Adding Negation-as-Failure to Intuitionistic Logic Programming

Anthony J. Bonner
L. Thorne McCarty

Rutgers University
Department of Computer Science
New Brunswick, NJ 08903
bonner@paul.rutgers.edu
mccarty@cs.rutgers.edu

Abstract

Intuitionistic logic programming is an extension of Horn-clause logic programming in which implications may appear "embedded" on the right-hand side of a rule. Thus, rules of the form $A(x) \leftarrow [B(x) \leftarrow C(x)]$ are allowed. These rules are called *embedded implications*. In this paper, we develop a language in which negation-as-failure is combined with embedded implications in a principled way. Although this combination has been studied by other researchers, Gabbay has argued in [10] that the entire idea is logically incoherent since modus ponens would not be valid in such a system. We show how to solve this problem by drawing a distinction between *rules* and *goals*. To specify the semantics of rules and goals, we then develop an analogue of Przymusinski's perfect model semantics for stratified Horn-clause logic [20]. Several modifications are necessary to adapt this idea from classical logic to intuitionistic logic, but we eventually show how to define a *preferred model* of a *stratified intuitionistic rulebase*, and this enables us to specify the semantics of such a rulebase by reference to its preferred models. Finally, we prove a soundness and completeness theorem. Throughout the paper, we discuss various examples of the use of intuitionistic embedded implications plus negation-as-failure, to demonstrate the utility of the language.

This paper appears in *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 681-703. MIT press. Conference held in Austin, TX, Oct 29–Nov 1 1990.

This paper is available at the following URL:
`ftp://db.toronto.edu/pub/bonner/papers/hypotheticals/naclp90.ps.gz`

1 Introduction

Intuitionistic logic programming is an extension of Horn-clause logic programming in which implications may appear "embedded" on the right-hand side of a rule. Thus, rules of the form $A(x) \leftarrow [B(x) \leftarrow C(x)]$ are allowed. Several researchers have investigated the properties of these "embedded implications" [11, 10, 14, 15, 17, 4], and have shown them to be useful for hypothetical reasoning [2], for legal reasoning [16], for modular logic programming [17], and for lexical scoping [18].

In a series of prior papers [4, 3, 6], Bonner, et al., have established various theoretical results on the complexity and expressibility of intuitionistic embedded implications in the function-free (Datalog) case. For the full language, in which universal quantifiers may also appear "embedded" on the right-hand side of a rule, the query problem is undecidable [6], but if embedded universal quantifiers are barred from the language, the query problem is decidable and its data-complexity is PSPACE-complete [4]. If negation-as-failure is added to the language, these complexity results can be extended to expressibility results. Thus, as shown in [6], the unrestricted language with embedded universal quantifiers plus negation-as-failure can express any typed, generic, computable database query in the sense defined by Chandra and Harel in [7, 8]. If the language includes negation-as-failure, but no embedded universal quantifiers, then it can express any typed, generic database query in PSPACE [4]. Additional syntactic restrictions yield additional complexity classes. If the rules are at most linear recursive, then the language can express any database query in NP, and if linear recursion alternates with negation-as-failure in a stratified manner, then the language can express the database queries at each level in the polynomial time hierarchy above P [3]. It is interesting that these complexity classes can all be generated by a single language with a natural sequence of syntactic restrictions.

Note the role played by negation-as-failure in these results. Without negation-as-failure, we can state complexity results, but we cannot state expressibility results since we cannot represent non-monotonic queries. Also, for many applications, negation-as-failure turns out to be a very useful addition to the language of embedded implications, just as it has been shown to be a very useful addition to the language of definite Horn clauses. Several examples are discussed in Sections 2 and 3.5 below. Thus a theory of negation-as-failure in the context of intuitionistic logic programming seems essential. In our prior papers [4, 3, 6], the treatment of negation-as-failure was entirely proof-theoretic. In this paper, to round out the picture, we develop a semantic theory and prove a soundness and completeness theorem.

There are reasons to think that the addition of negation-as-failure to intuitionistic logic programming will lead to serious semantic difficulties. Gabbay studied the problem in [10], and concluded that the entire idea was logically incoherent, since modus ponens would no longer be valid. We address this problem in Section 3.2. More recently, Harland [12] has proposed a semantics for negation-as-failure within intuitionistic logic programming, based on the fixed-point construction of Miller [17]. Harland's approach seems to overcome some of Gabbay's objections. However, Harland's semantics requires that we label predicates as "completely defined" or "incompletely defined", and this requirement precludes the use of the very rules that are needed for our expressibility results. This point is discussed further in Section 3.5. Our approach, instead, is based on Przymusinski's *perfect model semantics* for stratified Horn-clause logic augmented by negation-as-failure [20]. Although this approach requires that we restrict our attention to *stratified intuitionistic rulebases*, a term defined in Section 3.3, this restriction turns out to be acceptable for our intended applications.

In the case of classical logic programming, Przymusinski's idea was to define a *preference relation* on the classical models of a rulebase, and then to identify the *minimal* elements in the preference ordering as the intended models of the rules [20]. In the case of intuitionistic logic programming, the natural generalization would be to define a preference relation on the intuitionistic models of the rulebase, which are usually taken to be Kripke models [13]. However, it turns out that we need to make two modifications of Przymusinski's original idea in order to generalize it to intuitionistic logic. First, we need to generalize the notion of a Kripke structure, which is monotonic, to the notion of a *non-monotonic structure*, which is not. Second, we need to modify the definition of a preference relation. Przymusinski's preference relation minimizes atomic goals stratum by stratum, and this automatically maximizes negative goals stratum by stratum, since the logic is classical. For intuitionistic logic programming, we need to consider a broader class of goals, and within this broader class we need to minimize positive goals and maximize negative goals simultaneously, since intuitionistic logic lacks the law of excluded middle. These points are developed in Sections 4.2 through 4.4. Once these definitions have been modified, however, we can follow the perfect model analogy quite closely by using the preferred models of a rulebase to specify its intended meaning. In fact, as in the classical perfect model semantics, we show that every stratified intuitionistic rulebase has, up to equivalence, a single preferred model.

The development in this paper is for a language without function symbols. This is the kind of language typically used in database applications, and it is exactly the language needed for the expressibility results in [4, 3, 6]. Although it is function-free, we allow the language to have an *infinite* set of constant symbols. It should therefore be straightforward to generalize the development given here to include function symbols, since in a logic with function symbols but without equality (e.g., Prolog), each Herbrand term can be treated as a distinct constant symbol. The intuitionistic logic programming language discussed in [14, 15] treats function symbols in precisely this way.

2 Examples

In this section, we present two examples of the use of intuitionistic embedded implications combined with negation-as-failure. The reader will note the appearance of two implication symbols in these examples: $\leftarrow$ and $\Leftarrow$. The distinction between these two symbols will be explained in Section 3.2, but for the moment it should be ignored. More examples will be provided in section 3.5, after inference with negation-as-failure has been formally defined.

Example 1. Assume a database of atomic formulas $TAKE(s, c)$ asserting that student s has taken course c, and a rulebase that states the graduation requirements for a university. Thus $GRAD(s)$ means that student s is eligible to graduate. We can then define a predicate $NEAR\_GRAD(s)$, meaning that student s is within one course of satisfying the graduation requirements, as follows:

$$NEAR\_GRAD(s) \leftarrow COURSE(c),\ [GRAD(s) \Leftarrow TAKE(s, c)].$$

Intuitively, this rule states that student s is nearly a graduate if there exists some course c such that, if the student took this course, then he could graduate. In [2], it is shown that there is a precise sense in which such rules cannot be expressed in classical logic, even in full classical logic. However, the embedded implication in this rule has a natural interpretation in intuitionistic logic. It means simply: Add the formula $TAKE(s, c)$ to the database and then try to prove $GRAD(s)$.

Now imagine a set of rules defining a student's eligibility for financial aid, as follows:

$$STIPEND(s) \leftarrow ADMITTED(s),\ NEAR\_GRAD(s),\ \sim GRAD(s).$$
$$FELLOWSHIP(s) \leftarrow ADMITTED(s),\ \sim NEAR\_GRAD(s),\ \sim GRAD(s)$$

Intuitively, a student s who has been admitted to the university is eligible for a stipend if he is a near-graduate but not yet a graduate. On the other hand, if he is neither a near-graduate nor a graduate, then he is eligible for a fellowship. $\Box$

In this example, the symbol '$\sim$' denotes negation-as-failure. Thus, in applying the rule for $FELLOWSHIP(s)$, we would ask if there exists some course c such that, if $TAKE(s,c)$ were assumed to be true then $GRAD(s)$ would also be true. We would conclude that s is eligible for a fellowship only if this hypothetical test fails. Conversely, we would conclude that s is eligible for a stipend only if this hypothetical test succeeds. Note that $NEAR\_GRAD(s)$ is vacuously true if $GRAD(s)$ is true (as long as there exists a course somewhere in the database!). But we do not want to give a stipend to a student who has already graduated, so we also include the test $\sim GRAD(s)$ in the rule for $STIPEND(s)$. This means that a stipend is available to those students who need to take *exactly* one course in order to satisfy the graduation requirements. This observation becomes important in the following example.

Example 2. In [19], Pareschi studies the use of intuitionistic embedded implications in a definite clause version of Categorical Grammar. Classical Categorical Grammar (CG) encodes linguistic information by assigning syntactic types to lexical items. A translation of CG into first-order logic represents these type assignments by definite clauses in which lexical items appear inside a special predicate $CONN$. For example, the Horn rule below encodes the English rule for 'put'. This rule, along with appropriate rules for other lexical items, allows the derivation of the sentence: "Mary put a book on the table."

$$S(x,w) \leftarrow NP(x,y),\ CONN(\text{put},y,y+1),\ NP(y+1,z),\ PP(z,w).$$

In this rule, the variables x, y, z and w represent positions in a string of words. The rule states that a sentence occurs between positions x and w if a noun phrase ("Mary") occurs between x and y, the word 'put' connects positions y and $y+1$, another noun phrase ("the book") occurs between $y+1$ and z, and a prepositional phrase ("on the table") occurs between z and w.

Although this particular example can be handled adequately with Horn-clause logic, intuitionistic embedded implications are needed to represent the type assignments for relative pronouns. For example, the rule for 'which' might be written as follows:

$$\begin{aligned} REL(v-1,w) \quad \leftarrow \quad & CONN(\text{which},v-1,v),\ v \leq y,\ y \leq w, \\ & [S(v,w) \Leftarrow NP(y,y)]. \end{aligned}$$

Here, the relative clause is assumed to have a noun phrase of null length at some position y representing the "extraction site" for the relative pronoun. Intuitively, the rule states that if the word 'which' connects string positions

$v-1$ and v, and if there exists some position y between v and w such that, if a noun phrase were inserted at y then there would be a sentence between v and w, then it follows that there is a relative clause between string positions $v-1$ and w. These rules thus allow the derivation of the following relative clauses:

which Mary put a book on $\upharpoonleft$,
which Mary put $\upharpoonleft$ on the table,
which $\upharpoonleft$ put a book on the table,

where the symbol $\upharpoonleft$ marks the extraction site. But there is a problem. These rules also allow the derivation of the following string as a relative clause:

which Mary put a book on the table,

since the embedded implication $[S(v,w) \Leftarrow NP(y,y)]$ is vacuously true whenever $S(v,w)$ is true. Pareschi considers several possible solutions to this problem, including a modification of the intuitionistic proof theory and a filtering of certain kinds of proofs [19]. But there is an obvious solution using negation-as-failure. If we simply redefine the rule for relative pronouns as follows:

$$\begin{aligned} REL(v-1,w) \quad \leftarrow \quad & CONN(\text{which}, v-1, v),\ v \leq y,\ y \leq w, \\ & [S(v,w) \Leftarrow NP(y,y)], \\ & \sim S(v,w). \end{aligned}$$

then the ungrammatical derivation is blocked. □

These examples show the possible uses of intuitionistic embedded implications plus negation-as-failure in very simple cases. More complex examples are discussed in Section 3.5 below. However, in order to analyze these more complex examples, it is necessary to formulate the inference system of our language with greater precision. This is the subject of the following section.

3 Inference

A number of researchers have developed inference systems for intuitionisitic logic programming [11, 15, 17, 4]. However, most of this work has focused on the negation-free case. As pointed out by Gabbay in [10], negation-as-failure introduces subtle problems for intuitionistic logic.

In this section, we first review the negation-free case and discuss the problems of introducing negation. We argue that the problems pointed out by Gabbay can be resolved if one distinguishes two types of logical implication, one for rules and one for goals. Using this idea, we develop an inference system for intuitionistic logic programming plus negation-as-failure.

3.1 The Negation-Free Case: Review

This section defines a logical inference system for intuitionistic logic programming. Such systems have been developed by several researchers, and the one presented here is a simplified version which retains many of the essential properties of the more elaborate systems while admitting a clean theoretical analysis. It is an extension of function-free Horn logic (i.e., Datalog), both syntactically and proof theoretically. In [4], it is shown that the data complexity of this system is complete for polynomial space.

The syntax of the logic is first order. It includes three (possibly infinite) sets: a universe of constant symbols $a, b, c, \ldots$, a set of variables $x, y, z, \ldots$, and a set of predicate symbols $A, B, C, \ldots$. Note that the logic, as defined here, does not include function symbols.

As in Horn logic, a rule has the form $A \leftarrow \phi$, where ϕ is a conjunction of goals. However, because our inference system is hypothetical, we have a larger class of goals. In general, if ϕ is a legal goal (i.e., a query), then so is $\phi \leftarrow A_1, \ldots, A_n$, where each A_i is atomic. These considerations are summarized by the following rules, which define the syntax of our language:

$$\begin{array}{lcl} \mathit{rule} & := & \mathit{atom} \mid \mathit{atom} \leftarrow \mathit{goal}_1 \ldots \mathit{goal}_m \\ \mathit{goal} & := & \mathit{atom} \mid \mathit{goal} \leftarrow \mathit{atom}_1 \ldots \mathit{atom}_m \end{array}$$

where the terminal symbol *atom* refers to atomic formulas. Goals thus include the formulas B and $B \leftarrow C, D$ and $(B \leftarrow C) \leftarrow D$. When interpreted either classically or intuitionistically, a goal is equivalent to a definite Horn clause. Rules include the formulas $A \leftarrow B$ and $A \leftarrow (B \leftarrow C, D)$ and $A \leftarrow [(B \leftarrow C) \leftarrow D]$. Such rules are called *embedded implications* [14, 15]. A rulebase is defined to be a finite set of rules, and a database is defined to be a finite set of ground atomic formulas.

Definition 3.1 (Inference) *Suppose R is a rulebase. Then we define the following inference system, where DB is an arbitrary database,*

Axioms: $R + DB \vdash A$ *is an axiom for every atomic formula* $A \in DB$.

Inference Rules:

1. *for any rule $A \leftarrow \phi_1 \ldots \phi_m$ in R, and for any ground substitution θ,*

$$\frac{R + DB \vdash \phi_i\theta \quad \text{for each } i}{R + DB \vdash A\theta}$$

2. *for any goal ϕ,*

$$\frac{R + DB + \{B_1, \ldots, B_m\} \vdash \phi}{R + DB \vdash \phi \leftarrow B_1 \ldots B_m}$$

In this inference system, each inference rule has the following interpretation: If the formulas above the the horizontal line can be inferred, then those below the horizontal line can also be inferred. The following elementary example shows how inference can be performed in a "top-down" manner by inverting these rules.

Example 3. Suppose that R consists of the following three rules:

$$A \leftarrow (B \leftarrow D).$$
$$B \leftarrow C.$$
$$C \leftarrow D.$$

Then $R \vdash A$. This can be proved by a straightforward, top-down argument:

$$\begin{array}{ll} & R \vdash A \\ \textit{if} & R \vdash B \leftarrow D \\ \textit{if} & R + D \vdash B \\ \textit{if} & R + D \vdash C \\ \textit{if} & R + D \vdash D \end{array}$$

But the last line is trivially true since it is an axiom. □

It is implicit in Definition 3.1 that the variables of a rule are universally quantified. Furthermore, as in Horn logic, if a variable does not appear in the head of a rule, then the universal quantifier can be moved inside the rule body and converted to an existential. For example, in the embedded implication $A(x) \leftarrow [B(x,y) \Leftarrow C(x,y)]$, the variable y does not appear in the head. If this rule appears in a rulebase R, then the inference system above implies that for any constant symbol x,

$$R + DB \ \vdash \ A(x) \quad \textit{if} \quad R + DB \ \vdash \ B(x,y) \Leftarrow C(x,y) \quad \textit{for some } y.$$

Consequently, the rule can be read in two equivalent ways:

$$\forall_x \forall_y \ A(x) \leftarrow [B(x,y) \Leftarrow C(x,y)]$$
$$\forall_x \ A(x) \leftarrow \exists_y \ [B(x,y) \Leftarrow C(x,y)]$$

The latter interpretation enables our logic to represent the notion of being "within one course of graduation," as defined in Example 1.

3.2 Negation-As-Failure: Paradoxes

We now extend the syntax of our rules to allow negated goals. Thus, rules of the form $A \leftarrow \sim B$ are allowed. Operationally, the expression $\sim B$ is interpreted as the failure to prove B; that is, $R \vdash \sim B$ iff $R \not\vdash B$. As in Horn logic, difficult problems arise if rulebases with negation are not stratified. Section 3.3 develops the notion of stratification for intuitionistic rulebases. Even when they are stratified, however, embedded implications display interesting paradoxes, as originally pointed out by Gabbay [10]. This section examines some of these paradoxes and proposes a syntactic solution. Section 4 proposes a semantic solution.

If implicational queries are allowed, then paradoxes arise even for stratified Horn rulebases. The following example, for instance, shows that for such rulebases, implication is not transitive.

Example 4. Suppose that R consists of the following two rules:

$$A \leftarrow B, \sim C.$$
$$B \leftarrow C.$$

Then $R \vdash A \leftarrow B$ and $R \vdash B \leftarrow C$, but $R \not\vdash A \leftarrow C$. Each of these points is straightforward. For instance, to infer $A \leftarrow C$, we would have to add C to the rulebase and then infer A; that is, we would have to derive the expression $R + C \vdash A$. However, once C is in the rulebase, the first rule is permanently blocked, so there is no way to derive A. Thus, $R + C \not\vdash A$. Hence $R \not\vdash A \leftarrow C$. □

The next example shows that stratified embedded-implications violate a very basic property of logical systems: modus ponens.

Example 5. Suppose that R consists of the following two rules:

$$B \leftarrow \sim C.$$
$$D \leftarrow (B \leftarrow C).$$

Then $R \vdash D \leftarrow B$ and $R \vdash B$, but $R \not\vdash D$. Each of these points is straightforward. For instance, to infer D, we would have to infer $B \leftarrow C$. We would thus have to add C to the rulebase and infer B; that is, we would have to derive the expression $R+C \vdash B$. Once C is in the rulebase, however, the first rule is blocked, so there is no way to derive B. Thus $R+C \not\vdash B$ so $R \not\vdash B \leftarrow C$. Hence $R \not\vdash D$. □

Examples 4 and 5 show that stratified rulebases lack some basic properties that we normally associate with logical implication. It might seem, therefore, that embedded implications augmented with negation-as-failure cannot be treated logically. After all, if implication is not transitive, and if modus ponens is not valid, then what makes the system logical?

A solution to the problem is suggested by our distinction between rules and goals. Syntactically, rules and goals are implications, but these implications are treated differently by the inference system of Definition 3.1: Inference rule 1 deals with rules, and inference rule 2 deals with goals. Since they are treated differently, it should not be too surprising that they have different properties. The real surprise is that this difference does not make itself apparent until the system is augmented with negation-as-failure. It is the different treatment of rules and goals that is the source of the apparent problem with implication.

The problem can be resolved by a simple syntactic device: We distinguish between the two types of implication by using two types of implication symbols. Rules will continue to have the form $A \leftarrow B$, but goals will have the form $A \Leftarrow B$. Rules are transitive and can be used for modus ponens, whereas goals cannot. Proof theoretically, the expression $R \vdash A \Leftarrow B$ means simply that $R + B \vdash A$. Semantically, as shown in Section 4, goals and rules also have two distinct interpretations.

With this is mind, we now define the syntax of our language with negation-as-failure. As in Horn logic, we distinguish between positive and negative goals. If ϕ is a positive goal (i.e., it contains no negation signs), then $\sim \phi$ is a negative goal. Furthermore, because the inference system is hypothetical, if ϕ is a legal goal, then so is $\phi \Leftarrow A_1, ..., A_n,$ where each A_i is atomic. These considerations are summarized by the following rules, which define the syntax of our language:

$$\begin{array}{lcl}
rule & := & atom \leftarrow goal_1 ... goal_m \\
goal & := & pgoal \mid ngoal \\
pgoal & := & atom \mid pgoal \Leftarrow atom_1 ... atom_m \\
ngoal & := & \sim pgoal \mid ngoal \Leftarrow atom_1 ... atom_m
\end{array}$$

Negative goals include the formulas $\sim B$ and $\sim (B \Leftarrow C)$ and $(\sim B) \Leftarrow C$. If interpreted classically, a negative goal is equivalent to a negative Horn clause. Rules include the formulas $A \leftarrow \sim B$ and $A \leftarrow \sim (B \Leftarrow C)$ and $A \leftarrow (\sim B \Leftarrow C)$.

3.3 Stratified Inference

Aside from the paradoxes mentioned above, inference involving negation-as-failure is not always well-defined. For instance, given the two rules $A \leftarrow \sim B$ and $B \leftarrow \sim A$, it is not clear whether A should be inferred, or B, or both, or neither. As in Horn logic, however, it is not difficult to provide an operational semantics for negation-as-failure as long as there is no recursion through negation. In particular, a rulebase can be stratified, or layered, as described in [1]. This section develops the notion of stratification for intuitionistic rulebases.

Given a goal $P(x) \Leftarrow Q(x)$, we call the predicate symbol P the *head predicate* of the goal. Head predicates are central to our notion of stratification. They are defined recursively in the following definition.

Definition 3.2 *The predicate symbol in an atomic formula is its head predicate. The head predicate of a formula ϕ is also the head predicate of the formulas $\sim \phi$, $\phi \Leftarrow \psi$, and $\phi \leftarrow \psi$.*

Definition 3.3 (Stratification) *Let R be a rulebase. R is stratified iff it is the disjoint union of rulebases $R_0, ..., R_k$, where for each rule in R_i,*

- *If the rule has a negative premise whose head predicate is P, then every rule with head predicate P is contained in $\cup_{j<i} R_j$*
- *If the rule has a positive premise whose head predicate is P, then every rule with head predicate P is contained in $\cup_{j\leq i} R_j$*

In this case $R_0, ..., R_k$ is called a stratification of R, and R_i is its i_{th} stratum.

This definition guarantees that if recursion occurs, then it occurs only within a single stratum and that it never occurs through negation. In the rest of this section, we consider only stratified rulebases, and we always assume that we have a particular stratification in mind.

Notice that a predicate may be defined by many rules in many strata. We say that a predicate *belongs* to the highest stratum in which it is defined.

Only head predicates play a role in the definition of stratification. Thus, in the rule $A \leftarrow (B \Leftarrow C)$, the rules defining C can appear in any stratum. In particular, they can appear above the strata in which A and B are defined. Thus $\leftarrow$ affects the possible stratifications of a rulebase, but $\Leftarrow$ does not. Intuitively, in the formula $A \leftarrow B$, the atom B is "executed" in order to prove A, while in the formula $A \Leftarrow B$, the atom B is "assumed" to be true in order to prove A. This is why, in the latter case, B is not considered for the stratification; i.e., because B is not "executed", the formula $A \Leftarrow B$ cannot lead to recursion through negation.

We now define the operational semantics of stratified rulebases. The idea is to derive expressions of the form $R \vdash \phi$ stratum by stratum, using the derivations of one stratum as the starting point for the derivations of the stratum above.

Given a stratified rulebase R, we define a set of inference systems, one for each stratum. These systems generalize the inference system of Definition

3.1 in two ways. First, each inference system corresponds to a particular stratum of R, where the j^{th} inference system uses only the rules in the j^{th} stratum. Second, each inference system is provided with an unspecified set of axioms $\mathcal{A}$. Each inference system thus defines a mapping cl_j that takes a set of axioms $\mathcal{A}$ as input, and returns a set of inferred expressions $cl_j(\mathcal{A})$ as output. These mappings encapsulate inference as a single operation, thus providing a convenient separation between inference, which occurs within strata, and negation-as-failure, which occurs between strata.

The operational semantics of R are defined stratum-by-stratum in terms of these mappings: Given the output from cl_j, we apply the closed world assumption to it, and use the resulting expressions as the input to cl_{j+1}. In this way, the derivations of one stratum are the starting point for the derivations of the stratum above. The rest of this section makes these ideas precise.

Definition 3.4 *Suppose R is a stratified rulebase. An inference expression for R is an expression of the form $R + DB \vdash \phi$ where DB is a database and ϕ is a goal. The inference expression is positive iff ϕ is positive, and negative iff ϕ is negative.*

Definition 3.5 (Inference) *Let R be a stratified rulebase, and let $\mathcal{A}$ be a set of inference expressions for R. For each stratum of R, we define a distinct inference system, in which $\mathcal{A}$ is the set of axioms. The inference system associated with the j^{th} stratum is defined as follows, where DB is an arbitrary database:*

Axioms:

1. *The expression $R+DB \vdash A$ is an axiom for every atomic formula $A \in DB$ such that A belongs to the j^{th} stratum.*
2. *Each expression in $\mathcal{A}$ is an axiom.*

Inference Rules:

1. *for any rule $A \leftarrow \phi_1 ... \phi_m$ in the j^{th} stratum of R, and for any ground substitution θ,*

$$\frac{R + DB \vdash \phi_i\theta \quad \text{for each } i}{R + DB \vdash A\theta}$$

2. *for any goal ϕ,*

$$\frac{R + DB + \{B_1, ..., B_m\} \vdash \phi}{R + DB \vdash \phi \Leftarrow B_1 ... B_m}$$

$cl_j(\mathcal{A})$ denotes the set of inference expressions derivable in this system.

Like all Gentzen-style inference systems, this system is monotonic in the set of axioms. It is also idempotent and inflationary. We thus have the following basic results.

Lemma 3.1 *cl_j is a function which maps sets of inference expressions for R into sets of inference expressions for R. Furthermore, this function has the following properties:*

- *Monotonicity: If $\mathcal{A} \subseteq \mathcal{B}$ then $cl_j(\mathcal{A}) \subseteq cl_j(\mathcal{B})$.*
- *Idempotence: $cl_j(\mathcal{A}) = cl_j(cl_j(\mathcal{A}))$*
- *Inflationaryness: $\mathcal{A} \subseteq cl_j(\mathcal{A})$*

Given a set of inference expressions $\mathcal{A}$, we define a new set $\overline{\mathcal{A}}$. Informally, $\overline{\mathcal{A}}$ is the result of applying the *closed world assumption* to $\mathcal{A}$; i.e., if an expression is not true, then assume it is false. This is the basis of negation-as-failure.

Definition 3.6 *Let $\mathcal{A}$ be a set of inference expressions for R. Then $\overline{\mathcal{A}}$ is another set of inference expressions for R, where for any database DB and any positive goal ϕ,*

- $R + DB \vdash \phi \in \overline{\mathcal{A}}$ *iff* $R + DB \vdash \phi \in \mathcal{A}$
- $R + DB \vdash \sim \phi \in \overline{\mathcal{A}}$ *iff* $R + DB \vdash \phi \notin \mathcal{A}$

We have now assembled the main components of our operational semantics of stratified rulebases: (i) Definition 3.6 captures the idea of negation-as-failure, which occurs between strata, and (ii) Definition 3.5 captures the idea of inference, which occurs within strata. It remains to combine these ideas into a unified semantics of stratified rulebases. The following definition does exactly this. By alternating inference with negation-as-failure, it constructs a set of inference expressions stratum-by-stratum in a bottom-up fashion.

Definition 3.7

- $\mathcal{A}^0 \; = \; \{\}$
- $\mathcal{A}^{j+1} \; = \; \overline{cl_{j+1}(\mathcal{A}^j)}$ *for $j \geq 0$.*

Intuitively, $\mathcal{A}^j$ is the set of inference expressions derived by the first j strata of the rulebase. Using $\mathcal{A}^j$ as a starting point, $\mathcal{A}^{j+1}$ is generated by applying the rules in the $j + 1^{st}$ stratum until saturation, after which the closed world assumption is applied. Because a rulebase has a finite number of strata, this process eventually terminates. If a rulebase R has only k strata, then for $j \geq k$, the closure operator cl_{j+1} does not add any positive inference expressions to $\mathcal{A}^j$. However, the second inference rule of Definition 3.5 is still applicable to negative inference expressions. For instance, if the expression $R + B \vdash \sim A$ is in $\mathcal{A}^k$, then the expression $R \vdash (\sim A) \Leftarrow B$ is in $\mathcal{A}^{k+1}$. For this reason, $\mathcal{A}^k$ is a proper subset of $\mathcal{A}^{k+1}$. Note, however, that $\mathcal{A}^{k+j} = \mathcal{A}^{k+1}$ for $j \geq 1$. This prompts the following definition.

Definition 3.8 *Let R be a stratified rulebase with k strata. Then, for any database DB and any goal ϕ,*

$$R + DB \vdash_j \phi \quad \text{iff} \quad \text{the expression } R + DB \vdash \phi \text{ is in } \mathcal{A}^j.$$
$$R + DB \vdash \phi \quad \text{iff} \quad R + DB \vdash_{k+1} \phi.$$

This completes the definition of the operational semantics of stratified embedded-implications.

3.4 A Special Case

In the case of stratified Horn rules, the semantics developed above reduce to the semantics of Apt, Blair and Walker [1] and of Przymusinski [20]. It is worth noting, however, that these semantics differ from the semantics of Prolog in a special case: for rules of the form $A \leftarrow \sim B(x)$. Because of the possibility of floundering, Prolog gives such rules a special interpretation. This section discusses this interpretation and argues that by introducing intermediate predicates, our semantics can be forced to interpret rules as Prolog does, and vice-versa.

As in the negation-free case (Section 3.1), our operational semantics imply that the variables of a rule are universally quantified. Furthermore, if a variable does not appear in the head of a rule, then the universal quantifier can be moved inside the rule body and converted to an existential. For example, in the rule $A \leftarrow B(y), \sim C(y)$, the variable y does not appear in the head. This rule can thus be read in two equivalent ways:

$$\forall_y \quad A \leftarrow B(y), \sim C(y)$$
$$A \leftarrow \exists_y \, [B(y), \sim C(y)]$$

Note that Prolog interprets this rule in the same two ways. In this respect, our semantics treats Horn rules with negation in the same way that classical logic programming does.

However, our semantics, like those of [1] and [20], differ from the semantics of Prolog in one respect: In Prolog, rules such as $A \leftarrow \sim C(y)$ are given a special interpretation. In such rules, the value of the variable y is unguarded, that is, it is not constrained by a positive literal, such as the literal $B(y)$ in the previous example. Unguardedness can cause the Prolog interpreter to flounder. For this reason, Prolog does *not* interpret this rule in either of the following ways:

$$\begin{aligned} \forall_y \ A &\leftarrow \sim C(y) \\ A &\leftarrow \exists_y \sim C(y) \end{aligned} \tag{1}$$

Instead, Prolog interprets it this way:

$$A \leftarrow \sim \exists_y \ C(y) \tag{2}$$

That is, the existential quantifier is brought inside the negation sign.

This attempt by Prolog to avoid floundering is only partially successful however. Both in Prolog and in our semantics, we can effectively obtain either interpretation (1) or interpretation (2) by introducing intermediate predicates. For instance, by introducing a unary predicate $E(y)$, the following two rules effectively give us interpretation (1):

$$A \leftarrow E(y) \qquad\qquad E(y) \leftarrow \sim C(y)$$

This is true both in Prolog and in our semantics. Furthermore, in both systems, these rules cause floundering during top-down inference. Likewise, by introducing a zero-ary predicate D, the following two rules effectively give us interpretation (2):

$$A \leftarrow \sim D \qquad\qquad D \leftarrow C(y)$$

This is true both in Prolog and in our semantics. Furthermore, in both systems, these rules do not cause floundering during top-down inference. Prolog's special interpretation of unguarded rules is therefore a convenience that does not provide any additional expressive power and does not eliminate floundering.

Conceptually, then, the operational semantics described in Section 3.3 provides a uniform interpretation of stratified rulebases. In the special case of stratified Horn rulebases, this semantics is equivalent to others in the literature [1, 20]. In the very special case of unguarded Horn rules, this semantics can be translated into the semantics of Prolog in a straightforward way.

3.5 Examples

This section shows that stratified rulebases of embedded implications can solve some familiar problems. Note that in both the examples given, the predicate $SOMELEFT$ acts as an "intermediate predicate" as described in Section 3.4.

Example 6. Suppose R is the following stratified rulebase:

$$EVEN \leftarrow SELECT(x),\ [ODD \Leftarrow MARK(x)].$$
$$ODD \leftarrow SELECT(x),\ [EVEN \Leftarrow MARK(x)].$$
$$EVEN \leftarrow \sim SOMELEFT.$$
$$SOMELEFT \leftarrow SELECT(x).$$
$$SELECT(x) \leftarrow D(x),\ \sim MARK(x).$$

Then $R, DB \vdash EVEN$ iff DB contains an even number of entries of the form $D(x)$. □

In this example, the rulebase determines the parity of a relation D.[1] This rulebase is best understood from the top-down perspective. During inference, the first two rules select elements one-by-one from the relation D. Each time an element x is selected, the formula $MARK(x)$ is hypothetically added to the database. In this way, a record is kept of which elements have been selected, so that they are not selected again. (Note that the last rule selects only unmarked elements.) As elements are selected and marked, the first two rules "flip back and forth" between the two queries $EVEN$ and ODD. At each point in this computation, the predicate $EVEN$ is true iff the set of unmarked elements has even parity. The third and fifth rules terminate the recursion, stating that $EVEN$ is true when there are no unmarked elements left.

Example 7. Suppose that DB is a database representing a directed graph. That is, $NODE(a) \in DB$ iff a is a node in the graph, and $EDGE(a,b) \in DB$ iff there is an edge in the graph from a to b. Suppose also that R is the following stratified rulebase:

$$YES \leftarrow SELECT(x),\ [PATH(x) \Leftarrow MARK(x)].$$
$$PATH(x) \leftarrow SELECT(y),\ EDGE(x,y),\ [PATH(y) \Leftarrow MARK(y)].$$
$$PATH(x) \leftarrow \sim SOMELEFT.$$
$$SOMELEFT \leftarrow SELECT(x).$$
$$SELECT(x) \leftarrow NODE(x),\ \sim MARK(x).$$

Then $R, DB \vdash YES$ iff the graph represented by DB has a directed Hamiltonian path. □

[1] It is worthwhile noting that this problem cannot be solved in Datalog (function-free Horn logic) even when stratified negation is allowed [8].

The rulebase in this example is best understood from the top-down perspective. During inference, the rulebase tries to construct a Hamiltonian path one node at a time. The first rule selects a node x at which the path is to begin. The second rule is then applied repeatedly, selecting a node y connected by an edge to the last node in the path. Each time a node is selected, it is marked. In this way, no node is selected twice. The third and fourth rules say that a Hamiltonian path has been found when every node has been marked, that is, when every node has been visited exactly once. Note that node selection is non-deterministic, so in effect, the rulebase searches the graph for all possible Hamiltonian paths.

It is the ability to record facts, such as which nodes are marked, that distinguishes this logic from (function free) Horn logic and accounts for its computational complexity [4, 3].

In [12], Harland develops a semantics for intuitionistic embedded implications augmented with negation-as-failure. The two examples above highlight the differences between Harland's treatment and our own. Harland's approach does not depend on stratification, and assigns a meaning even to the rule: $A \leftarrow \sim A$. On the other hand, Harland's semantics cannot handle the rulebases above. To see this, we need to examine the operational semantics in [12] more closely. Harland draws a distinction between "completely defined" predicates, such as $APPEND$, and "incompletely defined" predicates, such as $CARCINOGEN$. Negation-as-failure is available only for completely defined predicates, while goals must have incompletely defined predicates in their premises. But look at the definition of $EVEN$ and ODD in Example 6 above. The predicate $MARK$ appears inside the negation-as-failure symbol in the fifth rule, and yet it also appears in the premise of a goal in the first two rules. This dual role for the predicate $MARK$ is crucial. A similar observation applies to the rulebase that computes Hamiltonian paths in Example 7, and to the Turing machine encodings that are needed for the expressibility results in [4, 3]. It does not appear that these rulebases would be handled correctly by Harland's semantics. Note, however, that the rules in Examples 1 and 2 of Section 2 cause no such problems. This suggests that it may be useful to develop distinct semantic theories for distinct species of negation.

4 Model-Theoretic Semantics

Section 3 provided a proof-theoretic development of intuitionistic logic programming with negation-as-failure. This section provides a model-theoretic development.

In classical Horn logic programming, the semantics is truly classical only in the negation-free case. When negation is introduced, new semantic devices are needed to account for the non-monotonicity of the resulting system. A common approach, due to Przymusinski, is to introduce a preference relation on the classical models of the rulebase, and then to focus on the most preferred, or *perfect*, models [20]. It is now well known that stratified Horn rulebases have a unique perfect model [20].

Similarly, in intuitionistic logic programming, the semantics is truly intuitionistic only in the negation-free case. When negation is introduced, a natural approach would be to introduce a preference relation on intuitionistic models to account for the non-monotonicity. When we attempt to do this, however, two difficulties emerge. First, a preference relation by itself does not fully account for non-monotonicity. Second, because intuitionistic logic is not a two-valued logic (i.e., it lacks the law of the excluded middle), the preference relation developed in [20] is not strong enough. The rest of this section addresses these two difficulties and proposes novel solutions.

This first section below provides a brief development of intuitionistic semantics. The following sections then show how this semantics can be modified to account for negation-as-failure.

4.1 Intuitionistic Semantics

The following is a simplified development of the Kripke semantics of intuitionistic logic. A complete development may be found in [9, 13].

Definition 4.1 (Structures) *An intuitionistic structure is a triple* $M = \langle S, \leq, \pi \rangle$*, where*

- S *is a non-empty set,*
- $\leq$ *is a transitive, reflexive relation on* S*,*
- π *is a mapping from elements of* S *to sets of ground atomic formulas,*
- *for any two elements* s_1 *and* s_2 *in* S*, if* $s_1 \leq s_2$ *then* $\pi(s_1) \subseteq \pi(s_2)$*.*

The elements of S *are called the substates of* M*.*

In a complete development of intuitionistic semantics, each substate may have its own distinct domain of constant symbols. We have assumed here, however, that the domain of each substate is equal to the universe of all constant symbols. In [6], we show that for embedded implications, these are the only kind of structures that one needs to consider. We continue to use these simplified structures in this paper, since they simplify the theoretical development.

Truth in an intuitionistic structure M is defined relative to its substates. Thus, one can ask whether a formula ψ is true at a particular substate s of M, written $s, M \models \psi$. The following definition makes this idea precise.

Definition 4.2 (Satisfaction) *Suppose M is an intuitionistic structure and s is a substate of M. Then,*

$$
\begin{array}{lll}
s, M \models A & \textit{iff} & A \in \pi(s) \quad \textit{when } A \textit{ is atomic.} \\
s, M \models \psi_1 \wedge \psi_2 & \textit{iff} & s, M \models \psi_1 \quad \textit{and} \quad s, M \models \psi_2 \\
s, M \models \forall_x \psi(x) & \textit{iff} & r, M \models \psi(b) \quad \textit{for all } r \geq s \\
 & & \qquad\qquad \textit{and all constant symbols } b. \\
s, M \models \psi_2 \leftarrow \psi_1 & \textit{iff} & r, M \models \psi_1 \quad \textit{implies} \quad r, M \models \psi_2 \quad \textit{for all } r \geq s
\end{array}
$$

Note that unlike classical logic, intuitionistic implication is not defined in terms of disjunction and negation. Rather, it has an independent semantic definition. An intuitive interpretation of this semantics may be found in [13] and [9].

Definition 4.3 (Models) *$M \models \psi$ iff $s, M \models \psi$ for all substates s of M. If $M \models \psi$, then M is a model of ψ.*

Definition 4.4 (Validity) *A formula ψ is valid iff $M \models \psi$ for all intuitionistic structures M.*

Definition 4.5 (Entailment) *Suppose ψ_1 and ψ_2 are formulas. Then $\psi_1 \models \psi_2$ iff the formula $\psi_2 \leftarrow \psi_1$ is valid.*

The following theorem is a central result of [6]:

Theorem 4.1 (Soundness and Completeness) *Let R be a negation-free rulebase and ϕ be a goal. Then $R \vdash \phi$ iff $R \models \phi$.*

4.2 Non-Monotonic Structures

In the tradition of logic programming, we would like to identify a single model that characterizes all the inferences sanctioned by a rulebase R. That is, we would like to find a single intuitionistic structure M_R that is a model of R and for which $M_R \models \phi$ iff $R \vdash \phi$ for all goals ϕ. We call such a structure a *canonical model* of R. In [6], it is shown that negation-free rulebases have a canonical intuitionistic model.

When intuitionistic logic is augmented with negation-as-failure, it is not possible to construct a canonical model. Let us forget for the moment that we have introduced two types of implication in the proof theory of stratified rulebases. Suppose that R is the rulebase consisting of the single rule $A \leftarrow B, \sim C$. Then $R \vdash A \leftarrow B$ but $R \not\vdash A \leftarrow B, C$. For any intuitionistic structure M, however, if $M \models A \leftarrow B$ then $M \models A \leftarrow B, C$. In general, therefore, a stratified rulebase does not have a canonical, intuitionistic model.

The problem is that intuitionistic structures are "biased" towards monotonicity. This is due to the fourth item in Definition 4.1. This could be called the *monotonicity condition*: Intuitively, it requires that substates get larger monotonically as they get higher in the structure. The simplest way of eliminating the monotonic "bias" in intuitionistic structures is to eliminate this condition. We say that the resulting structures are non-monotonic.

Definition 4.6 (Structures) *A non-monotonic structure is a triple* $M = \langle S, \leq, \pi \rangle$, *where*

- S *is a non-empty set,*
- $\leq$ *is a transitive, reflexive relation on* S,
- π *is a mapping from elements of* S *to sets of ground atomic formulas.*

The elements of S *are called the substates of* M.

Note that intuitionistic structures are a special case of non-monotonic structures. For brevity, we often refer to non-monotonic structures simply as *structures*. We shall show that every stratified rulebase has a canonical, non-monotonic structure. First, however, we must define what it means for a non-monotonic structure to be a model of a stratified rulebase.

4.3 Non-Monotonic Satisfaction

In Section 3, we argued that stratified rulebases really demand two types of implication: Implications of the form $A \leftarrow B$ to denote rules, and implications of the form $A \Leftarrow B$ to denote goals. The inference systems in Definitions 3.1 and 3.5 treat these two forms of implication differently. Inference rule 1 deals with rules, and inference rule 2 deals with goals. It is an interesting coincidence that in the negation-free case, rules and goals have the same semantic interpretation, i.e., they both correspond to intuitionistic implication. The situation with stratified rulebases is more complex. We have already shown that rules and goals have different proof-theoretic properties. They must, therefore, have different semantic definitions.

This section defines a notion of satisfaction for non-monotonic structures. There are a number of equivalent ways to do this. The formulation below is chosen because of its similarity to classical logic. In particular, except for $\Leftarrow$, every logical connective has a classical interpretation. The connective $\Leftarrow$ can thus be viewed as a modal operator that causes inference to shift from one classical world to another. For instance, to determine whether $s, M \models A \Leftarrow B$, we move to the closest worlds above s in which B is true, and then we ask whether A is true in these worlds. The idea of "closest", or minimal, worlds captures the non-monotonic nature of the connective $\Leftarrow$. The semantics of the two connectives $\leftharpoondown$ and $\Leftarrow$ are thus quite different.

Definition 4.7 (Satisfaction) *Suppose that $M = \langle S, \leq, \pi \rangle$ is a non-monotonic structure and that s is a substate of M. Then*

$$
\begin{array}{lll}
s, M \models A & \textit{iff} & A \in \pi(s), \text{ when } A \text{ is atomic.} \\
s, M \models \psi_1 \wedge \psi 2 & \textit{iff} & s, M \models \psi_1 \text{ and } s, M \models \psi_2. \\
s, M \models \sim \psi & \textit{iff} & s, M \not\models \psi \\
s, M \models \forall_x \psi(x) & \textit{iff} & s, M \models \psi(b) \text{ for all constant symbols } b. \\
s, M \models \psi_2 \leftharpoondown \psi_1 & \textit{iff} & s, M \models \psi_1 \text{ implies } s, M \models \psi_2. \\
s, M \models \psi_2 \Leftarrow \psi_1 & \textit{iff} & r, M \models \psi_2 \text{ for all minimal substates } r \geq s \\
 & & \qquad\qquad \text{such that } r, M \models \Box \psi_1. \\
s, M \models \Box \psi & \textit{iff} & r, M \models \psi \text{ for all substates } r \geq s.
\end{array}
$$

In this definition, we have introduced a convenient box notation. Intuitively, anything that is "boxed" represents a monotonic truth: If $\Box\phi$ is true at a substate s, then it remains true as we climb to higher substates. This corresponds to the proof-theoretic notion of adding a formula to a rulebase: If B is added to a rulebase R, then B remains true no matter how many other formulas we add to R. In this way, the semantic notion of minimal substates corresponds nicely to the proof-theoretic notion of hypothetical insertion.

Definition 4.8 (Models) *Let M be a non-monotonic structure. Then*

- *If ϕ is a goal, then $M \models \phi$ iff $s, M \models \phi$ for all minimal substates s of M.*
- *If ψ is a rule, then $M \models \psi$ iff $s, M \models \psi$ for all substates s of M.*

If $M \models \psi$, then we say that M is a *model* of ψ. M is a model of a rulebase R iff it is a model of every rule in R.

Because of its emphasis on minimal substates, this semantics differs from intuitionistic semantics in several important respects. First, it is not monotonic. That is, if $M \models \phi$ then it is not necessarily the case that $M \models \phi \Leftarrow \psi$.

This is exactly the kind of behavior that we want to capture. Second, a non-monotonic structure may be inconsistent: If M has no minimal substates, then $M \models (\sim \phi \wedge \phi)$ for all goals ϕ. Such inconsistencies will exist if, for instance, M consists of an infinite descending chain of substates with no minimal element. Even if M has minimal substates, a subtler inconsistency may exist. For instance, if M has no substates such that $M \models \Box\psi$, then $M \models (\sim \phi \wedge \phi) \Leftarrow \psi$ for all formulas ϕ. These considerations prompt the following definition.

Definition 4.9 (Inconsistency) *A structure M is inconsistent iff $M \models \phi \Leftarrow \psi$ and $M \models (\sim \phi) \Leftarrow \psi$ for some goal $\phi \Leftarrow \psi$. M is consistent iff it is not inconsistent.*

The following lemma is a basic result about consistency.

Lemma 4.2 *A structure M is consistent iff for every database DB, there is a minimal substate s of M for which the set $\{r \geq s \mid r, M \models \Box DB\}$ has a minimal element.*

4.4 Preferred Models

Section 4.2 defined a generalization of intuitionistic structures, called non-monotonic structures. Section 4.3 then defined what it means for such a structure to be a model of a stratified rulebase. This section defines a preference relation on structures. Development of a preference relation is the final step in the semantics of stratified Horn rulebases, and it is also the final step in the semantics of stratified, intuitionistic rulebases.

In the perfect model semantics of stratified Horn rulebases, the preference relation minimizes the number of true atomic facts in a disciplined way. First, those atoms defined in the lowest stratum of the rulebase are minimized; then those defined in the second-lowest stratum are minimized; then those in the third-lowest stratum, etc. This process is a generalized version of the closed world assumption.

In Horn logic programming, it is natural to minimize atomic facts, since these are the only inferences that one makes. In intuitionistic logic programming, however, one infers a more general class of formulas, including goals of the form $A \Leftarrow B$. It is therefore natural to generalize the notion of minimizing atoms to that of minimizing positive goals.

It turns out, however, that minimizing positive goals is not enough. In classical logic, minimizing the set of true atoms automatically maximizes the number of false atoms. This is because, in classical structures, an atom is either true or false. However, in non-monotonic structures (as in intuitionistic structures) an atom may be neither true nor false: It may be true in some minimal substates of the structure, and false in others. For this reason, our preference relation must do two things simultaneously: It must minimize positive goals and maximize negative goals. Furthermore, it must do this stratum by stratum, from the bottom up. These ideas are formalized by the following definitions.

Definition 4.10 (Projection) *Suppose G is a set of goals. Then $G|_j$ is the set of goals in G whose head predicates belong to the j^{th} stratum. In addition, $G|_j^+$ and $G|_j^-$ denote the positive and negative goals in $G|_j$, resp.*

Using this notation, we define a preference relation on sets of goals. Within each stratum, this relation prefers the set that has the fewest positive goals and the most negative goals. Furthermore, the comparison is lexicographic in that two sets of goals are compared at the j^{th} stratum only if they are identical at all lower strata. In this way, lower strata are given higher priority.

Definition 4.11 (Preference) *Let G and H be two sets of goals. Then $G \preceq H$ iff for each stratum j,*

$$\begin{array}{ll} \textit{if} & G|_i = H|_i \;\; \textit{for all} \;\; 1 \leq i < j, \\ \textit{then} & G|_j^+ \subseteq H|_j^+ \;\; \textit{and} \;\; H|_j^- \subseteq G|_j^-. \end{array}$$

Note that $\preceq$ is transitive and reflexive and that $G = H$ iff $G \preceq H$ and $H \preceq G$. This preference relation on sets of goals induces a preference relation on structures, as follows.

Definition 4.12 *If M is a structure, then $goals(M)$ denotes the set of goals satisfied by M. That is, $goals(M) = \{\phi \mid M \models \phi\}$, where each ϕ is a goal.*

Definition 4.13 *Let M_1 and M_2 be two structures. Then, $M_1 \preceq M_2$ iff $goals(M_1) \preceq goals(M_2)$. In addition, $M_1 \simeq M_2$ iff $M_1 \preceq M_2$ and $M_2 \preceq M_1$.*

Note that $\preceq$ is a transitive, reflexive relation on structures and that $\simeq$ is an equivalence relation. Thus, two structures are equivalent iff they satisfy exactly the same goals. If $M_1 \preceq M_2$ then we say that M_1 is *preferable* to M_2. As in the perfect model semantics of stratified Horn rulebases, we focus on the minimal elements of this relation.

Definition 4.14 (Preferred Models) *Let R be a stratified rulebase, and let $\mathcal{M}_R$ be the set of consistent models of R. A preferred model of R is a minimal element of $\mathcal{M}_R$, that is, minimal with respect to the preference relation $\preceq$.*

The following definition completes our generalization of the classical, perfect-model semantics.

Definition 4.15 (Entailment) *Let R be a stratified rulebase and ϕ be a goal. Then $R \models \phi$ iff $M \models \phi$ for all preferred models M of R.*

5 Soundness and Completeness

We have now assembled the main components of our theory of intuitionistic logic programming augmented by negation-as-failure: (i) the inference system of Section 3, and (ii) the semantics of Section 4. The remaining task is to equate these two components by proving a soundness and completeness theorem.

We do this in two steps: We first construct a *canonical model*, M_R, of a stratified rulebase R. We then show that the canonical model M_R is a preferred model of R, and, up to equivalence, the only preferred model of R. Soundness and completeness then follow immediately. Because of space limitations, all proofs are omitted. The interested reader is referred to [5] for details.

To construct the canonical model, we first define a box notation for our inference system analogous to the box notation introduced in Section 4.3.

Definition 5.1 (Monotonic Inference) *Let R be a stratified rulebase and let ϕ be a closed goal. Then $R \vdash \Box\phi$ iff $R + DB \vdash \phi$ for all databases DB. If Φ is a set of goals, then $R \vdash \Box\Phi$ iff $R \vdash \Box\phi$ for each goal $\phi \in \Phi$.*

Note that if $R \vdash \Box\phi$ then $R + DB \vdash \Box\phi$ for any database DB. For this reason, if $R \vdash \Box\phi$ then we say that R *infers ϕ monotonically.* We use monotonic inference to define a binary relation on databases. We then use this relation to construct a non-monotonic structure M_R, which we call the *canonical model* of the stratified rulebase R.

Definition 5.2 *Let R be a stratified rulebase and let DB_1 and DB_2 be two databases. $DB_2 \geq DB_1$ iff $R + DB_2 \vdash \Box DB_1$.*

Definition 5.3 (Canonical Model) *Let R be a stratified rulebase. The canonical model of R is the tuple $M_R = \langle S, \leq, \pi \rangle$, where*

$$S \;=\; \textit{The set of all databases.}$$

$$\leq \;\; \textit{is the binary relation given by Definition 5.2.}$$

$$\pi(DB) \;=\; \{B \mid R + DB \vdash B \;\; \textit{where } B \textit{ is atomic}\}$$

It is not hard to show that $\leq$ is transitive and reflexive [5]. The canonical model is therefore a valid, non-monotonic structure, by Definition 4.6. The following theorem is our first major result about canonical models.

Theorem 5.1 *Let R be a stratified rulebase. Then for any goal ϕ,*

$$R + DB \vdash \phi \quad \textit{iff} \quad DB, M_R \models \phi$$

Corollary 5.2 *The canonical model M_R is a model of R.*

Corollary 5.3 *$M_R \models \phi$ iff $R \vdash \phi$ for all goals ϕ.*

Using Corollary 5.3, we get the theorem below, which is our second major result about canonical models.

Theorem 5.4 *Suppose that N is a model of a stratified rulebase R such that $goals(M_R)|_i = goals(N)|_i$ for all $1 \leq i < j$. Then $goals(M_R)|_j^+ \subseteq goals(N)|_j^+$.*

Theorem 5.4 gives us half of what we need to prove that M_R is a preferred model of R. In addition, we need to show that $goals(N)|_j^- \subseteq goals(M_R)|_j^-$. The notion of consistent models makes this task possible.

Lemma 5.5 *Suppose that N is a consistent model of R such that $goals(M_R)|_j^+ \subseteq goals(N)|_j^+$. Then $goals(N)|_j^- \subseteq goals(M_R)|_j^-$.*

The following result is an immediate consequence of Theorem 5.4 and lemma 5.5.

Corollary 5.6 *If N is a consistent model of R, then $M_R \preceq N$.*

Corollary 5.6 shows that the canonical model M_R is a preferred model of R. Thus, every stratified rulebase has at least one preferred model. Furthermore, if N is any other preferred model of R, then $N \simeq M_R$. Thus, up to equivalence, the canonical model is the *unique* preferred model of R. Soundness and completeness follow immediately from of this uniqueness.

Corollary 5.7 (Soundness and Completeness) *Let R be a stratified rulebase and ϕ be a goal. Then $R \vdash \phi$ iff $R \models \phi$.*

6 Conclusion

This paper has developed a semantics for the addition of stratified negation-as-failure to intuitionistic logic programming, as a complement to the proof-theoretic treatment in our earlier work [4, 3, 6]. Several difficulties were encountered along the way, and overcome. Problems pointed out by Gabbay in [10] were handled by drawing a distinction between rules: $A \leftarrow B$, and goals: $A \Leftarrow B$. The "bias" of intuitionistic Kripke structures towards monotonic rules was handled simply by dropping the monotonicity condition, and satisfaction in non-monotonic structures was then defined separately for the connectives $\leftarrow$ and $\Leftarrow$. Our basic approach was inspired by Przymusinski's perfect model semantics for stratified Horn-clause logic [20], but with various modifications dictated by the intuitionistic setting. Thus, we defined a preference ordering on non-monotonic structures that minimizes positive goals and maximizes negative goals simultaneously, a more complex criterion than was needed in classical logic. Finally, we defined the semantics of a rule base R in terms of the preferred models of R, and we proved a soundness and completeness theorem using this definition. Note that the class of rules for which this approach works, namely, stratified intuitionistic rulebases, as given by Definition 3.3, is exactly the class of rules needed for the expressibility results in [4, 3]. Harland's proposal for the semantics of negation-as-failure [12], discussed in Sections 1 and 3.5, is complementary to our proposal in this respect.

There are several directions in which the work of the present paper might be extended. First, the logic could be extended to include function symbols. The development in this paper assumes that the logic includes constants but no function symbols. Since we allow the set of constants to be infinite, however, incorporating function symbols without equality, as in Prolog, should be straightforward. See [14, 15] for an example of how this might be done. Second, the requirement that an intuitionistic rulebase be globally stratified could be relaxed to the requirement of *local stratification* [20]. In fact, the rules in Example 2 are not globally stratified, since a relative clause may appear inside the derivation of a sentence, but such rules would be locally stratified by virtue of the usual constraints on string positions. This extension to locally stratified rulebases also seems straightforward. Third, although the full language of intuitionistic logic programming described in [14, 15, 6] includes embedded universal quantifiers, we have considered in this paper only the restricted (and decidable!) sublanguage in which universal quantification is confined to the top level of the rules. Adding negation-as-failure to the full language raises additional problems, and requires another subtle adjustment of the semantics. We will develop these ideas further in subsequent papers.

References

[1] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[2] A.J. Bonner. A Logic for Hypothetical Reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 480–484, Saint Paul, MN, August 21–26 1988. Published in expanded form as Technical Report TR-DCS-230, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903.

[3] A.J. Bonner. Hypothetical Datalog: Negation and Linear Recursion. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 286–300, Philadelphia, PA, March 1989.

[4] A.J. Bonner. Hypothetical Datalog: Complexity and Expressibility. *Theoretical Computer Science (TCS)*, 76:3–51, 1990. Special issue on the 2^{nd} International Conference on Database Theory (ICDT), Bruge, Belgium, Aug/Sept 1988. Also appears, in abridged form, in volume 326 of *Lecture Notes in Computer Science*, pages 144–160. Springer-Verlag, 1988.

[5] A.J. Bonner and L.T. McCarty. Adding Negation-as-Failure to Intuitionistic Logic Programming. Technical report, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, 1992.

[6] A.J. Bonner, L.T. McCarty, and K. Vadaparty. Expressing Database Queries with Intuitionistic Logic. In *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 831–850, Cleveland, Ohio, October 16–20 1989. MIT Press.

[7] A.K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences (JCSS)*, 21(2):156–178, 1980.

[8] A.K. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences (JCSS)*, 25(1):99–128, 1982.

[9] M.C. Fitting. *Intuitionistic Logic, Model Theory and Forcing.* North-Holland, 1969.

[10] D.M. Gabbay. N-Prolog: an Extension of Prolog with Hypothetical Implications. II. Logical Foundations and Negation as Failure. *Journal of Logic Programming (JLP)*, 2(4):251–283, 1985.

[11] D.M. Gabbay and U. Reyle. N-Prolog: an Extension of Prolog with Hypothetical Implications. I. *Journal of Logic Programming (JLP)*, 1(4):319–355, 1984.

[12] J. Harland. A Kripke-like model for negation as failure. In *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 626–642, Cleveland, Ohio, October 1989. MIT Press.

[13] S. Kripke. Semantical Analysis of Intuitionistic Logic. I. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130. North Holland, Amsterdam, 1965.

[14] L.T. McCarty. Clausal Intuitionistic Logic. I. Fixed-Point Semantics. *Journal of Logic Programming (JLP)*, 5(1):1–31, 1988.

[15] L.T. McCarty. Clausal Intuitionistic Logic. II. Tableau Proof Procedures. *Journal of Logic Programming (JLP)*, 5(2):93–132, 1988.

[16] L.T. McCarty. A Language for Legal Discourse. I. Basic Features. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 180–189. ACM Press, June 1989.

[17] D. Miller. A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming (JLP)*, 6:79–108, 1989.

[18] D. Miller. Lexical scoping as universal quantification. In G. Levi and M. Martelli, editors, *Logic Programming: Proceedings of the Sixth International Conference*, pages 268–283, Cambridge, MA, 1989. MIT Press.

[19] R. Pareschi. A definite clause version of categorial grammar. In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, pages 270–277, Buffalo, NY, June 1988.

[20] T. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.