A Kripke-like Model for Negation as Failure

James Harland
Department of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ, UK
jah@lfcs.ed.ac.uk

Abstract

We extend the Kripke-like model theory given in [10] for a fragment of first-order hereditary Harrop formulae to include negated atoms in goals. This gives us a formal framework in which to study the role of Negation as Failure rule. The class of predicates for which Negation As Failure is applicable is discussed, as well as the predicates for which some other form of negation will need to be used. We show how the former class may be incorporated into the model theory, giving a generalisation of the usual T^{ω} construction. No restriction on the class of programs is needed for this approach; the construction may be used for programs which are not locally stratified[14]. This is accomplished by the use of a *success level* and a *failure level* of a goal, either or both of which may be infinite. The resulting T operator is not monotonic, which necessitates a slight departure from the standard procedure, but the important properties of the construction still hold.

1 Introduction

A classic problem in the discussion on the foundations of logic programming is the problem of how to incorporate negation. The usual method is the *Negation as Failure* (NAF) rule [1], i.e. that for a given program P, if a ground atom A fails, then $\neg A$ succeeds. This is justified by the *Closed World Assumption* (CWA), which says that the information given in the program is complete, and so the truth of $P \vdash \neg A$ is the same as that of $P \not\vdash A$. In this way negation is usually restricted to atoms, so that this extension allows goals to be built up from literals, rather than atoms alone.

A common description of the NAF rule is to consider the *completion* of a program, which may be thought of as adding to the program the negation of every atom which does not follow from the program, so that the NAF rule may be thought of as a complementation process. There is an underlying assumption here that the definition of every predicate in the program is complete, so that it is sensible to consider as false everything that is not explicitly stated to be true. Thus in the completion of a program, the law of excluded middle holds for every predicate.

However, whilst this assumption of "universal completeness" will hold for many programs, there are some for which it will not. For example, the append predicate given below is complete in the sense that there is no additional clause we can insert which would correctly extend the append relation; all the information we ever want to consider about appending lists together is given, and so it is correct to apply NAF and the completion process. Thus we may think of the append predicate below as *completely defined.*

$$\begin{array}{l} \forall x \ \mathrm{append}([], x, x) \\ \forall x \forall y \forall z \forall w \ \mathrm{append}(x.y, z, x.w) \subset \mathrm{append}(y, z, w) \end{array}$$

On the other hand, not every predicate will have such a complete definition. For example, a predicate containing information about carcinogens we would wish to consider *incompletely defined*, as it is possible that our list of carcinogens is not complete. Thus whilst we wish to be able to prove $\neg\mathrm{append}([], [1, 2], [3])$, we may be undecided about $\neg\mathrm{carcinogen}(\mathrm{chocolate})$, and so the NAF approach is inappropriate here, as whilst $\exists x \neg\mathrm{carcinogen}(x)$ may be true, it is not necessarily true that everything not known to be a carcinogen is known not to be a carcinogen.

Thus we distinguish between completely defined predicates, i.e. those for which we may apply NAF, and incompletely defined predicates, for which some other form of negation will be necessary.

Note that NAF is an implicit form of negation, as the definition relies on the complement of a given definition, rather than an explicitly defined relation. This is not necessarily the only form of negation possible; indeed, another form is needed for incompletely defined predicates. One such form is the Negation as Inconsistency (NAI) principle introduced by Gabbay and Sergot [6], which is an explicit form of negation in that formulae such as $\neg\mathrm{p}(t)$ form part of the program. Thus whilst our information may not be complete, we may know a particular piece of negative information. For example, although the list of carcinogens may be incomplete, it may be known that bananas are not carcinogenic, and so we would wish to state $\neg\mathrm{carcinogen}(\mathrm{bananas})$ as part of the program.

In this paper we mainly restrict our attention to NAF and completely defined predicates. Our aim is to construct a Kripke-like model theory which precisely captures the behaviour of negated atoms, so that if an atom A fails, then we wish for $\neg A$ to be true in the model. Naturally, our approach is inspired by the possible worlds idea of Kripke [16, 17].

It is common in dealing with negation in logic programming to focus attention on a restricted class of programs, such as the locally stratified programs [14], or some similar weak form of stratification ([13] for example). We have no such restriction; programs such as $p \subset \neg p$ are dealt with by our semantics.

2 Worlds and Accessibility

It has been shown in [12] how first-order hereditary Harrop formulae, a fragment of first-order intuitionistic logic, may be considered as a logic programming language, and that they form a generalisation of Horn clauses. An operational notion of consequence was presented, so that if P is a program and G is a goal, then $P \vdash_o G$ if and only if G succeeds given the program P.

In [10], a Kripke-like model theory was constructed for the following set of formulae:

$$D := A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A$$
$$G := A \mid \exists x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

where A stands for an atom.

This fragment may thought of as a restriction of the set of first-order hereditary Harrop formulae, with the restriction being that goals are not allowed to contain universal quantifications. We refer to formulae satisfying the D or G definition above as definite formulae and goal formulae respectively, and denote by $\mathcal{D}$ and $\mathcal{G}$ the set of all definite formulae and the set of all goal formulae respectively. A program is a closed definite formula, and a goal is a closed goal formula.

Let $\mathcal{U}$ be the set of all closed terms (the Herbrand Universe), $\mathcal{H}$ be the set of all closed atomic formulae (the Herbrand Base) and $\mathcal{P}$ be the set of all programs. The an *interpretation* is any function $I : \mathcal{P} \rightarrow \text{powerset}(\mathcal{H})$ such that $\forall P_1, P_2$ with $P_1 \subseteq P_2, I(P_1) \subseteq I(P_2)$.

The worlds in this case are just programs, with the accessibility relation between worlds being set inclusion. A consequence relation $\models\!\!\!\mid$ was given in [10] in which $I, P \models\!\!\!\mid G$ precisely when G is true in the world P. This differs from the standard consequence relation $\models$ in the treatment of implication, as the definition of $\models$ involves quantification over all worlds above the current one, whereas the $\models\!\!\!\mid$ definition looks at only one particular world. This has the advantage that the construction process may be described syntactically, as we only need to consider truth in one world, rather than looking at all reachable worlds. An added advantage is that the $\models\!\!\!\mid$ relation leads to a monotonic operator, which simplifies the proofs, whereas the $\models$ version of the same operator is not monotonic. This makes $\models\!\!\!\mid$ suitable for a generalisation of the usual T^ω process.

This version of the T operator is shown to be monotonic, and so the least fixed point is $T^\omega(I_\perp) = \sqcup_{i=1}^{\infty} T^i(I_\perp)$. It is then shown how $P \vdash_o G$ iff $T^\omega(I_\perp), P \models\!\!\!\mid G$. It is also true that this holds if and only if $T^\omega(I_\perp), P \models G$, [11] i.e. that if we interpret the constructed interpretation directly as a Kripke model (where the valuation is constant, being given by $T^\omega(I_\perp)(P)$), then we get exactly the same consequence relation as we did for $\models\!\!\!\mid$. Thus we may think of $\models\!\!\!\mid$ as an adaptation of $\models$ in our special case, so that whilst the two relations differ on the intermediate steps, they agree on the final result.

We may gain a geometric insight into the structure of the collection of models by viewing it as an inverted cone, with the empty program at the bottom. The cone extends infinitely upwards in an ever-widening way, as every possible extension (of which there are infinitely many) of a world w is reachable from w.

Now in order to incorporate negated atoms as goals, we wish to define the G formulae as follows:

$$G := A \mid \neg A \mid \exists x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

This will require an extension to the relations $\vdash_o$ and $\models\!\models$.

The extension to $\models\!\models$ must satisfy the "never" condition: $I, w \models\!\models \neg A$ iff for every world w' such that $w' \geq w$, A is not true in w'. If the relation $\leq$ between worlds is set inclusion ($\subseteq$), then there is no I,w and A such that $I, w \models\!\models \neg A$, as the world $P \cup \{A\}$ is reachable from P for any A, and we know that $A \in T^{\omega}(I_{\perp})(P \cup \{A\})$ i.e. $T^{\omega}(I_{\perp}), P \cup \{A\} \models\!\models A$ for any atom A.

Thus the model theory of [10] needs to be selectively pruned, so that we may make some worlds unreachable, but allow access to others in that a given world w may only reach worlds which do not extend completely defined predicates of w. Hence, we wish the accessability relation between worlds to be:

$$P \cup \{A\} \text{ is reachable from } P \text{ iff } A \text{ is not completely defined in } P$$

For example, let P be the append program above. As append is completely defined in P, we want the larger program $P \cup \{\text{append}([], [1,2], [3])\}$ not to be reachable from P.

As the notion of completely or incompletely defined is dependent on how much is known about predicates, we consider a program (and thus a world) as a pair $\langle D, N \rangle$, where $N \subseteq \text{names}(D)$ is the set of names of all the completely defined predicates of D.

In what follows we use the convention that programs are sets of C formulae (C for clause formulae), which are defined as follows:

$$C := A \mid \forall x C \mid G \supset A$$

with G formulae defined as above. This allows us to consider a program as a set of clauses without sacrificing any expressive power. We denote head of a clause C by head(C), and the set of all clause heads of D by heads(D).

Thus we arrive at the following definition of a program, i.e. a world:

Definition 1 *Given an atom $A = p(t_1, \ldots, t_n)$, we define name(A) $= p$, and for any formula F, names(F) $= \{$name(A) $\mid A$ appears in $F\}$. A* program *is a pair $\langle D, N \rangle$, where D is a closed definite formula, and $N \subseteq$ names(D).*

The natural partial order for $\mathcal{D} \times \text{powerset}(\text{names}(\mathcal{H}))$ which reflects the above condition on reachability between worlds is as follows:

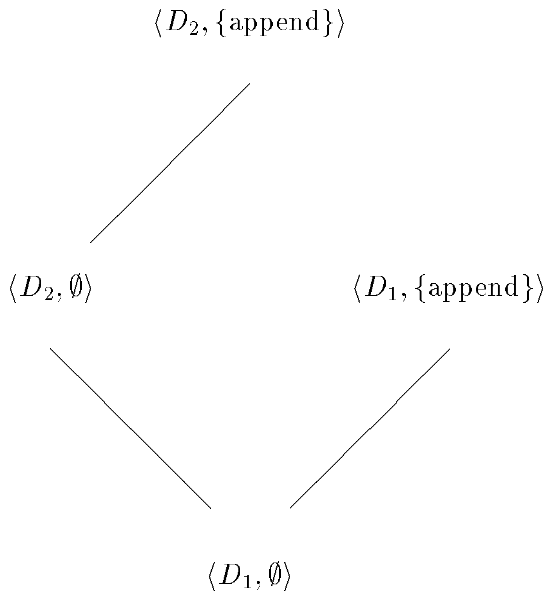

Figure 1: The append program and its possible extensions

Definition 2 *Let D_1, D_2, N_1 and N_2 be such that D_1 and D_2 are definite formulae, and $N_i \subseteq names(D_i)$, i = 1,2. Then $\langle D_1, N_1\rangle \leq \langle D_2, N_2\rangle$ iff*

1. $D_1 \subseteq D_2$
2. $N_1 \subseteq N_2$
3. *for each* $C \in D_2 \backslash D_1$, $name(head(C)) \notin N_1$

For example, let D_1 be the two clauses for append given above, and let $D_2 = D_1 \cup \{\text{append([],[1,2],[3])}\}$. The accessibility relations between the programs $\langle D_i, N\rangle$ where $i = 1, 2$, $N = \emptyset, \{\text{append}\}$ are given in Figure 1.

Note that $\langle D_1, \emptyset\rangle \leq \langle D_2, N\rangle$ but that $\langle D_1, \{\text{append}\}\rangle \not\leq \langle D_2, N\rangle$, where N is either $\emptyset$ or $\{\text{append}\}$.

In this way we need to explicitly know that the larger world is a consistent extension of the smaller one.

3 Extending the Framework

We saw above that an interpretation was defined as any function I mapping definite formulae to sets of closed atoms such that whenever $P_1 \subseteq P_2$ then

$I(P_1) \subseteq I(P_2)$. Thus we may think of an interpretation as an indication of which atoms are true. However, in order to serve as an indication of which atoms are false, we will need more information, as if we merely take as the set of atoms which are false the complement of the set of atoms which are true, then the law of excluded middle holds for every predicate, and so it is difficult to see how this is consistent with a constructivist philosophy. Computability considerations also affect this approach, as the complement of a recursively enumerable set may not be recursively enumerable, and so the complement of the set of successful goals is a difficult thing with which to deal. Another cogent point that has been argued previously ([3, 4, 5, 9]) is that some sort of three-valued logic is useful for programming, so that successful goals may be assigned the value true, failed goals may be assigned the value false, and looping goals can be given an explicit value, i.e. $\perp$, representing "undefined". As our interest is to model the behaviour of the program, it seems natural to allow the possibility that an interpretation I may neither force G to be true nor force G to be false. For example, if p is a looping goal, then we would expect that the corresponding interpretation would neither force p nor force $\neg p$, even if p is a completely defined predicate.

The fact that we are working within an intuitionistic framework means that we may incorporate this "3-valued" behaviour other than by the method of a third explicit truth value $\perp$. The lack of the law of excluded middle means that intuitionistic truth is inherently three valued; the set of true statements together with the set of false statements is not necessarily exhaustive, just as the set of successful goals and the set of goals which (finitely) fail is not necessarily exhaustive. Thus we can capture the three-valued behaviour of programs directly, rather than by the addition of $\perp$ (as is done in [5, 8] for example). As pointed out in [10], the Kripke-like semantics neatly captures this trichotomy; either $A \in I(P)$ or $A \notin I(P)$ but A is true in some world above P or A is false in all worlds.

Thus it seems natural to extend the definition of an interpretation as follows (similar to the notion of 3-valued interpretation of Przymusinki [15]):

Definition 3 *Let P be a program. An* interpretation *is any function $I : \mathcal{P} \rightarrow \mathcal{H} \times \mathcal{H}$ satisfying the following conditions, where $I(P) = \langle S, F \rangle$:*

1. $S \cap F = \emptyset$
2. *for all worlds w_1, w_2 such that $w_1 \leq w_2, I(w_1) = \langle S_1, F_1 \rangle, I(w_2) = \langle S_2, F_2 \rangle$, we have $S_1 \subseteq S_2$ and $F_1 \subseteq F_2$*

Let $I(w) = \langle S, F \rangle$. We define left$(I)(w) = S$ and right$(I)(w) = F$.

We think of left(I) as specifying which atoms are true, and of right(I) as specifying which atoms are false. Thus, we may think of the definition of an interpretation given in [10] as the special case of our definition obtained when right$(I)(w) = \emptyset$ for any world w. The first side condition ensures that no atom is specified as being both true and false, and so this condition ensures that interpretations are internally consistent. The second condition is a generalisation of the previous condition of internal monotonicity. This is justified by the perception that as programs increase, the knowledge contained in the program cannot decrease, and so no extension to a program is allowed to decrease either the set of atoms known to be true or the set of atoms known to be false. Thus we preserve the principle of monotonicity of information.

Note that we do not explicitly require that $S \cup F = \mathcal{H}$. This is possible, of course; indeed, the CWA may be thought of as requiring this to be the case for any interpretation, i.e. that if $\text{left}(I) = S$, then $\text{right}(I) = \mathcal{H} \backslash S$. However, there are many programs for which such an interpretation will be inappropriate. For example, consider the program $\langle p \subset p, \emptyset \rangle$. There are no successful goals, but the set of goals which fail does not include p, and so if we wish for some form of interpretation which precisely matches up with computational behaviour, we cannot insist that $S \cup F = \mathcal{H}$.

The partial order $\sqsubseteq$ on interpretations is extended in the obvious way, as are the operators $\sqcup$ and $\sqcap$.

Definition 4 *Let I_1, I_2 be interpretations, and let $I_1(w) = \langle S_1, F_1 \rangle$ and $I_2(w) = \langle S_2, F_2 \rangle$. Then*

$$
\begin{array}{l}
I_1(w) \leq I_2(w) \textit{ iff } S_1 \subseteq S_2 \textit{ and } F_1 \subseteq F_2 \\
I_1 \sqsubseteq I_2 \textit{ iff } \forall w \in \mathcal{W} \textit{ we have } I_1(w) \leq I_2(w) \\
(I_1 \sqcup I_2)(w) = \langle S_1 \cup S_2, F_1 \cup F_2 \rangle \\
(I_1 \sqcap I_2)(w) = \langle S_1 \cap S_2, F_1 \cap F_2 \rangle
\end{array}
$$

This partial order differs from that of [10] in that whilst there is still one minimal interpretation, i.e. $I_\perp$ where $I_\perp(w) = \langle \emptyset, \emptyset \rangle$ for any world w, there is no unique maximal interpretation, as there was in the earlier case. For example, the two interpretations I^T and I^F are both maximal, where $I^T(w) = \langle \mathcal{H}, \emptyset \rangle$ and $I^F(w) = \langle \emptyset, \mathcal{H} \rangle$. Hence, $I_1 \sqcup I_2$ may not be an interpretation, but no such case will arise from the T^ω construction.

Now we come to the generalisation of the relation $\models\!\!\!\models$ defined in [10]. From now on, we consider goal formulae as any formula G which satisfies

$$G := A \mid \neg A \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid \exists x G \mid D \supset G$$

Given a program $P = \langle D, N \rangle$, an interpretation I and a goal formula G we have

- $I, P \models\!\!\!\models A$ iff $A \in \text{left}(I)(P)$
- $I, P \models\!\!\!\models G_1 \vee G_2$ iff $I, P \models\!\!\!\models G_1$ or $I, P \models\!\!\!\models G_2$
- $I, P \models\!\!\!\models G_1 \wedge G_2$ iff $I, P \models\!\!\!\models G_1$ and $I, P \models\!\!\!\models G_2$
- $I, P \models\!\!\!\models \exists x G$ iff $I, P \models\!\!\!\models G[x/t]$ for some $t \in \mathcal{U}$

- $I, \langle D, N\rangle \models D' \supset G$ iff $I, \langle D \wedge D', N\rangle \models G$ and $\langle D \wedge D', N\rangle \geq \langle D, N\rangle$
- $I, P \models \neg A$ iff $A \in \text{right}(I)(P)$

This is little different from [10] except for the last two cases. In the case of [10] the side condition on implication is vacuously true, as $D \cup \{D'\} \supseteq D$. Here we explicitly require that $\langle D \wedge D', N\rangle$ be reachable from $\langle D, N\rangle$, i.e. that the new world is reachable from the first. This may be thought of as ensuring that the assumption is consistent. There may be weaker restrictions which we may use, but this one seems to be a natural choice.

Note that this definition of an interpretation may be used for the negation of open predicates, in that if we know $\neg p(b)$, then we may capture this information in an interpretation I by ensuring that $p(b) \in \text{right}(I)(w)$. In this way we may incorporate Negation as Inconsistency [6], which may be thought of as an extension to the programs defined above by allowing $\neg A$ as a definite formula. Naturally the construction process described below would need to be extended to cope with this case, but the notion of an interpretation would not need any modification.

As mentioned above, we also need to extend the notion of operational provability so that the relation $\vdash_o$ is defined over $\mathcal{P} \times \mathcal{G}$, where $\mathcal{G}$ is the larger set of goal formulae defined above. We do this by identifying $P \vdash_o \neg A$ with $P \not\vdash_o A$ when name(A) is a completely defined predicate, i.e. for completely defined predicates we identify negation ($P \vdash_o \neg A$) with failure ($P \not\vdash_o A$). The definition of $\vdash_o$ is now as follows:

Let $P = \langle D, N\rangle$ be a program, and let $(D) = \{C' \mid C'$ is a closed instance of a clause $C \in D\}$. Then

- $P \vdash_o A$ iff $A \in (D)$ or $\exists G \supset A \in (D)$ such that $P \vdash_o G$
- $P \vdash_o G_1 \vee G_2$ iff $P \vdash_o G_1$ or $P \vdash_o G_2$
- $P \vdash_o G_1 \wedge G_2$ iff $P \vdash_o G_1$ and $P \vdash_o G_2$
- $P \vdash_o \exists x G$ iff $P \vdash_o G[x/t]$ for some $t \in \mathcal{U}$
- $\langle D, N\rangle \vdash_o D' \supset G$ iff $\langle D \wedge D', N\rangle \vdash_o G$ and names(heads$(D')) \cap N = \emptyset$
- $\langle D, N\rangle \vdash_o \neg A$ iff $\langle D, N\rangle \not\vdash_o A$ and name$(A) \in N$

This is the same as in [10] except for the last two cases. As we may think of the relation $\vdash_o$ given in [10] as the special case of the one above when $N = \emptyset$, this version is exactly the same as the earlier one for goals which contain no negations. The rule for implications given here ensures that only incompletely defined predicates may be extended by the larger program. The case for negation ensures that the NAF rule is only applied to completely defined predicates of P, i.e. those whose names appear in N.

It is important to note that there are some programs P and goals G such that neither $P \vdash_o G$ nor $P \not\vdash_o G$. For example, if P is just $\langle p \supset p, \emptyset\rangle$, then according to the above definition $P \vdash_o p \Leftrightarrow P \vdash_o p$. Thus the definition of $\not\vdash_o$ is not very clear from the above, and so we give a direct definition of the relation $\not\vdash_o \subseteq \mathcal{P} \times \mathcal{G}$ below.

Let $P = \langle D, N \rangle$ be a program. Then we define

- $P \not\vdash_o A$ iff $A \notin (D)$ and $\nexists G \supset A \in (D)$ such that $P \vdash_o G$
- $P \not\vdash_o G_1 \vee G_2$ iff $P \not\vdash_o G_1$ and $P \not\vdash_o G_2$
- $P \not\vdash_o G_1 \wedge G_2$ iff $P \not\vdash_o G_1$ or $P \not\vdash_o G_2$
- $P \not\vdash_o \exists x G$ iff $P \not\vdash_o G[x/t]$ for all $t \in \mathcal{U}$
- $\langle D, N \rangle \not\vdash_o D' \supset G$ iff $\langle D \wedge D', N \rangle \not\vdash_o G$ and names(heads(D')) $\cap N = \emptyset$
- $\langle D, N \rangle \not\vdash_o \neg A$ iff $\langle D, N \rangle \vdash_o A$ and name(A) $\in N$

Note that the cases for negation and implication here are not strict complements of the previous cases. This is really a question of scope; the above definitions are sufficient for our purposes in that NAF is only used for completely defined predicates, and we may only extend programs via extensions to incompletely defined predicates. Weaker restrictions may be found, but it seems that any such treatment will need to deal thoroughly with inconsistencies, we use the simpler definitions here. Also, the above definitions have the desirable property that we may read $P \vdash_o G$ as "G succeeds" and $P \not\vdash_o G$ as "G fails", and it is not intuitively clear what would happen to this property for any weaker cases.

Thus we have given an extension to the notion of interpretation, and of the two consequence relations $\models\!\models$ and $\vdash_o$. We show how we may use this to produce the desired interpretation in the next section.

4 The Construction Process

We proceed in a similar manner to that in [10], i.e. we build ordinal powers of a T operator, and use the union of all such powers to produce the desired interpretation. Before we do so, let us consider the nature of the construction process. The desired process is one that builds upwards, so that as the process goes on, the knowledge we have is always increasing, but never in an inconsistent fashion. We may think of this process as ascending a cone similar to the worlds described above, except that each node is an ordered pair $\langle S, F \rangle$ where S and F are sets of ground atoms with the accessibility relation being pairwise set inclusion. The process begins at the base node and continues upward monotonically, in that when going from node $\langle S_1, F_1 \rangle$ to $\langle S_2, F_2 \rangle$, we have $S_1 \subseteq S_2$ and $F_1 \subseteq F_2$. This means that whenever the construction process places an atom A in right($T^k(I)$)(w), we must be sure that A will not be placed in left($T^j(I)$)(w) for some $j \geq k$. This is related to the problem of disproving a goal G; to prove G it is only necessary to find a proof, whereas to disprove G (i.e. show that G fails) we need to show that there cannot possibly be any proof of G. Because of this requirement, we need to introduce the concept of the *success level* of a goal, and the *failure level* of a goal. These are measures of the length of time it takes to prove or disprove a given goal G.

The failure level of a goal will be more useful to us than the success level, as it will provide us with a way of determining that a given goal G which is not true in a certain interpretation will never be true in any larger interpretation constructed later in the process, and so we can satisfy the "never" condition for intuitionistic negation. We define the success and failure levels below.

Definition 5 *Let $P = \langle D, N \rangle$ be a program. Let $\mathcal{G}$ be the set of all closed goals. We define two mappings* $\mathrm{slevel}_P, \mathrm{flevel}_P : \mathcal{G} \to \mathbb{N} \cup \infty$ *(where $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$) as follows:*

If A is a self-dependent atom in P, then we initially define $\mathrm{slevel}_P(A) := \mathrm{flevel}_P(A) := \infty$. *Next, for any closed goal we define*

$$
\begin{array}{lll}
\mathrm{slevel}_P(A) & := 0 \text{ if } A \in (D) \\
 & := 1 + \min\{\mathrm{slevel}_P(G) \mid G \supset A \in (D)\} \\
 & := \infty \text{ otherwise} \\
\mathrm{slevel}_P(G_1 \vee G_2) & := \min(\mathrm{slevel}_P(G_1), \mathrm{slevel}_P(G_2)) \\
\mathrm{slevel}_P(G_1 \wedge G_2) & := \max(\mathrm{slevel}_P(G_1), \mathrm{slevel}_P(G_2)) \\
\mathrm{slevel}_P(\exists x G') & := \min\{\mathrm{slevel}_P(G'[x/t]) \mid t \in \mathcal{U}\} \\
\mathrm{slevel}_P(D' \supset G') & := \mathrm{slevel}_{P \cup D'}(G') \text{ if } \langle D \wedge D', N \rangle \geq \langle D, N \rangle \\
 & := \infty \text{ otherwise} \\
\mathrm{slevel}_P(\neg A) & := \mathrm{flevel}_P(A) \text{ if } \mathrm{name}(A) \in N \\
 & := \infty \text{ otherwise}
\end{array}
$$

$$
\begin{array}{lll}
\mathrm{flevel}_P(A) & := 0 \text{ if } A \notin (D) \text{ and } \nexists G \supset A \in (D) \\
 & := 1 + \max\{\mathrm{flevel}_P(G)) \mid G \supset A \in (D)\} \\
 & := \infty \text{ otherwise} \\
\mathrm{flevel}_P(G_1 \vee G_2) & := \max(\mathrm{flevel}_P(G_1), \mathrm{flevel}_P(G_2)) \\
\mathrm{flevel}_P(G_1 \wedge G_2) & := \min(\mathrm{flevel}_P(G_1), \mathrm{flevel}_P(G_2)) \\
\mathrm{flevel}_P(\exists x G') & := \max\{\mathrm{flevel}_P(G'[x/t]) \mid t \in \mathcal{U}\} \\
\mathrm{flevel}_P(D' \supset G') & := \mathrm{flevel}_{P \cup D'}(G') \text{ if } \langle D \wedge D', N \rangle \geq \langle D, N \rangle \\
 & := 0 \text{ otherwise} \\
\mathrm{flevel}_P(\neg A) & := \mathrm{slevel}_P(A) \text{ if } \mathrm{name}(A) \in N \\
 & := \infty \text{ otherwise}
\end{array}
$$

The reason for the two stages is that if there is a clause $p \subset p \vee q$ in the program where q succeeds, we have that p succeeds, but that p is self-dependent. In this case the self-dependency is redundant as far as $\mathrm{slevel}_P(p)$ is concerned. Hence, whilst the first stage sets $\mathrm{slevel}_P(p)$ to ∞, the second stage makes it finite.

Note that we consider that $\max(n, \infty) = \infty$, and $\min(n, \infty) = n$. The reason for this is the difference between the criteria for proving G and for disproving G. For example, for $G_1 \vee G_2$ to fail, we need both G_1 and G_2 to fail, but for $G_1 \vee G_2$ to succeed, we only need one disjunct to succeed. Hence if G_1 loops, then $\text{slevel}_P(G_1 \vee G_2) = \text{slevel}_P(G_2)$ and $\text{flevel}_P(G_1 \vee G_2) = \infty$. This reflects the fact that any fair computation rule will find that $G_1 \vee G_2$ succeeds if G_2 succeeds, whereas $G_1 \vee G_2$ will never fail if G_1 loops.

Note that if $\text{flevel}_P(G)$ is finite, then G fails. Hence if for a ground atom A all clauses of the form $G \supset A$ in (D) are such that $\text{flevel}_P(G)$ is finite, then we know that A must fail, and so once an interpretation "knows enough" to disprove every such goal G, the next stage will have sufficient knowledge to disprove A.

The possibility that both levels may be infinite is what allows us to deal with programs such as $\langle p \subset \neg p, \{p\} \rangle$ which are not locally stratified [14, 13]. In this case we have $\text{slevel}_P(p) = \text{flevel}_P(p) = \infty$, and similarly for $\neg p$. Hence, we will have $T^\omega(I_\perp), P \not\models p$ and $T^\omega(I_\perp), P \not\models \neg p$, which is to be expected, as both p and $\neg p$ loop. Thus, we need not restrict the class of programs at all.

As our desired interpretation is $T^\omega(I_\perp)$, we are mainly interested in the ordinal powers of T rather than interpretations and fixpoints per se. It is important to note that a consequence of the monotonicity of the operator given in [10] is that if $T^k(I_\perp), w \models G$, then $T^{k+1}(I_\perp), w \models G$. This may be seen by the fact that $I_\perp \sqsubseteq T(I_\perp)$, and so as T is monotonic, we have $T(I_\perp) \sqsubseteq T^2(I_\perp)$ and so on. This is perhaps a more important consequence of the monotonicity of T than the fact that a least fixed point exists. Whilst the fixpoint semantics does display a certain mathematical elegance, if the method used to construct the desired interpretation does not respect provability via $\models$, it is difficult to see what use it would be. It is possible, by some technical tweaks, to view the construction below as a fixpoint construction, but we feel it is more natural without such machinery.

The main difficulty is to know when a goal G cannot succeed. As indicated above, we will use $\text{flevel}_P(G)$ as a measure of this. However, it does not seem natural to incorporate this notion into the fixpoint framework, as, as shown below, the construction of $\text{right}(T^k(I))(w)$ depends upon k, and so we cannot really consider T as an operator in its own right. Hence, we pursue a slightly different method of construction.

We define the ordinal powers of T by the following, where $P = \langle D, N \rangle$:

$$\begin{aligned}
\text{left}(T^0(I))(w) &= \{A \mid A \in (D)\} \cup \text{left}(I)(P) \\
\text{left}(T^{k+1}(I))(w) &= \{A \mid \exists G \supset A \in (D) \text{ such that } T^k(I), w \models G\} \\
&\quad \cup \text{left}(T^k(I))(w) \\
\text{right}(T^0(I))(w) &= \{A \mid \text{name}(A) \in N \text{ and } A \notin (D) \text{and } \nexists G \supset A \in (D)\} \\
&\quad \cup \text{right}(I)(P) \\
\text{right}(T^{k+1}(I))(w) &= \{A \mid \text{name}(A) \in N \text{ and } \forall G \supset A \in (D), T^k(I), w \not\models G \\
&\quad \text{and } \text{flevel}_P(G) \leq k\} \cup \text{right}(T^k(I))(w)
\end{aligned}$$

$$T^\omega(I)(w) = \langle \bigcup_{i=1}^{\infty} \text{left}(T^i(I))(w), \bigcup_{i=1}^{\infty} \text{right}(T^i(I))(w) \rangle$$

The critical part of the definition is the condition for right$(T^{k+1}(I))$ that flevel$_P(G) \leq k$, as it is this condition which limits how far the interpretation $T^{k+1}(I)$ may "see".

For an example of how this process works, consider the program below.

$$\text{even}(0) \wedge \forall x \ \text{even}(s(x)) \subset \neg\text{even}(x)$$

Let $P = \langle D, \{even\} \rangle$ where D is the code in the program above. Then we have

$$\begin{aligned}
T^0(I_\perp)(P) &= \langle \{\text{even}(0)\}, \emptyset \rangle \\
T^1(I_\perp)(P) &= \langle \{\text{even}(0)\}, \{\text{even}(s(0))\} \rangle \\
T^2(I_\perp)(P) &= \langle \{\text{even}(0), \text{even}(s^2(0))\}, \{\text{even}(s(0))\} \rangle \\
T^3(I_\perp)(P) &= \langle \{\text{even}(0), \text{even}(s^2(0))\}, \{\text{even}(s(0)), \text{even}(s^3(0))\} \rangle \\
&\ldots
\end{aligned}$$

The key step in this process is getting from $T^0(I_\perp)(P)$ to $T^1(I_\perp)(P)$. It is clear that slevel$_P$(even(0)) = 0, and so flevel$_P(\neg$even(0)) = 0. So as $T^0(I_\perp), P \not\models \neg\text{even}(0)$, from the definition of $T^1(I_\perp)(P)$ it is easily seen that even$(s(0)) \in$ right$(T^1(I_\perp))(P)$.

In this way we may think of the powers of T as using the program to define an increasing sequence of interpretations which is used to model the behaviour of the program.

5 Properties of the Construction Process

It is obvious from the definition of $T^k(I)$ that $T^k(I) \sqsubseteq T^{k+1}(I)$, and so it is easy to show that the powers of T respect $\models$. However, we can say more about the precise power of T at which the "cut off" point for a given goal is reached, i.e. the value of j such that if $T^j(I), P \not\models G$, then $T^\omega(I), P \not\models G$.

The relationship between the powers of T and the success or failure level of a goal G is given in the lemma below. The proof of this and other results quoted here may be found in [7].

Lemma 5.1 *Let P be a program, and let G be a goal. Then for any $i \geq 0$*

1. slevel$_P(G) \leq i \Leftrightarrow T^i(I_\perp), P \models G$
2. flevel$_P(G) \leq i \Rightarrow T^i(I_\perp), P \not\models G$

Note that the converse to 2 is not true. For a counterexample, consider an atom p with slevel$_P(p)$ = flevel$_P(p) = \infty$. Then from 1 we have that

$T^i(I_\perp), P \not\models p$, as there can be no i such that $p \in \mathrm{left}(T^i(I_\perp))(P)$. We will have more to say about this case later.

Note also that a consequence of 1 is that if $\mathrm{slevel}_P(G) = k$, we have $T^k(I_\perp), P \models G$ and that $T^{k-1}(I_\perp), P \not\models G$. Thus $\mathrm{slevel}_P(G)$ gives a precise value of the power of T for which G first becomes true in the Kripke-like model, as remarked earlier.

A similar result holds for $\vdash_o$, relating $\mathrm{slevel}_P(G)$ and $\mathrm{flevel}_P(G)$ to $P \vdash_o G$ and $P \not\vdash_o G$.

Lemma 5.2 *Let P be a program, and let G be a goal. Then*

1. $P \vdash_o G \Leftrightarrow \mathrm{slevel}_P(G)$ *is finite.*
2. $P \not\vdash_o G \Leftrightarrow \mathrm{flevel}_P(G)$ *is finite.*

The main result about the equivalence of $\vdash_o$ and $\models$ now follows immediately, and is given below. Note that we are dealing with arbitrary goals, which includes goals G for which $\mathrm{slevel}_P(G) = \mathrm{flevel}_P(G) = \infty$.

Theorem 5.3 *Let P be a program, and let G be a goal. Then*

1. $P \vdash_o G \Leftrightarrow T^\omega(I_\perp), P \models G$
2. $P \not\vdash_o G \Rightarrow T^\omega(I_\perp), P \not\models G$

As noted above, the converse to 2 cannot hold, as there are goals G and programs P such that $T^\omega(I_\perp), P \not\models G$, but that $P \not\vdash_o G$ does not hold. This is due to the fact that $\vdash_o$ is defined in a top-down fashion, whilst $T^\omega(I_\perp)(P)$ is constructed in a bottom-up way. Thus we have that for any program P and for any goal G either $T^\omega(I_\perp), P \models G$ or $T^\omega(I_\perp), P \not\models G$, but that a similar property does not hold for $\vdash_o$, as loops behave differently in each case. For example, given the program $P = \langle p \supset p, \emptyset\rangle$ and the goal p, we have that $T^\omega(I_\perp)(P) = \langle\emptyset, \emptyset\rangle$, and so $T^\omega(I_\perp), P \not\models p$. However, neither $P \vdash_o p$ nor $P \not\vdash_o p$, as the goal p will loop forever.

Thus, the theorem above is the best possible result for an arbitrary closed goal G. It is obvious from lemma 5.2 and lemma 5.1 that if $\mathrm{flevel}_P(G)$ is finite, then $P \not\vdash_o G \Leftrightarrow T^\omega(I_\perp), P \not\models G$.

6 Comparative Results

Although we have been concerned with the powers of T themselves, rather than interpretations in general and fixpoints, it may be instructive to investigate the precise nature of the relationship between our presentation and the traditional approach. It is possible to show that if we define our version of the T operator "in isolation", i.e. $T(I)$ is defined in terms of I, as is traditionally done, then the T operator so defined is not monotonic, but that $T^\omega(I_\perp)$ is nevertheless a fixpoint of T.

This "isolated" definition of the T operator is not a definition in which we are particularly interested; it is included for purposes of comparison.

In order to perform the construction process, we need to associate a non-negative integer k with an interpretation I. This will play the role that the current power of T in the earlier definition when dealing with $\text{flevel}_P(G)$, so that given a goal G for which $\text{flevel}_P(G) \leq k$ and $I, P \not\models G$, for any interpretation J such that $I \sqsubseteq J$ we cannot have $J, P \models G$. Hence we also need to impose the extra condition that if k_i is associated with I_i, then $I_1 \sqsubseteq I_2$ implies that $k_1 \leq k_2$. Now if k is associated with I, then we associate $k+1$ with $T(I)$, which is defined as follows, where $w = \langle D, N\rangle$:

$$\begin{aligned}\text{left}(T(I))(w) &= \{A \mid A \in (D) \text{ or } \exists G \supset A \in (D) \text{ such that } I, w \models G\}\\ \text{right}(T(I))(w) &= \{A \mid \text{name}(A) \in N \text{ and } A \notin (D) \text{ and } \forall G \supset A \in (D),\\ &\qquad I, w \not\models G \text{ and } \text{flevel}_P(G) \leq k\}\end{aligned}$$

The powers of T are as expected, i.e. we associate 0 with $T(I_\perp)$, 1 with $T^2(I_\perp)$ and so on, and that $T^\omega(I_\perp) = \bigsqcup_{i=1}^{\infty} T^i(I_\perp)$. It is obvious that both this definition and the previous one define the same interpretation $T^\omega(I_\perp)$. Note that in our earlier definition, we had $T^0(I_\perp), P \models A$ iff $A \in (D)$. Here we have $T(I_\perp), P \models A$ iff $A \in (D)$. No confusion should arise from this, as we are mainly concerned with $T^\omega(I_\perp)$.

Note that this T operator is not monotonic. For a counterexample, let $P = \langle\{p, q \supset r\}, \{r\}\rangle$, and $I_1(P) = \langle\{p\}, \emptyset\rangle$, $I_2(P) = \langle\{p, q\}, \emptyset\rangle$. We associate 0 with I_1 and 1 with I_2. Obviously $I_1 \sqsubseteq I_2$. Now $T(I_1)(P) = \langle\{p\}, \{r\}\rangle$, as $r \in \{r\}$, $I_1, P \not\models q$ and $\text{flevel}_P(q) = 0$. However, $T(I_2)(P) = \langle\{p, r\}, \emptyset\rangle$, as $q \supset r \in (D)$ and $I_2, P \models q$. Hence, we have that $T(I_1) \not\sqsubseteq T(I_2)$.

The problem is that I_2 above forces q to be true, when in fact $P \not\vdash_o q$. Hence, we cannot consider arbitrary interpretations, but only those which make "computational" sense, in that they match up with the operational notion of success. Naturally the values of k_1 and k_2 are crucial as well. In the light of these considerations, it seems best to define the powers of T without reference to arbitrary interpretations, thus ensuring compatibility with $\vdash_o$.

Another reason not to use the isolated definition of the powers of T is that $\text{flevel}_P(G)$ is heavily dependent on the program, whereas an arbitrary interpretation I need not necessarily lead to the same consequences of the program as the relation $\vdash_o$. However, the T operator itself is heavily dependent on the program, and so we have merely cut out the middle step by defining the powers of T directly. The only thing we lose by this direct process is the ability to think of T as an operator on arbitrary interpretations, and so produce other constructions based on it. Traditionally the only other construction to gain much interest has been to produce the greatest fixpoint of T in order to deal with NAF. However, we have seen that the constructivist approach necessitates a more general notion of interpretation, which allows use to directly incorporate similar semantic properties to those of the greatest fixpoint of T. Thus we capture the same descriptive power as the least fixpoint/greatest fixpoint approach, but in a more concrete way, and for a more general class of programs.

Thus the lack of monotonicity of the isolated form of the T operator is not of great concern. Indeed, we can still show that $T^\omega(I_\perp)$ is the least fixpoint of T, which is the usual reason for desiring T to be monotonic in the first place.

Theorem 6.1 *Let T be defined as above, and let*

$$T^\omega(I_\perp) = \bigsqcup_{i=1}^{\infty} T^i(I_\perp) = T^1(I_\perp) \sqcup T^2(I_\perp) \sqcup T^3(I_\perp) \sqcup \ldots$$

Then $T^\omega(I_\perp)$ is the least fixpoint of T, i.e. $T(T^\omega(I_\perp)) = T^\omega(I_\perp)$, and for any other fixpoint J of T, $T^\omega(I_\perp) \sqsubseteq J$.

Thus our latter definition is indeed the least fixpoint of T, even though T is not monotonic. Naturally this takes more work than would have been necessary for a monotonic T operator. Although we prefer to think of the powers of T without reference to arbitrary interpretations, this result may be seen as evidence that our approach is in keeping with the flavour of the traditional methods.

7 Conclusions and Further Work

We have given an extension to a fragment of first-order hereditary Harrop formulae so that negated atoms may be used as goals, in a manner which does not require any restriction to be placed on the class of programs. This lack of a restriction together with the first-order hereditary Harrop formula framework means that our model theory is valid for a wide class of programs. Our approach also incorporates greater flexibility in the application of the NAF rule than in the traditional case, as it is possible, but not necessary, that $N = \text{names}(D)$, which is precisely the case when the CWA holds, i.e. that every predicate is completely defined.

Our framework can also incorporate the negation of incompletely defined predicates to some extent by allowing negated atoms as definite formulae, so that the definitions of D and G formulae become

$$\begin{aligned} D &:= A \mid \neg A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A \\ G &:= A \mid \neg A \mid \exists x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G \end{aligned}$$

We may then extend the definition of $\vdash_o$ so that the case for $\neg A$ is now

$$\langle D, N \rangle \vdash_o \neg A \text{ iff } \neg A \in [D] \text{ or } \text{name}(A) \in N \text{ and } P \not\vdash_o A$$

Note that the relation $\models\!\models$ needs no extension.

Such a class of programs naturally involves questions of consistency, which are beyond the scope of this paper. There is however an obvious class of consistent programs obtained from the previous class of programs (i.e. those for which $\neg A$ is not allowed as a definite formulae) by allowing memoisation to take place. In other words, if $P \vdash_o \neg A$, we may wish to store this information so that if the same goal is asked subsequently, we need not recompute the answer. As discussed in [10], this involves adding successful goals to the program - in this case $\neg A$. The extended program must be consistent as the only way for $\neg A$ to succeed is for A to fail. Note that if P is a progam to which we have memoised $\neg A$, then the goal $A \supset G$ does not succeed, as $\langle P \cup \{A\}, N\rangle \not\geq \langle P, N\rangle$. Thus we may extend completly defined predicates *negatively*, but not positively.

In this way our framework not only covers a very wide class of programs which use NAF, but also may be used for a limited class of programs in which negated atoms may sometimes be considered as definite formulae.

The Kripke-like model naturally captures a notion of program development, and so it may provide a useful framework for formal development of logic progams. Another possibility for further work is the extension of G formulae to allow universal quantifiers. The problem here is mainly with the operational concepts than the model theory, as the relation $\models\!\!\!\models$ may easily be extended via

$$I, w \models\!\!\!\models \forall x G \text{ iff } I, w \models\!\!\!\models G[x/t] \text{ for every } t \in \mathcal{U}$$

The problem is to then find the appropriate extension to $\vdash_o$.

Acknowledgements

My thanks go to Dale Miller for introducing me to the problem, for many enlightening discussions, and for useful suggestions for transforming raw ideas into polished argument. A debt of gratitude is owed also to my supervisor, Don Sannella, who, as always, has been very helpful, and a constant source of inspiration, encouragement and good advice. Comments from David Pym, Colin Stirling, Robin Milner and some anonymous referees have also been instructive.

References

[1] K. Clark, Negation as Failure, in *Logic and Databases*, H. Gallaire and J. Minker (eds.), Plenum Press, 1978.

[2] M.H. van Emden and R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Journal of the Association for Computing Machinery* 23:4: 733-742, October, 1976.

[3] M. Fitting, A Kripke-Kleene Semantics for Logic Programming, *Journal of Logic Programming* 2:295-312, 1985.

[4] M. Fitting, Partial Models and Logic Programming, *Theoretical Computer Science* 48:229-255, 1986.

[5] M. Fitting and M. Ben-Jacob, Stratified and Three-valued Logic Programming Semantics, *Proceedings of the 5th International Conference and Symposium on Logic Programming* 1054-1069, August, 1988.

[6] D. Gabbay and M.J. Sergot, Negation as Inconsistency.I, *Journal of Logic Programming* 3:1-35, 1986.

[7] J. Harland, *On Hereditary Harrop Formulae as a Basis for Logic Programming*, Ph.D. thesis, University of Edinburgh, forthcoming.

[8] K. Kunen, Negation in Logic Programming, *Journal of Logic Programming* 3:1-35, 1986.

[9] J-L. Lassez and M.J. Maher, Optimal Fixedpoints of Logic Programs, *Theoretical Computer Science* 39, 1985.

[10] D.A. Miller, A Logical Analysis of Modules in Logic Programming, *Journal of Logic Programming* 6:79-108, 1989.

[11] D.A. Miller, private communication, January, 1988.

[12] D.A. Miller, G. Nadathur and A. Scedrov, Hereditary Harrop Formulas and Uniform Proof Systems, *Proceedings of the Second Annual Symposium on Logic in Computer Science*, 98-105, Ithaca, June, 1987.

[13] H. Przymusinska and T. Przymusinski, Weakly Perfect Semantics for Logic Programs, *Proceedings of the 5th International Conference and Symposium on Logic Programming* 1106-1121, August, 1988.

[14] T. Przymusinski, On the Declarative Semantics of Deductive Databases and Logic Programs, *Foundations of Deductive Databases and Logic Programming*, (ed. J. Minker) 193-216, Morgan Kaufmann, 1988.

[15] T. Przymusinski, Non-monotonic Formalisms and Logic Programming, *Proceedings of the 6th International Conference on Logic Programming* 655-674, June, 1989.

[16] C. Smorynski, Applications of Kripke Models, Chapter V in [17].

[17] A.S. Troelstra, *Metamathematical Investigations of Intuititionistic Arithmetic and Analysis*, Lecture Notes in Mathematics 344, Springer-Verlag, Berlin, 1973.