive formulae, one for the negative formulae. Essentially, they will be symmetric. But, we will not need a closed world assumption as we will have another domain for negative and negated propositions and sentences.
    \item Prolog search tree algorithms: In order to spot where the program needs the closed world assumption some graph theoretical and discrete mathematical notions can be employed in such a way that an bisimular graph will be constructed where the lazy evaluation of the failed proposition will
\end{itemize}
be considered last. Therefore, according to the main connective of the searched formula, the lazy evaluation may yield the formula (for instance if the main connective is a disjunction, it is a bit more easy to observe)

- Game Theoretical Approach: One can come up with a zero-sum game in which the program will lose if it needs to use the closed world assumption, and will win if it does not require the closed world assumption. In order to be precise, one should show that this game is zero-sum (or at least $f(n)$-sum for some monotonic continuous function $f$, and natural number $n$.)

To conclude, we would like to underline that the negation problem is not an easy one. The mismatch between the formal logic and the programming paradigm is significant. Therefore, as we presented several of them, one should refer to nonstandard logics, even to paraconsistent ones to come up with a precise solution for the problem.

References


[4] FREGE, G; “ber Sinn und Bedeutung”.


[7] HINTIKKA, J.; “Knowledge and Belief”.


