An Intuitionistic Interpretation of Finite and Infinite Failure

(Preliminary Version)

L. Thorne McCarty
Computer Science Department
Rutgers University
New Brunswick, NJ 08903, USA
mccarty@cs.rutgers.edu

Ron van der Meyden
Information Sciences Laboratory
NTT Basic Research Laboratories
3-9-11 Midori-cho Musashino-shi
Tokyo 180, Japan
meyden@ntt-20.ntt.jp

Abstract

In this paper, we propose an intuitionistic semantics for negation-as-failure in logic programs. The basic idea is to work with the *completion* of the program, not in classical logic, but in intuitionistic (or, more precisely, minimal) logic. Moreover, we consider two forms of completion: (1) first-order predicate completion, as defined by Clark, which is related to SLDNF resolution; and (2) second-order completion, using *circumscription*. Specifically, given any program $\mathcal{R}$, we write a sentence in second-order intuitionistic logic, called the *partial intuitionistic circumscription* axiom, and we declare this sentence to be the "meaning" of $\mathcal{R}$. We then show that our semantics – called the *PIC* semantics – agrees with the perfect model semantics in the case of a locally stratified program. For nonstratified programs, we show that the *PIC* semantics is strictly stronger than the (3-valued) wellfounded semantics. We also show a more complex relationship to the (2-valued) stable model semantics. One advantage of our approach, we claim, is that it is "declarative" in the traditional sense, i.e., the meaning of a program is just the set of logical consequences of a single sentence in second-order intuitionistic logic.

1 Introduction

Historically, there have been two main approaches to the problem of negation in logic programming: (1) the *program completion* approach [6, 9, 19, 24]; and (2) the *canonical model* approach [2, 35, 12, 41]. These approaches are known to differ on certain critical examples.

Example 1.1: This example appears in [42], but similar examples are discussed elsewhere. The following *rules* are intended to define 'R' as the difference between the transitive closure of 'B' and the transitive closure of 'A':

$$\mathrm{P}(x,y) \Leftarrow \mathrm{A}(x,y) \tag{1}$$

$$\mathrm{P}(x,y) \Leftarrow \mathrm{A}(x,z) \wedge \mathrm{P}(z,y) \tag{2}$$

$$\mathrm{Q}(x,y) \Leftarrow \mathrm{B}(x,y) \tag{3}$$

$$\mathrm{Q}(x,y) \Leftarrow \mathrm{B}(x,z) \wedge \mathrm{Q}(z,y) \tag{4}$$

$$\mathrm{R}(x,y) \Leftarrow \mathrm{Q}(x,y) \wedge \sim \mathrm{P}(x,y) \tag{5}$$

Suppose we add the following *facts*: A(1,2), A(2,1), B(2,3), B(3,2). The query 'R(3,2)?' succeeds using SLDNF resolution, since the goal 'P(3,2)' fails finitely. However, the query 'R(2,3)?' does not succeed. Although the goal 'P(2,3)' fails, too, it does so in an "infinite" failure mode which is not detectable by SLDNF resolution. □

All known versions of the "completed program" agree with the results of SLDNF resolution on this example, that is, they either give no answer to the query 'R(2,3)?' [6] or they explicitly declare the answer to be undefined [9, 19]. In response, there have been various proposals for a "canonical model" to serve as the intended meaning of such a program: the perfect model [2, 35]; the (2-valued) stable model [12]; or the (3-valued) wellfounded model [41, 42, 36]. Although disagreements about specific details still exist, most researchers now seem to agree that the canonical model approach, in general, provides the better intuitive explanation of the meaning of negation in a general logic program.

Nevertheless, there are advantages to the program completion approach. The basic idea, due to Clark [6], is very appealing. Clark pointed out that a logic program provides *sufficient* conditions for a set of defined predicates, while a programmer usually thinks of the program as a set of *necessary and sufficient* conditions. Accordingly, Clark's "predicate completion" added the "only if" half of the definitions – written out explicitly in first-order logic – back to the program. Since the completed program was just a set of sentences in first-order logic, its "meaning" could then be defined by the ordinary concept of logical *entailment*. Thus the semantics was "declarative" in the traditional sense. Unfortunately, the various canonical model approaches do not have this property. The canonical model is identified either by an explicit reference to the stratification of the program [2, 35], or by a syntactic transformation that depends on the model in which the program is being

interpreted [12, 36], or by a fixed-point computation [2, 41, 42, 40]. Fixed-point computations are by far the most common device, but these are more proof-theoretic than model-theoretic in the traditional sense.

In this paper, we propose a semantics for negation-as-failure that shares some of the advantages of the program completion approach, and yet agrees with the canonical model approach on the critical examples. We work within the framework of *intuitionistic logic programming* [10, 11, 28, 29, 33, 34, 15], which views a program as a set of sentences in intuitionistic logic rather than classical logic. More precisely, we use a minor variant of intuitionistic logic called *minimal logic* [17]. Our first step is to rewrite every occurrence of negation-as-failure ($\sim$) as an occurrence of ordinary negation ($\neg$) in minimal logic. For example, rule (5) becomes:

$$\mathrm{R}(x,y) \Leftarrow \mathrm{Q}(x,y) \ \wedge \ \neg\mathrm{P}(x,y) \tag{6}$$

We then "complete" a program, $\mathcal{R}$, in two different ways: (1) We define the *first-order* completion of $\mathcal{R}$ in minimal logic, and we show that this provides a reasonable interpretation of "finite" failure in Example 1.1 and similar examples. (2) We define the *second-order* completion of $\mathcal{R}$ in minimal logic, using *circumscription* [26, 27], and we show that this provides, in addition, a reasonable interpretation of "infinite" failure in Example 1.1 and similar examples. In general, given any $\mathcal{R}$, we write a second-order sentence, called the *partial intuitionistic circumscription* axiom and denoted by $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$, and we declare this sentence to be the "meaning" of $\mathcal{R}$.

To study the implications of our proposed semantics – called the *PIC* semantics, for short – we develop the concept of a *final Kripke model*, which is analogous to the concept of an initial model in classical logic [14, 25]. Basically, once we have shown that $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ has a final Kripke model, $\boldsymbol{K}$, we can determine the positive and negative ground literals that are entailed by $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ just by looking at $\boldsymbol{K}$. Using this device, we can compare our semantics with the various other semantics that have been proposed for negation-as-failure. For example, we show that the *PIC* semantics is strictly stronger than the (3-valued) wellfounded semantics [42, 36], i.e., any positive or negative ground literal that is entailed by the (3-valued) wellfounded semantics is also entailed by $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$. (We also present examples to show that the converse of this proposition is not true, and we let the reader decide whether the additional strength of the *PIC* semantics is warranted.) The relationship to the (2-valued) stable model semantics [12] is more complex. First, we show that $\mathcal{R}$ has (2-valued) stable models if and only if $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ is noncontradictory. Second, if this is the case, then the final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ has both a least element and a set of maximal elements that correspond exactly to the (2-valued) stable models of $\mathcal{R}$. Finally, if $\mathcal{R}$ is locally stratified, then the various canonical models coincide, and the final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ has exactly one element which is equivalent to the perfect model [2, 35] of $\mathcal{R}$. In this case, though, it is

important to note that the *PIC* semantics makes no reference at all to the stratification of $\mathcal{R}$.

The paper is organized as follows: Section 2 covers the necessary background material on intuitionistic and minimal logic; Section 3 discusses the first-order completion of a program; and Section 4 discusses the second-order completion using circumscription. Section 5 compares the *PIC* semantics to the various canonical model semantics. Section 6 summarizes our results, and outlines an interesting direction for future research. We confine our discussion to the function-free case, but, at the cost of additional technical complications, we could easily extend our analysis to incorporate the kinds of function symbols (i.e., data structures) that occur in conventional logic programs. Because of space limitations, theorems are stated without proofs. Full proofs will be included in an expanded version of the paper.

2 Foundations: Minimal Logic

We begin with some technical background. We assume that the reader is generally familiar with the Kripke semantics for first-order intuitionistic logic, as given in [18, 8], and we simply review our notation here. Minimal logic [17] differs from intuitionistic logic only in the treatment of negation, as we will see below. We also define in this section the concept of a final Kripke model, and we present enough of the rudiments of second-order intuitionistic logic [38] to explain the partial circumscription axiom in Section 4. Our exposition is, necessarily, terse. We urge the reader to skim it on a first pass, and refer back as needed.

Let $\mathcal{L}$ be a first-order language, and let $\mathcal{L}(\mathbf{c})$ be the language $\mathcal{L}$ augmented by an arbitrary set of new constants $\mathbf{c}$. For simplicity, we assume that $\mathbf{c}$ is countable. We write a Kripke structure for $\mathcal{L}$ as a quadruple $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$, where $\mathbf{K}$ is a nonempty set of *states*, '$\leq$' is a partial order on $\mathbf{K}$, and $\mathbf{u}$ is a monotonic mapping from the states of $\mathbf{K}$ to nonempty sets of individual constants in $\mathcal{L}(\mathbf{c})$. Intuitively, the third component, $\mathbf{h}$, tells us the ground atomic formulae that are true at each state of $\mathbf{K}$, but since we eventually want to extend our language to second-order, we use a slightly more complicated definition of $\mathbf{h}$ than usual. We first define an *intuitionistic relation* R of arity n to be a function that assigns to every state $s \in \mathbf{K}$ a subset of the n–fold Cartesian product of $\mathbf{u}(s)$ with itself, subject to the requirement that $R(s_1) \subseteq R(s_2)$ whenever $s_1 \leq s_2$. We then define $\mathbf{h}$ to be a mapping from the predicate constants in $\mathcal{L}$ to the set of intuitionistic relations on $\mathbf{K}$. The atomic clause of the "forcing" relation [8] is thus:

$$s, \mathbf{K} \models P(c_1, \ldots, c_n) \quad \text{iff} \quad \langle c_1, \ldots, c_n \rangle \in \mathbf{h}(P)(s), \text{ for } P \text{ a predicate constant of arity } n.$$

The remaining clauses are defined as usual. The most important, for our purposes, are the following:

$s, \mathbf{K} \models \mathcal{B} \Leftarrow \mathcal{A}$ iff $s', \mathbf{K} \models \mathcal{A}$ implies $s', \mathbf{K} \models \mathcal{B}$ for every $s' \geq s$ in $\mathbf{K}$, and all the individual constants in $\mathcal{A}$ and $\mathcal{B}$ are in $\mathbf{u}(s)$,

$s, \mathbf{K} \models (\forall x)\mathcal{A}(x)$ iff $s', \mathbf{K} \models \mathcal{A}(c)$ for every $s' \geq s$ in $\mathbf{K}$, and for all individual constants c in $\mathbf{u}(s')$.

If $s, \mathbf{K} \models \mathcal{A}$ for every $s \in \mathbf{K}$, we say that $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ *satisfies* $\mathcal{A}$. If $s, \mathbf{K} \models \mathcal{A}$ for every $s \in \mathbf{K}$ such that the individual constants in $\mathcal{A}$ are in $\mathbf{u}(s)$, then we say that $\mathcal{A}$ is *true* in $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$. Finally, if Φ is a set of sentences and ψ is a sentence, we write $\Phi \models \psi$ if and only if ψ is true in every Kripke structure that satisfies Φ.

In this paper, we are primarily interested in a restricted subset of intuitionistic (or minimal) logic, illustrated by rule (6). This rule is called an *embedded negation*, and it is a special case of the *embedded implications* which have been discussed recently by several authors [10, 11, 28, 29, 33]. Negation in intuitionistic (or minimal) logic is treated as follows: We add a special nullary predicate '$\perp$' to $\mathcal{L}$ to denote a contradiction, and we write '$\neg\mathcal{A}$' as an abbreviation of '$\perp \Leftarrow \mathcal{A}$'. Semantically, if we want our logic to be intuitionistic, we stipulate that $\mathbf{h}(\perp)(s) = \emptyset$ for every $s \in \mathbf{K}$, and this forces $\Phi \models \perp$ to imply $\Phi \models \psi$ for every ψ. However, since '$\perp$' only appears in the conclusion of an implicational goal in rules such as (6), we have no reason to apply the principle "*ex falso sequitur quodlibet*" in our restricted language. Instead, we use minimal logic, which allows $\mathbf{h}(\perp)(s) = \{\langle\rangle\}$, i.e., it allows the proposition '$\perp$' to be "true" in a state s of $\mathbf{K}$. Such a state is said to be "contradictory", but it is not excluded from the Kripke structure. We will see examples of this phenomenon in subsequent sections.

We now define the concept of a final Kripke model, which is analogous to the concept of an initial model in classical logic [14, 25]. We also define the closely related concept of a *generic* model. First, let ι be an isomorphic mapping on the domains of two Kripke structures that is constrained to be an identity on the constants in $\mathcal{L}$. Write $\simeq_\iota$ to mean "isomorphic under ι" and extend this notation in the obvious way to sets and relations. We define a homomorphism τ relative to a fixed ι as follows:

Definition 2.1: Let $\boldsymbol{J}_1 = \langle \mathbf{J}_1, \leq_1, \mathbf{h}_1, \mathbf{u}_1 \rangle$ and $\boldsymbol{J}_2 = \langle \mathbf{J}_2, \leq_2, \mathbf{h}_2, \mathbf{u}_2 \rangle$ be two Kripke structures for $\mathcal{L}$. A mapping $\tau : \mathbf{J}_1 \rightarrow \mathbf{J}_2$ is a *homomorphism* from $\boldsymbol{J}_1$ into $\boldsymbol{J}_2$ if and only if:

1. For every $s, s' \in \mathbf{J}_1$, if $s \leq_1 s'$ then $\tau(s) \leq_2 \tau(s')$.
2. For every $s \in \mathbf{J}_1$,
   (a) $\mathbf{u}_1(s) \simeq_\iota \mathbf{u}_2(\tau(s))$, and
   (b) $\mathbf{h}_1(P)(s) \simeq_\iota \mathbf{h}_2(P)(\tau(s))$ for every predicate constant P.

Now let $\mathcal{K}$ be an arbitrary class of Kripke structures and assume that $\boldsymbol{K} = \langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ is a member of $\mathcal{K}$.

Definition 2.2: $\boldsymbol{K}$ is a *final Kripke structure* for $\mathcal{K}$ if and only if, for every $\boldsymbol{J} \in \mathcal{K}$ and every domain isomorphism ι, there exists a *unique* homomorphism from $\boldsymbol{J}$ into $\boldsymbol{K}$.

It is easy to see that two final Kripke structures for $\mathcal{K}$ are isomorphic, and thus either one could be designated as "the" final Kripke structure. Now let Ψ be a class of sentences. We say that $\boldsymbol{K}$ is *generic in* $\mathcal{K}$ *for* Ψ if and only if, for every $\psi \in \Psi$, ψ is true in $\boldsymbol{K}$ if and only if ψ is true in every $\boldsymbol{J} \in \mathcal{K}$.

Theorem 2.3: Let Ψ be the class of Horn clauses. Then any final Kripke structure for $\mathcal{K}$ is generic in $\mathcal{K}$ for Ψ.

We typically use this result as follows: We take $\mathcal{K}$ to be the class of Kripke structures that satisfy some set of rules $\mathcal{R}$, and we try to find a final Kripke structure, $\boldsymbol{K}$, for $\mathcal{K}$. If such a $\boldsymbol{K}$ exists, we call it a *final Kripke model* for $\mathcal{R}$. Theorem 2.3 then tells us that a Horn clause is entailed by $\mathcal{R}$ if and only if it is true in $\boldsymbol{K}$.

For example, suppose $\mathcal{R}$ is a set of embedded negations. We outline here the construction of a final Kripke model for $\mathcal{R}$, and refer the reader to [28] for details. First, for any $\mathcal{L}(\mathbf{c})$, let $\mathbf{K}$ be the set of all pairs $\langle I, U\rangle$, where U is any nonempty set of constants in $\mathcal{L}(\mathbf{c})$ that includes the constants in $\mathcal{R}$, and I is any Herbrand interpretation for $\mathcal{L}$ over the universe U. Set $\langle I_1, U_1\rangle \leq \langle I_2, U_2\rangle$ if and only if $I_1 \subseteq I_2$ and $U_1 \subseteq U_2$. Define $\mathbf{u}(\langle I, U\rangle) = U$, and define:

$$\mathbf{h}(P)(\langle I, U\rangle) = \{\langle c_1, , \ldots, c_n\rangle \mid P(c_1, \ldots, c_n) \in I\}$$

for every predicate constant P. Now let $\mathbf{K^*}$ be the largest subset of $\mathbf{K}$ such that $\langle \mathbf{K^*}, \leq, \mathbf{h}, \mathbf{u}\rangle$ satisfies $\mathcal{R}$. (This set can always be constructed as the greatest fixed point of the monotonic "deletion" operator discussed in [28].) It is straightforward to show:

Theorem 2.4: $\langle \mathbf{K^*}, \leq, \mathbf{h}, \mathbf{u}\rangle$ is a final Kripke model for $\mathcal{R}$.

Theorem 2.3 now tells us that a Horn clause is entailed by $\mathcal{R}$ if and only if it is true in $\langle \mathbf{K^*}, \leq, \mathbf{h}, \mathbf{u}\rangle$.

However, the use of Theorem 2.3 is not limited to first-order logic. It applies also to second-order logic, which we now define. If $\mathcal{L}$ is understood as a second-order language, we add an explicit assignment σ to our semantic definition to handle the predicate variables, X, Y, Z, etc., thus generalizing the three-place relation '$s, \mathbf{K} \models \mathcal{A}$' to a four-place relation '$\sigma, s, \mathbf{K} \models \mathcal{A}$'. Specifically, we define σ to be a mapping from the predicate variables in $\mathcal{L}$ to the set of intuitionistic relations on $\mathbf{K}$, and we add the following clause to the definition of the "forcing" relation:

$$\sigma, s, \mathbf{K} \models X(c_1, \ldots, c_n) \quad \text{iff} \quad \langle c_1, \ldots, c_n\rangle \in \sigma(X)(s), \text{ for } X \text{ a predicate variable of arity } n.$$

The remaining clauses are unchanged, except that '$s, \mathbf{K} \models \mathcal{A}$' is replaced everywhere by '$\sigma, s, \mathbf{K} \models \mathcal{A}$'. Finally, we add two clauses for the second-order

quantifiers. Let σ_R^X denote the assignment that is identical to σ except that the variable X is mapped to the intuitionistic relation R. Using this notation, we define:

$$\sigma, s, \mathbf{K} \models (\forall X)\mathcal{A}(X) \quad \text{iff} \quad \sigma_R^X, s, \mathbf{K} \models \mathcal{A}(X), \text{ for every intuitionistic relation } R \text{ on } \mathbf{K} \text{ with the same arity as } X,$$

and similarly for the second-order existential quantifier. The definition of entailment is unchanged, and therefore Theorem 2.3 still applies. In Section 4, we will show how to construct final Kripke models for certain sentences in second-order intuitionistic (or minimal) logic.

3 Predicate Completion

We are primarily interested in circumscription in this paper, but we first present a proof procedure for *first-order predicate completion* and show that it is sound and complete with respect to minimal logic. We also analyze, briefly, the relationship between our proof procedure and SLDNF resolution.

The definition of predicate completion in minimal logic is the same as Clark's original definition [6]. For example, if $\mathcal{R}$ consists of the rules and facts in Example 1.1, then the "only if" half of the completed definition of 'P' and 'A' would be:

$$\mathrm{P}(x, y) \Rightarrow \mathrm{A}(x, y) \ \vee \ (\exists z)[\mathrm{A}(x, z) \wedge \mathrm{P}(z, y)] \tag{7}$$

$$\mathrm{A}(x, y) \Rightarrow [x = 1] \wedge [y = 2] \ \vee \ [x = 2] \wedge [y = 1] \tag{8}$$

As usual, we add these sentences to $\mathcal{R}$. We also add Clark's equational theory, written in minimal logic, e.g., we add $\bot \Leftarrow [c = d]$ for all distinct constants c and d in $\mathcal{L}$. Thus the completion of $\mathcal{R}$, written $Comp(\mathcal{R})$, consists of: (i) embedded negations, some of which may have '$\bot$' as a conclusion; and (ii) disjunctive and existential assertions in the form:

$$P(\mathbf{x}) \Rightarrow \bigvee_{i=1}^{k} (\exists \mathbf{y}_i) \bigwedge_{j=1}^{n(i)} \mathcal{B}_{ij}(\mathbf{x}; \mathbf{y}_i),$$

where each $\mathcal{B}_{ij}$ is either an equality or a (positive or negative) literal. For this syntactically restricted language, the following simplified "sequent calculus" suffices as a proof theory:

(1) $\Phi \vdash A$ if there is a rule $P(\mathbf{x}) \Leftarrow \bigwedge_{i=1}^{k} \mathcal{A}_i(\mathbf{x})$ in Φ and a ground substitution θ such that $A = P(\mathbf{x})\theta$ and $\Phi \vdash \mathcal{A}_i(\mathbf{x})\theta$ for $i = 1, \ldots, k$.

(2) $\Phi \vdash \neg A$ if $\Phi \cup \{A\} \vdash \bot$.

(3) $\Phi \vdash \psi$ if there is a rule $P(\mathbf{x}) \Rightarrow \bigvee_{i=1}^{k} (\exists \mathbf{y}_i) \bigwedge_{j=1}^{n(i)} \mathcal{B}_{ij}(\mathbf{x}; \mathbf{y}_i)$ in Φ and a ground substitution θ such that $\Phi \vdash P(\mathbf{x})\theta$, and for each $i = 1, \ldots, k$,

$$\Phi \cup \bigcup_{j=1}^{n(i)} \{\mathcal{B}_{ij}(\mathbf{x}; \mathbf{c}_i)\theta\} \vdash \psi,$$

where $\mathbf{c}_i$ is a tuple of constants that do not appear anywhere in Φ or ψ.

Let ψ be a (positive or negative) ground literal in $\mathcal{L}$.

Theorem 3.1: *Comp*$(\mathcal{R}) \vdash \psi$ iff *Comp*$(\mathcal{R}) \models \psi$.

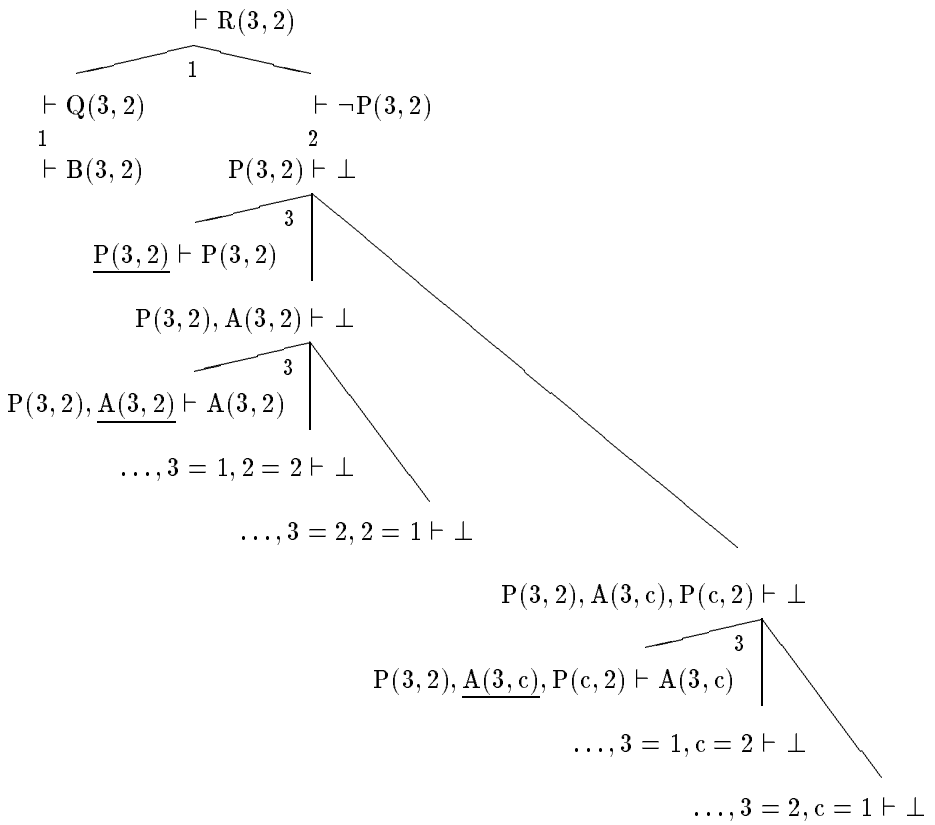

Figure 1: A Proof for Example 1.1

Figure 1 shows a proof that *Comp*$(\mathcal{R}) \vdash$ R(3,2) in this system when $\mathcal{R}$ consists of the rules and facts in Example 1.1. The main point to note is that the goal: $\vdash \neg$P(3,2) is reduced by proof step (2) to the goal: {P(3,2)} $\vdash \bot$, and proof step (3) is then applied repeatedly to generate the possible "expansions" of P(3,2). Eventually, all the branches of this proof tree succeed, using inequalities such as $\bot \Leftarrow [3 = 1]$ and $\bot \Leftarrow [3 = 2]$. By contrast, an attempt to show *Comp*$(\mathcal{R}) \vdash$ R(2,3) would not succeed.

Notice that the proof tree in Figure 1 is isomorphic to the SLDNF tree for the goal R(3,2). In particular, the "expansions" of P(3,2) by proof step (3) are isomorphic to the finitely failed SLD trees for the goal P(3,2). However, our proof procedure is stronger than SLDNF resolution, in general, as we now observe.

Example 3.2: Consider the following $\mathcal{R}$:

$$\mathrm{P}(x) \Leftarrow \mathrm{Q}(x) \wedge \mathrm{R}(x) \wedge \sim \mathrm{S}(x) \tag{9}$$

$$\mathrm{S}(x) \Leftarrow \mathrm{Q}(x) \wedge \mathrm{R}(x) \tag{10}$$

$$\mathrm{Q}(x) \Leftarrow \mathrm{R}(x) \tag{11}$$

$$\mathrm{R}(x) \Leftarrow \mathrm{Q}(x) \tag{12}$$

Can the query 'P(a)' succeed? Intuitively, this can happen only if 'S(a)' both succeeds and fails, which is a contradiction. Figure 2 shows a proof that $\mathit{Comp}(\mathcal{R}) \vdash \neg\mathrm{P(a)}$, as expected. Again, proof step (3) is used to "expand" the atom P(a), but this time the expansion includes the negated literal $\neg$S(a). Proof step (1) is now used to construct a proof of S(a), which succeeds by the use of rule (10) $\Box$

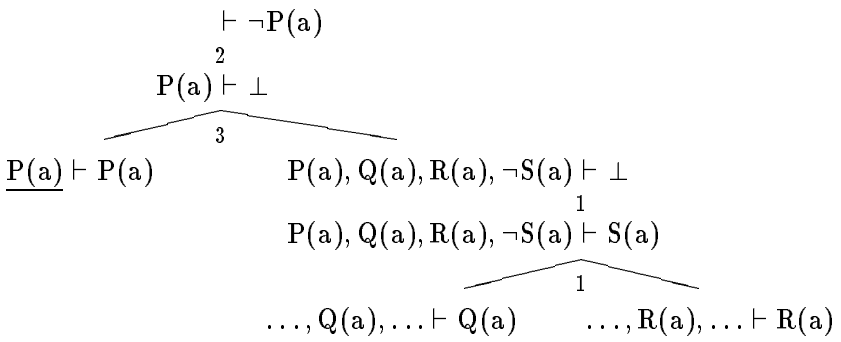

Figure 2: A Proof for Example 3.2

Notice that the proof in Figure 2 is *almost* isomorphic to SLDNF resolution: To show that P(a) fails, we try to show that S(a) succeeds, etc. However, the proof of S(a) is allowed to succeed in Figure 2 by using the "expansions" generated by P(a), and it therefore detects a contradiction that would not be detected by SLDNF resolution. It is natural to call this proof procedure SLDNF+CC, i.e., SLDNF with "contradiction checking". It shares an important property with SLDNF, namely, the fact that proof step (3) is only applied to expand those atoms that already appear on the left-hand side of the sequent. We now observe that this property cannot be guaranteed, in general.

Example 3.3: This pathological example appears in almost every discussion of negation in logic programming. Let $\mathcal{R}$ be:

$$\mathrm{P}(x) \Leftarrow \sim \mathrm{P}(x) \tag{13}$$

We have $Comp(\mathcal{R}) \vdash \neg \mathrm{P(a)}$ and $Comp(\mathcal{R}) \vdash \mathrm{P(a)}$, and therefore $Comp(\mathcal{R}) \vdash \bot$. A proof of this latter fact is shown in Figure 3. Notice that proof step (3) is used at the top of this proof tree, before any atoms have been added to the left-hand side of the sequent. The positive literal 'A(a)' is not provable in this system, despite the proof of a contradiction here, since we are using minimal logic rather than intuitionistic logic. However, we can show that $Comp(\mathcal{R}) \vdash \neg \mathrm{A(a)}$ by a minor modification of the proof in Figure 3. We would thus have a proof of a negative literal that is not isomorphic to an SLDNF tree. □

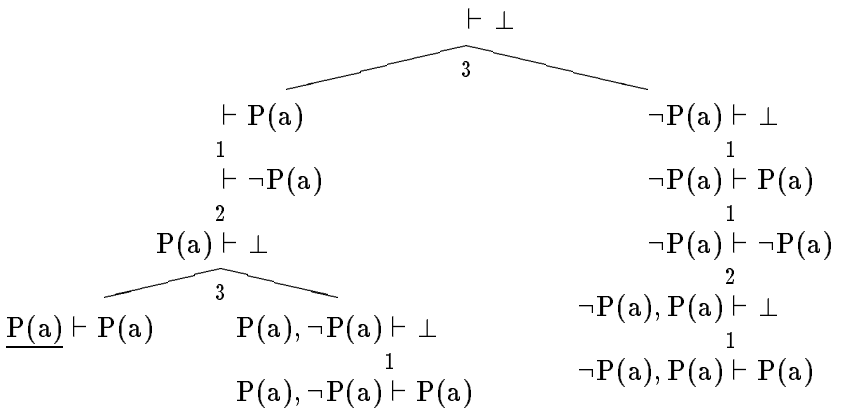

Figure 3: A Proof for Example 3.3

It turns out that our observations about these examples can be generalized. We assume familiarity with the definitions of "allowed", "strict" and "call consistent" as given by Kunen [20]. Note that Examples 3.2 and 3.3 are not strict, and Example 3.3 is neither allowed nor call consistent. The following result follows from [20]:

Theorem 3.4: If $\mathcal{R}$ is allowed and strict, then $Comp(\mathcal{R}) \vdash \psi$ if and only if ψ is provable by SLDNF.

Let us define SLDNF+CC to consist of proof steps (1)–(3), but with the restriction that proof step (3) can only be applied to expand atoms that appear explicitly on the left-hand side of the sequent. We then have the following stronger result:

Theorem 3.5: Assume that $\mathcal{R}$ is allowed and call consistent. Then $Comp(\mathcal{R}) \vdash \psi$ if and only if ψ is provable by SLDNF+CC.

It is also possible to relax the "allowed" condition in these theorems, and establish a connection between our proof procedure and Chan's "constructive negation" [5], but this is beyond the scope of the present paper.

4 Circumscription

Example 1.1 shows that $Comp(\mathcal{R})$ is too weak to capture our intuitions about negation in logic programming, and we now investigate a stronger form of "completion" based on *circumscription* [26, 27]. The circumscription axiom has usually been studied as a sentence in second-order classical logic. However, in this section, extending previous results reported in [32, 30], we show that circumscription in intuitionistic (or minimal) logic has several interesting properties that do not appear in the classical version.

We first define our notation. Let $\boldsymbol{P} = <P_1, P_2, \ldots, P_k>$ be a tuple of predicates. Let $\mathcal{R}(\boldsymbol{P})$ denote the conjunction of the sentences in $\mathcal{R}$, with the predicate symbols in $\boldsymbol{P}$ treated as free parameters, and let $\mathcal{R}(\boldsymbol{X})$ be the same as $\mathcal{R}(\boldsymbol{P})$ but with the predicate constants $<P_1, P_2, \ldots, P_k>$ replaced by predicate variables $<X_1, X_2, \ldots, X_k>$. The *circumscription axiom* is the following sentence in second order intuitionistic (or minimal) logic:

$$\mathcal{R}(\boldsymbol{P}) \wedge (\forall \boldsymbol{X})[\mathcal{R}(\boldsymbol{X}) \wedge \bigwedge_{i=1}^{k} (\forall \mathbf{x})[X_i(\mathbf{x}) \Rightarrow P_i(\mathbf{x})] \Rightarrow \bigwedge_{i=1}^{k} (\forall \mathbf{x})[P_i(\mathbf{x}) \Rightarrow X_i(\mathbf{x})]]$$

We denote this sentence by $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$.

We now show how to construct a final Kripke model for $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ when $\mathcal{R}(\boldsymbol{P})$ is a set of Horn clauses. We assume that $\boldsymbol{P}$ is a tuple consisting of the "defined predicates", i.e., the predicates that appear in the conclusion of some Horn clause in $\mathcal{R}$. All other predicates are "base predicates". For example, if $\mathcal{R}$ consists of rules (1)–(2) in Example 1.1, then $\boldsymbol{P} = <\mathrm{P}>$ and the set of base predicates is $\{\mathrm{A}, \bot\}$, but if $\mathcal{R}$ also includes the facts 'A(1,2)' and 'A(2,1)', then $\boldsymbol{P} = <\mathrm{P}, \mathrm{A}>$ and the set of base predicates is $\{\bot\}$. The construction of a final Kripke model for $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ is similar to the construction in Section 2, except that we work with Herbrand interpretations over base predicates only. Specifically, let $\mathbf{C}$ be the set of all pairs $\langle I, U\rangle$, where U is any nonempty set of constants in $\mathcal{L}(\mathbf{c})$ that includes the constants in $\mathcal{R}$, and where I is any Herbrand interpretation for the *base predicates* in $\mathcal{L}$ over the universe U. The definitions of '$\leq$' and $\mathbf{u}$ are the same as in Section 2, but the definition of $\mathbf{h}$ is different. Intuitively, we want $\mathbf{h}$ to give us the smallest Herbrand model of $\mathcal{R}$ over the universe U that contains I. Formally, we do this by first defining $\mathbf{T}_U(I) = I \cup T_{\mathcal{R}}(I, U)$, where $T_{\mathcal{R}}(I, U)$ is the van Emden-Kowalski [39, 3] fixed-point operator for $\mathcal{R}$ applied to I in a universe U. We then define, for every predicate constant P:

$$\mathbf{h}(P)(\langle I, U\rangle) = \{\langle c_1, , \ldots, c_n\rangle \mid P(c_1, \ldots, c_n) \in \mathbf{T}_U^{\omega} \uparrow (I)\}$$

Theorem 4.1: Let $\mathcal{R}$ be a set of Horn clauses, and let $\boldsymbol{P}$ be a tuple consisting of the defined predicates in $\mathcal{R}$. Then $\langle \mathbf{C}, \leq, \mathbf{h}, \mathbf{u}\rangle$ is a final Kripke model for $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$.

Theorem 2.3 now tells us that a (positive or negative) ground literal is entailed by $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ if and only if it is true in $\langle \mathbf{C}, \leq, \mathbf{h}, \mathbf{u}\rangle$.

However, if we wish to use circumscription to provide an interpretation of negation-as-failure in general logic programs, we need to consider more than just Horn clauses. In [30], we showed that $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ itself does not produce acceptable results when $\mathcal{R}$ is a set of embedded implications or embedded negations. There are two solutions to this problem. One solution, when $\mathcal{R}$ is stratified, is to use *prioritized* circumscription [27, 21]. Thus, in Example 1.1, we could set $\boldsymbol{P}_1 = <\mathrm{P}, \mathrm{A}>$ and $\boldsymbol{P}_2 = <\mathrm{R}, \mathrm{Q}, \mathrm{B}>$, and then circumscribe $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ in that order. We denote the *prioritized circumscription axiom* in intuitionistic (or minimal) logic by $Circ(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}_1,\boldsymbol{P}_2,\ldots,\boldsymbol{P}_n)$, and we refer the reader to [30] for a discussion of its properties. A second solution, which we investigate here, is to circumscribe only certain occurrences of the predicates $\boldsymbol{P}$ in $\mathcal{R}$. Note that the occurrence of 'P' in rules (1) and (2) is part of the definition of transitive closure, but the occurrence of 'P' in rule (6) is quite different. Recall that $\neg\mathrm{P}(x,y)$ is an abbreviation for $\bot \Leftarrow \mathrm{P}(x,y)$. Intuitively, we are asserting $\mathrm{P}(x,y)$ hypothetically in rule (6) and asking whether '$\bot$' follows from this assertion, and we do not want to vary the extension of 'P' in such a situation. This is what we mean by *partial* intuitionistic circumscription.

Formally, let us define $\mathcal{R}(\boldsymbol{P}|\boldsymbol{P}')$ to be the result of replacing every occurrence of a predicate P_i in $\boldsymbol{P}$ that appears inside a negation sign in $\mathcal{R}$ by a new predicate P_i' in $\boldsymbol{P}'$. We can then write $\mathcal{R}(\boldsymbol{X}|\boldsymbol{P}')$ to denote the replacement of the remaining occurrences of predicate constants in $\boldsymbol{P}$ with predicate variables in $\boldsymbol{X}$. The *partial circumscription axiom* is the following sentence in second order intuitionistic (or minimal) logic:

$$Circ(\mathcal{R}(\boldsymbol{P}|\boldsymbol{P}');\boldsymbol{P}) \ \wedge \ \bigwedge_{i=1}^{k} (\forall \mathbf{x})[P_i(\mathbf{x}) \Leftrightarrow P_i'(\mathbf{x})]$$

In other words, we are using the ordinary circumscription axiom in the first conjunct to vary the extension of $\boldsymbol{P}$ in $\mathcal{R}(\boldsymbol{P}|\boldsymbol{P}')$ without varying the extension of $\boldsymbol{P}'$, and we are then asserting in the second conjunct that $\boldsymbol{P}$ and $\boldsymbol{P}'$ are equivalent. We denote the full sentence by $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$.

We now show how to construct a final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$. Let $\mathcal{L}'$ be an extended language that includes a predicate $NotQ$ for every predicate Q in $\mathcal{L}$. Let $\mathcal{R}'$ be the same as $\mathcal{R}$, but with all occurrences of $\neg Q(\mathbf{x})$ replaced by a new atomic formula $NotQ(\mathbf{x})$. Note that $\mathcal{R}'$ is just a set of Horn clauses, so that Theorem 4.1 applies. We also need the following *definitional rules* for our new predicates:

$$\bot \ \Leftarrow \ Q(\mathbf{x}) \wedge NotQ(\mathbf{x}) \tag{14}$$

$$NotQ(\mathbf{x}) \ \Leftarrow \ [\bot \Leftarrow Q(x)] \tag{15}$$

We now construct the desired Kripke model in three stages:

1. Let $\langle \mathbf{C}_1, \leq, \mathbf{h}_1, \mathbf{u}_1 \rangle$ be the final Kripke model for $Circ(\mathcal{R}'(\boldsymbol{P});\boldsymbol{P})$ in the extended language $\mathcal{L}'$.

2. Let $\mathbf{C}_2$ be the largest subset of $\mathbf{C}_1$ such that $\langle \mathbf{C}_2, \leq, \mathbf{h}_1, \mathbf{u}_1 \rangle$ satisfies the definitional rules in (14)–(15). This set can always be constructed as the greatest fixed point of the monotonic "deletion" operator discussed in [28].

3. Set $\langle \mathbf{C}^*, \leq, \mathbf{h}, \mathbf{u} \rangle = \langle \mathbf{C}_2, \leq, \mathbf{h}_1, \mathbf{u}_1 \rangle$.

As an optional final step, we can delete the predicates *NotQ* from the definition of $\mathbf{h}$, thus obtaining a Kripke structure for the original language $\mathcal{L}$. In [30], we stated the following result:

Theorem 4.2: Assume $\mathcal{R}$ is stratified. Then $\langle \mathbf{C}^*, \leq, \mathbf{h}, \mathbf{u} \rangle$ is a final Kripke model for $Circ(\mathcal{R}(\boldsymbol{P}); \boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n)$.

The main result of the present section is:

Theorem 4.3: Whether $\mathcal{R}$ is stratified or not, $\langle \mathbf{C}^*, \leq, \mathbf{h}, \mathbf{u} \rangle$ is a final Kripke model for the partial circumscription axiom $PIC(\mathcal{R}(\boldsymbol{P}); \boldsymbol{P})$.

For a proof of these theorems, but in a more general setting, see [31]. Combining these results with Theorem 2.3, we observe that prioritized circumscription and partial circumscription entail exactly the same (positive or negative) ground literals when $\mathcal{R}$ is stratified.

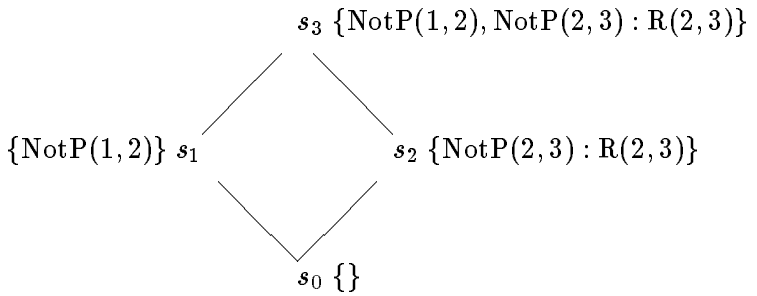

Figure 4: $\langle \mathbf{C}_1, \leq, \mathbf{h}_1, \mathbf{u}_1 \rangle$ in Example 1.1

Example 4.4: Let $\mathcal{R}$ consist of the rules and facts in Example 1.1. Figure 4 shows a portion of the final Kripke model $\langle \mathbf{C}_1, \leq, \mathbf{h}_1, \mathbf{u}_1 \rangle$ in the first stage of our construction. For simplicity, we are only looking at states s in which $\mathbf{u}(s) = \{1, 2, 3\}$, and we are only looking at the base predicates 'NotP(1,2)' and 'NotP(2,3)'. Furthermore, we have omitted the atomic formulae added by $\mathbf{T}_U^\omega \uparrow (\emptyset)$ to s_0, which would also be added to s_1, s_2 and s_3. Thus the reader should imagine that the facts, A(1,2), A(2,1), B(2,3), B(3,2), and the transitive closures of these facts, P(1,2), P(2,1), Q(2,3), Q(3,2), have been added to all the states in Figure 4. We have explicitly listed here only 'R(2,3)', which is added to s_2 and s_3 by the Horn clause version of rule (6) . Let us now apply the "deletion" operator in the second

stage of our construction. States s_1 and s_3 would be deleted, since 'P(1,2) $\wedge$ NotP(1,2)' is true there but '$\bot$' is not. Also, state s_0 would be deleted, since '$[\bot \Leftarrow \mathrm{P}(2,3)]$' is true there but 'NotP(2,3)' is not. We are thus left with a single state, s_2. (As an optional final step, the formula 'NotP(2,3)' could be deleted from s_2.) Thus 'R(2,3)' would remain true in the least state of the final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$. A similar argument applies to 'R(3,2)'. The actual construction would involve many more states than this, but $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$ would still have a unique minimal noncontradictory state in which 'R(2,3)' and 'R(3,2)' are both true. □

Example 4.5: For the pathological rule in Example 3.3, $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$ has a single state, s_U, which is contradictory, for every set of constants U in $\mathcal{L}(\mathbf{c})$. For example, given $U = \{\mathrm{a}\}$, the state s_U with $\mathbf{u}(s_U) = \{\mathrm{a}\}$ would be represented by $\{\mathrm{P(a)}, \bot\}$. □

Example 4.6: For the rules in Example 3.2, $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$ has a single noncontradictory state, s_U, which is empty, for every set of constants U in $\mathcal{L}(\mathbf{c})$. It follows that $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ entails $\neg$P(a), $\neg$S(a), $\neg$Q(a) and $\neg$R(a). □

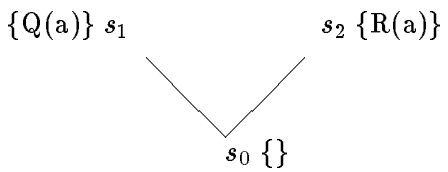

Figure 5: $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$ in Example 4.7

Example 4.7: Consider the following $\mathcal{R}$:

$$\mathrm{P}(x) \Leftarrow \mathrm{Q}(x) \wedge \mathrm{R}(x) \wedge \sim \mathrm{S}(x) \tag{16}$$

$$\mathrm{S}(x) \Leftarrow \mathrm{Q}(x) \wedge \mathrm{R}(x) \tag{17}$$

$$\mathrm{Q}(x) \Leftarrow \sim \mathrm{R}(x) \tag{18}$$

$$\mathrm{R}(x) \Leftarrow \sim \mathrm{Q}(x) \tag{19}$$

This is a variant of Example 3.2. Figure 5 shows all the noncontradictory states of the final Kripke model $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$ with domain $\mathbf{u}(s) = \{\mathrm{a}\}$. It is apparent from this figure that $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \neg$P(a) and $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \neg$S(a). However, both 'Q(a)' and 'R(a)' are unknown, according to the *PIC* semantics. □

In Section 3, we showed that $Comp(\mathcal{R}) \vdash \neg$P(a) when $\mathcal{R}$ consists of the rules in Example 3.2, and the same proof goes through when $\mathcal{R}$ consists of rules (16)–(19) in Example 4.7. In addition, for rules (16)–(19), we can show that $Comp(\mathcal{R}) \vdash \neg$S(a). Neither 'Q(a)' nor 'R(a)' would be provable in this example, as shown by the following:

Theorem 4.8: If $Comp(\mathcal{R}) \models \psi$ then $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \psi$.

5 Canonical Models

Examples 3.2 and 4.7 are useful for understanding the relationship between the *PIC* semantics and the various canonical model semantics that have been proposed for negation-as-failure: the (2-valued) stable model semantics [12]; the (3-valued) wellfounded semantics [42]; and the perfect model semantics [2, 35]. The states s_1 and s_2 in Figure 5 are both (2-valued) stable models of a program $\mathcal{R}$ consisting of rules (16)–(19), but the state s_0 is not stable. Since there is no unique stable model for this program, the (2-valued) stable model semantics gives no answer here at all. For the same program, the (3-valued) wellfounded model is empty, i.e., it contains no true atoms and no false atoms. Thus the (3-valued) wellfounded semantics asserts that $\{\mathrm{P(a)}, \mathrm{S(a)}, \mathrm{Q(a)}, \mathrm{R(a)}\}$ are all unknown, a different answer from the one given by the *PIC* semantics. In contrast, for a program $\mathcal{R}$ consisting of rules (9)–(12), which just happens to be stratified, the *PIC* semantics agrees with all the other semantics. Here, the single state in the final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ is equivalent to the perfect model of $\mathcal{R}$, which is also equivalent to the unique stable model of $\mathcal{R}$ and the total wellfounded model of $\mathcal{R}$. In this section, we show that these relationships are not accidental.

We first establish two results on the structure of $\langle \mathbf{C^*}, \leq, \mathbf{h}, \mathbf{u} \rangle$. These results are relativized to the domain $\mathbf{u}(s)$, as were the examples in Section 4.

Lemma 5.1: For any constant c in $\mathcal{L}$, there exists a *unique minimal* state s in $\mathbf{C^*}$ such that $c \in \mathbf{u}(s)$.

Lemma 5.2: Let s be any noncontradictory state in $\mathbf{C^*}$, and let U be any set of constants in $\mathcal{L}(\mathbf{c})$ such that $\mathbf{u}(s) \subseteq U$. Then there exists a *maximal* noncontradictory state $s' \geq s$ in $\mathbf{C^*}$ such that $\mathbf{u}(s') = U$.

We now establish a relationship between the *PIC* semantics and the (2-valued) stable model semantics. Gelfond and Lifschitz [12] originally defined stable models using a transformation of the program $\mathcal{R}$, but Van Gelder observed in [40] that an equivalent definition could be stated in terms of a *stability transformation* $\mathbf{S}_{\mathcal{R}}(I)$ which operates on the negative literals in the interpretation I. Recall that our construction of a final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ in Theorem 4.3 makes use of the construction $\mathbf{T}^{\omega}_{U} \uparrow (I)$ from Theorem 4.1, in which I is a Herbrand interpretation over a set of base predicates in the form $NotQ$. It is easy to see that $\mathbf{S}_{\mathcal{R}}(I)$ on a universe U is equivalent to $\mathbf{T}^{\omega}_{U} \uparrow (I)$. We thus have the following result:

Lemma 5.3: A state s is a (2-valued) stable model of $\mathcal{R}$ on U if and only if s is a maximal noncontradictory state of $\mathbf{C^*}$ with $\mathbf{u}(s) = U$.

Applying Lemmas 5.1, 5.2 and 5.3, we have the following:

Theorem 5.4: $\mathcal{R}$ has (2-valued) stable models iff $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \not\models \bot$. Furthermore, for any ground atom P:

1. $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \neg P$ iff P is false in all stable models of $\mathcal{R}$.

2. $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models P$ iff
   $P \in \mathbf{S}_{\mathcal{R}}(\{\neg Q \mid Q \text{ is false in all stable models of } \mathcal{R}\})$.

The second part of this theorem is particularly interesting. It says that a positive ground literal is true in the *PIC* semantics if and only if it can be computed by applying the stability transformation to the *intersection* of the *negative literals* in the stable models of $\mathcal{R}$. Notice that the least state of the final Kripke model is not (in general) a stable model of the program, as illustrated in Figure 5, nor is it (in general) the ordinary intersection of the stable models of the program.

Applying known results on the relationship between the (2-valued) stable model semantics and the (3-valued) wellfounded semantics, we now have:

Theorem 5.5: For any ground literal ψ, if $\mathcal{R}$ entails ψ in the (3-valued) wellfounded semantics then $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \psi$.

Finally, for locally stratified programs, it is well known that the various canonical models coincide. Thus:

Theorem 5.6: If $\mathcal{R}$ is locally stratified, then the final Kripke model for $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P})$ has exactly one noncontradictory state, s_U, for each universe U that includes the constants in $\mathcal{R}$. In this case, the state s_U is equivalent to the perfect model of $\mathcal{R}$ on the universe U.

6 Discussion

There is a tension in the logic programming literature between two opposing goals: Should a declarative semantics for negation-as-failure attempt to replicate the behavior of our current generation of PROLOG interpreters? Or should we adopt a declarative semantics that seems intuitively correct, and then modify our PROLOG interpreters accordingly? Most researchers have struck a compromise between these two extremes. In his 1985 article, Shepherdson wrote:

> Perhaps the most useful approach here ... is to devise ways of making negation as failure more complete by cutting loops or by failing queries which contain more or less evident contradictions. [37, p. 190]

As we study these revisions to the procedural semantics, however, it makes sense to search for a declarative semantics that approximates the behavior of this "more complete" interpreter. Ideally, the declarative and procedural semantics should meet each other halfway.

We have suggested in this paper that *partial intuitionistic circumscription (PIC)* provides a reasonable candidate for such a declarative semantics. It has a first-order approximation, $Comp(\mathcal{R})$, which assigns **false** to queries containing "more or less evident contradictions". Thus $Comp(\mathcal{R}) \models \neg$P(a)

in Example 3.2 because 'S(a)' cannot both succeed and fail. *PIC* is a "program completion" semantics, following Clark's original idea [6], but it is not first-order. The completion of a program is given by a sentence in second-order intuitionistic (or minimal) logic called the *partial circumscription axiom*, and it is this axiom which provides the semantic equivalent of "cutting loops". Note that $Comp(\mathcal{R}) \not\models \neg$S(a) in Example 3.2, corresponding to the fact that SLDNF resolution "loops" on the query 'S(a)?', but $PIC(\mathcal{R}(\boldsymbol{P});\boldsymbol{P}) \models \neg$S(a) because the extensions of 'Q' and 'R' are minimized at every state of every Kripke structure that satisfies $\mathcal{R}$.

For stratified programs, such as Examples 1.1 and 3.2, the *PIC* semantics is equivalent to the various canonical model semantics [2, 35, 12, 42], but it is "declarative" in the traditional sense and it makes no reference at all to the stratification. By contrast, most previous applications of circumscription to the semantics of negation-as-failure, such as [22] and [35], have used a prioritized version of the circumscription axiom. Recently (and independently), Fangzhen Lin defined a version of partial circumscription in *classical* logic [23], and showed it to be equivalent to the (2-valued) stable model semantics of Gelfond and Lifschitz [12]. Thus, for stratified programs, partial circumscription in intuitionistic logic gives us exactly the same answers as partial circumscription in classical logic. See [31] for a further comparison of these results.

For nonstratified programs, there are differences among the various semantics, as we have seen. The most interesting, perhaps, is the divergence between the *PIC* semantics and the (3-valued) wellfounded semantics revealed in Theorem 5.5 and Example 4.7. Should 'P(a)' and 'S(a)' be false in Example 4.7? We let the reader decide, but we add a comment here on the source of these differences. Although three-valued logic and intuitionistic logic both reject the law of excluded middle, they do so in different ways. Three-valued logic is *truth-functional*. Sentences are evaluated in a single state of the world, but with three truth values, $\langle$**true**, **undefined**, **false**$\rangle$. Intuitionistic logic is not truth-functional. Sentences are evaluated on sets of states, and in each state there are only two truth values, $\langle$**true**, **unknown**$\rangle$. Think of **true**, in either logic, as: "This query succeeds." In Example 4.7, we can imagine a state of the world in which 'Q(a)' succeeds and one in which 'R(a)' succeeds, but not both, and neither case gives us a state of the world in which 'S(a)' succeeds or in which 'P(a)' succeeds. We thus conclude, in intuitionistic logic, that 'S(a)' and 'P(a)' both fail. In contrast, three-valued logic can only imagine a single state of the world, and all it can do is propagate the truth value **undefined** from 'Q(a)' and 'R(a)' to 'S(a)' and 'P(a)'. Hence the different results.

A possible criticism of the *PIC* semantics concerns the conclusions drawn from a contradictory program, such as Example 3.3. We do not infer all *positive* literals in our system, since we are using minimal logic, but we do infer all *negative* literals, and this may not be the intended interpretation. One solution is to use multiple '$\perp$' predicates in minimal logic, to isolate the

contradictory portions of the program. Another solution is to use a weaker form of implication, such as relevant implication [1, 7] or linear implication [13]. Both relevance logic and linear logic have been proposed as alternative foundations for logic programming [4, 16]. But there is a difficulty to overcome here. We needed a circumscription axiom and a final Kripke model to develop the *PIC* semantics in this paper, and we would need an analogue of these constructs to extend our current approach to a weaker logic. We will discuss these issues in a future paper.

References

[1] A.R. Anderson and N.D. Belnap. *Entailment: The Logic of Relevance and Necessity*. Princeton University Press, 1975.

[2] 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.

[3] K.R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[4] A.W. Bollen. Relevant logic programming. *Journal of Automated Reasoning*, 7(4):563–585, 1991.

[5] D. Chan. Constructive negation based on the competed database. In *Proceedings, Fifth International Conference and Symposium on Logic Programming*, pages 111–125, 1988.

[6] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, 1978.

[7] J.M. Dunn. Relevance logic and entailment. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume III, pages 117–224. D. Reidel, 1986.

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

[9] M.C. Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[10] D.M. Gabbay and U. Reyle. N-PROLOG: An extension of PROLOG with hypothetical implications. I. *Journal of Logic Programming*, 1:319–355, 1984.

[11] D.M. Gabbay and M.J. Sergot. Negation as inconsistency. I. *Journal of Logic Programming*, 3:1–35, 1986.

[12] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings, Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.

[13] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[14] J.A. Goguen and J. Meseguer. EQLOG: Equality, types, and generic modules for logic programming. In D. Degroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations, and Equations*, pages 295–363. Prentice-Hall, 1986.

[15] L. Hallnäs and P. Schroeder-Heister. A proof-theoretic approach to logic programming. I. Clauses as rules. *Journal of Logic and Computation*, 1(2):261–283, 1990.

[16] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic (extended abstract). In *Proceedings, Sixth Annual Symposium on Logic in Computer Science*, pages 32–42, 1991.

[17] I. Johansson. Der minimalkalkül, ein reduzierter intuitionistischer formalismus. *Compositio Mathematica*, 4:119–136, 1937.

[18] S.A. 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, 1965.

[19] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[20] K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.

[21] V. Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, 1985.

[22] V. Lifschitz. On the declarative semantics of logic programs with negation. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 177–192. Morgan Kaufmann, 1988.

[23] F. Lin. *A Study of Nonmonotonic Reasoning*. PhD thesis, Stanford University, 1991. Technical Report STAN-CS-91-1385.

[24] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.

[25] J.A. Makowsky. Why horn formulas matter in computer science: Initial structures and generic examples. In G. Goos and J. Hartmanis, editors, *Mathematical Foundations of Software Development*, pages 374–387. Springer-Verlag, 1985. Lecture Notes in Computer Science, volume 185.

[26] J. McCarthy. Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[27] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[28] L.T. McCarty. Clausal intuitionistic logic. I. Fixed-point semantics. *Journal of Logic Programming*, 5(1):1–31, 1988.

[29] L.T. McCarty. Clausal intuitionistic logic. II. Tableau proof procedures. *Journal of Logic Programming*, 5(2):93–132, 1988.

[30] L.T. McCarty. Circumscribing embedded implications. In A. Nerode et al., editors, *Proceedings, First International Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 211–227. MIT Press, 1991.

[31] L.T. McCarty. Circumscribing embedded implications (without stratifications). Submitted for publication, 1992.

[32] L.T. McCarty and R. van der Meyden. Indefinite reasoning with definite rules. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 890–896, 1991.

[33] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79–108, 1989.

[34] G. Nadathur and D.A. Miller. An overview of λPROLOG. In *Proceedings, Fifth International Conference and Symposium on Logic Programming*, pages 810–827, 1988.

[35] T. Przymusinski. On the declarative semantics of stratified deductive databases and logic programming. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, 1988.

[36] T. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, XIII:445–463, 1990.

[37] J.C. Shepherdson. Negation as failure, II. *Journal of Logic Programming*, 2(3):185–202, 1985.

[38] A. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction.* North-Holland, 1988.

[39] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[40] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Proceedings, Eighth ACM Symposium on the Principles of Database Systems*, pages 1–10, 1989.

[41] A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings, Seventh ACM Symposium on the Principles of Database Systems*, pages 221–230, 1988.

[42] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.