Theoretical Computer Science 65 (1989) 343-371
North-Holland

A SOUND AND COMPLETE SEMANTICS FOR A VERSION OF NEGATION AS FAILURE

J.C. SHEPHERDSON

Mathematics Department, University of Bristol, University Walk, Bristol, U.K. BS8 1TW

Communicated by G. Jäger
Received May 1988

Abstract. Negation as failure is sound both for the closed world assumption and the completed database or completion, *comp*(P) of a program P. In general it is not complete for either of these declarative semantics. Indeed there can be no semantics for which it is both sound and complete, for all programs and queries, because non-ground negative literals cannot be dealt with, and cause floundering. By extending the negation as failure rule we exclude floundering and we give a semantics $\bar{T}_\omega(P)$ for which the extended rule is both sound and complete. $\bar{T}_\omega(P)$ is a weak version of *comp*(P) based on an iterative construction. We show that the soundness and completeness results still hold if the classical consequence relation $\vdash$ is replaced by a weaker relation $\vdash_{3l}$ which is sound for both 3-valued logic and intuitionistic logic.

Contents

1. Introduction

By negation as failure we mean, in the narrow sense (NF) the rule that, for ground atoms A,

the goal $\neg A$ *succeeds if* A *fails,*

the goal $\neg A$ *fails if* A *succeeds,*

and in the broad sense the SLDNF-resolution procedure of [13] obtained by adding this inference rule to the SLD-resolution used for Horn clause logic programming. We shall use the words program and goal to mean *normal program* and *normal goal* in the sense of [13], i.e. a program is a finite set of program clauses $A \leftarrow L_1, \ldots, L_n$

and a goal is a clause $\leftarrow L_1, \ldots, L_n$, where A is an atom and $L_1, \ldots, L_n$ are positive or negative literals. We also use the word *query* to denote the negation of a goal, and we say the query $?\text{-}L_1, \ldots, L_n$ succeeds from the program P if $P \cup \{\leftarrow L_1, \ldots, L_n\}$ has an SLDNF-refutation, and that $?\text{-}L_1, \ldots, L_n$ fails from P if $P \cup \{\leftarrow L_1, \ldots, L_n\}$ has a finitely failed SLDNF-tree.

Clark [6] showed that SLDNF-resolution was *sound* for *comp*(P), both for *success*:

If $?\text{-}Q$ *succeeds from* P *with answer* θ *then* $comp(P) \vdash Q\theta$

and for *failure*:

If $?\text{-}Q$ *fails from* P *then* $comp(P) \vdash \neg Q$.

For *comp*(P) to be considered a satisfactory declarative semantics for SLDNF-resolution we would also require it to be *complete* for *success*:

If $comp(P) \vdash Q\theta$ *then* $?\text{-}Q$ *succeeds with answer including* θ

(i.e. with answer θ' such that there exists a substitution φ such that $\theta = \theta'\varphi$), and perhaps for *failure* as well:

If $comp(P) \vdash \neg Q$ *then* $?\text{-}Q$ *fails.*

The fact that SLDNF-resolution is also sound for the closed world assumption *CWA*(P) [18], which can be very different from *comp*(P), and that the soundness for *comp*(P) holds when $\vdash$ is replaced by $\vdash_1$, the intuitionistic derivability relation, [19], or by $\models_3$ [10], the consequence relation of 3-valued logic, lead us to expect that SLDNF-resolution will be incomplete for *comp*(P), and that is usually the case.

Indeed the fact that negation as failure cannot sensibly be applied to non-ground negative literals means that some queries flounder, i.e. give rise to goals containing only non-ground negative literals. Thus for the program $p(x) \leftarrow \neg q(x)$ the query $?\text{-}p(a)$ succeeds but the query $?\text{-}p(x)$ flounders. This means that SLDNF-resolution cannot be both sound and complete for any semantics (or default operator in the sense of Jäger [9]) $H(P)$, i.e. we cannot have, for all queries Q,

$$?\text{-}Q \text{ succeeds with answer including } \theta \text{ iff } H(P) \vdash Q\theta \qquad (\star)$$

because for the above program the query $?\text{-}p(x)$ should succeed with answer including $\{x/a\}$. Whether a query flounders is recursively undecidable [2], so the usual way of dealing with flounders is by imposing strong conditions on the program and the query which prevent floundering. If the *query is allowed*, i.e. every variable in it occurs in a positive literal of it, and each program clause is *allowed* i.e. every variable in it occurs in a positive literal of its body then [13, Proposition 15.1] the query cannot flounder. For programs and queries which are both allowed, there are some classes of program for which SLDNF-resolution has been proved to be complete for *comp*(P), viz definite, hierarchical [13, Theorems 16.1, 16.2, 16.3], finite tree property [19, Theorems 5.5, 5.6, 5.7], and semi-strict programs which are strict with respect to the query [11, 4]. These last results are very illuminating and the conditions imposed are so natural that one feels they may be close to best possible.

Since SLDNF-resolution is not in general complete for $comp(P)$ it is of at least technical interest to try and find an alternative semantics $H(P)$ for which it is both sound and complete, i.e. for which (*) above is true. As noted above we cannot hope to achieve this for all queries because of the possibility of floundering. Rather than impose the blanket condition of allowedness, which is very stringent and excludes many common Prolog constructs {such as the definition of equality ($equal(X, X)$) and both clauses in the standard definition of $member(X, L)$}, we extend the use of NF. This is done by allowing it to be applied to non-ground literals by means of a preliminary substitution, i.e.

If $A\theta$ fails then $\neg A$ succeeds with answer θ.

This is justified by that fact that $\neg\exists A\theta \Rightarrow \forall\neg A\theta$ is logically valid. In fact if SLDNF-resolution is sound for $H(P)$ for both success and failure, then it will remain sound if NF is extended in this way, since if $A\theta$ fails then $H(P) \vdash \neg A\theta$ so it is sound for $\neg A$ to succeed with answer θ, and to obtain completeness, $\neg A$ must succeed with answer including θ. We call the resulting procedure *SLDNFS-resolution* (SLD-resolution with Negation as Failure with Substitution). We define it more precisely in Section 2 and show (Theorem 2.2) that it has the same effect as SLDNF-resolution for queries which do not flounder under the latter. So for such queries, in particular for allowed queries and programs, the completeness results obtained in Section 3 for SLDNF-resolution give corresponding results for SLDNF-resolution.

In Section 3 we define a semantics $\bar{T}_\omega(P)$ for which SLDNFS-resolution is sound for both success and failure, complete for success, and complete for failure for positive queries. The definition is by an iterative construction, starting with $comp(\bar{P})$, where $\bar{P}$ is a definite Horn clause program obtained from P by replacing occurrences of a predicate symbol $\neg p$ by a new predicate symbol $\bar{p}$ and then successively adding formulae which identify $\bar{p}(t_1, \ldots, t_n)$ with $\neg p(t_1, \ldots, t_n)$ in just those cases which correspond to the use of NF. In Section 4 we show that $\bar{T}_\omega(P)$ has a least Herbrand model obtainable as a least fixed point model, that this is a generic model and hence that $CWA(\bar{T}_\omega(P))$ is consistent. In Section 5 we observe that these results also hold if the derivability relation $\vdash$ is replaced by a weaker one, $\vdash_{3I}$ defined by axioms and rules of inference which are valid both in intuitionistic logic and classical 3-valued logic. Theorems 5.4 and 5.6 imply the results of [19, 10] on the soundness of SLDNF-resolution for $comp(P)$ with respect to $\vdash_I$ and $\models_3$. In Section 6 we consider variants of $\bar{T}_\omega$ and relate them to versions of SLDNF-resolution. Section 7 contains some concluding remarks.

2. SLDNFS-resolution

SLDNFS-resolution is obtained from SLDNF-resolution by extending the NF rule as described in Section 1. This is done by allowing the selection of a non-ground

negative literal $\neg A$ in a goal $\leftarrow L_1, \ldots, L_{i-1}, \neg A, L_i, \ldots, L_n$ if there exists θ such that the goal $\leftarrow A\theta$ fails, to obtain the new goal $\leftarrow (L_1, \ldots, L_{i-1}, L_i, \ldots, L_n)\theta$. This is only to be allowed when constructing success branches; the rule for failing $\neg A$ is the same as before; for *ground* A,

if $\leftarrow A$ *succeeds then* $\leftarrow \neg A$ *fails.*

We make this precise by modifying appropriately the definitions given for SLDNF-resolution in [13, pp. 85–88].

Definition. Let P be a normal program, G a normal goal and $k = 0, 1, 2, \ldots$. An *SLDNFS-refutation of rank k of* $P \cup \{G\}$ consists of a sequence $G_0 = G, G_1, \ldots, G_n = \square$ of normal goals, a sequence $C_1, \ldots, C_n$ of variants of program clauses of P or negative literals, and a sequence $\theta_1, \ldots, \theta_n$ of substitutions, such that, for each i, either

(i) G_{i+1} is derived from G_i and C_{i+1} using θ_{i+1}, *or*

(ii) G_i is $\leftarrow L_1, \ldots, L_m, \ldots, L_p$, the selected literal L_m in G_i is a negative literal $\neg A_m$ and there is a finitely failed SLDNFS-tree of rank $<k$ for $P \cup \{\leftarrow A_m\theta_{i+1}\}$. In this case G_{i+1} is $\leftarrow (L_1, \ldots, L_{m-1}, L_{m+1}, \ldots, L_p)\theta_{i+1}$ and C_{i+1} is $\neg A_m$.

A *finitely failed SLDNFS-tree of rank k for* $P \cup \{G\}$ is a tree satisfying the following:

(a) The tree is finite and each node of the tree is a non-empty normal goal.

(b) The root node is G.

(c) Let $\leftarrow L_1, \ldots, L_m, \ldots, L_p$ be a node in the tree and suppose that L_m is selected. Then *either*

(i) L_m is an atom and for each variant $A \leftarrow M_1, \ldots, M_q$ of a program clause (chosen so as to have no variables in common with $L_1, \ldots, L_m, \ldots, L_p$), if L_m and A are unifiable with mgu θ, the node has a child

$$\leftarrow (L_1, \ldots, L_{m-1}, M_1, \ldots, M_q, L_{m+1}, \ldots, L_p)\theta;$$

if there are no such children the node has no successors i.e. is a leaf node; *or*

(ii) L_m is a ground negative literal $\neg A_m$ and there is a finitely failed SLDNFS-tree of rank $<k$ for $P \cup \{\leftarrow A_m\}$, in which case the only child is $\leftarrow L_1, \ldots, L_{m-1}, L_{m+1}, \ldots, L_p$; *or*

(iii) L_m is a ground negative literal $\neg A_m$ and there is an SLDNFS-refutation of rank $<k$ of $P \cup \{\leftarrow A_m\}$, in which case the node is a leaf node.

Note that the case $k = 0$ is included in these definitions, so in proofs by induction on k there is no need to treat it separately. We have included (c)(ii) for comparison with [13], but if, as here, we are only interested in finitely failed trees, it can be omitted; there is no need to make steps of this kind which simply delete a literal L, because if the resulting tree finitely fails so does the tree with the literal L added to all subsequent goals.

SLDNFS-resolution is not feasibly implementable because one has to consider all possible substitutions to apply to the negative literals. However the same is

already true of SLDNF-resolution since the presence of NF steps means that one loses the result which holds for SLD-resolution that all computation rules (rules for selecting literals) are equivalent, and although there are maximal rules [18, 19] there may be no recursive maximal rules [21], so one has to search through all possible derivation trees.

Two other extensions of SLDNF-resolution have recently been proposed. The SLD-CNF resolution (SLD resolution with Constructive Negation as Failure), of Chan [5], returns as answers to ?-$\neg A$ the negation of the answers to ?-A. Thus if the answers to the latter are $x = a$, $x = b$, the answer to the former is $x \neq a \wedge x \neq b$. This is sound for *comp*(P) but is not complete because it cannot proceed if the computation of ?-A does not terminate, or if it produces infinitely many answers. It is not comparable with SLDNFS-resolution; there are examples where SLD-CNF resolution produces an answer but SLDNFS-resolution does not, and vice versa. Chan shows how to implement SLD-CNF resolution, but just as for SLDNF-resolution one would have to consider all computation rules.

Przymusinski [16] defines a notion of SLD-resolution (Linear Resolution with Selected Function for Stratified programs). This is defined only for stratified programs. It is not even theoretically implementable because it calls for a query to fail not only if it fails finitely, but also if all branches of its derivation tree are infinite, which in general cannot be determined. He shows that it is sound and complete for stratified programs and non-floundering queries, for the perfect model semantics. But it is not sound for *comp*(P) because for the program $p \leftarrow p$ it fails ?-p.

A further extension of SLDNFS-resolution (which is sound for *comp*(P)) is considered in Section 6 below. It extends the negation as failure rule by failing $\neg A$ if A succeeds with answer the identity.

2.1. Lemma. *$P \cup \{G\}$ cannot have both an SLDNFS-refutation and a finitely failed SLDNFS-tree.*

Proof. Either like the corresponding result for SLDNF-resolution [18, Theorem 4] by induction on rank and on the length of the refutation, or as an immediate consequence of Lemma 3.1 below and the consistency result implied by Lemma 3.3(b) (Lemma 2.1 is not used in these proofs of Lemmas 3.1, 3.3). □

The effect of extending SLDNF-resolution to SLDNFS-resolution is essentially to allow some queries to succeed which previously floundered. To make this precise we must generalise the notion of flounder to include flounders occurring in the subtrees of selected ground negative literals. Informally we may define a *generalised flounder* of $P \cup \{G\}$ for SLDNF-resolution by a recursive definition like that of an SLDNF-refutation of $P \cup \{G\}$ except that instead of the final goal G_n being empty it is *either*

(a) a flounder, i.e. a goal containing non-ground negative literals and nothing else, *or*

(b) its selected literal is a ground negative literal $\neg A$ such that $P \cup \{\leftarrow A\}$ has a generalised flounder.

Formally, the definition of

- *a generalised flounder* $P \cup \{G\}$ *of rank* 0 is like the definition of an SLDNF-refutation of rank 0 in [13, p. 85] except that the final goal G_n is not the empty goal but is a flounder,
- *a generalised flounder of* $P \cup \{G\}$ *of rank* $k+1$ is like the definition of an SLDNF-refutation of rank $k+1$ [14, pp. 85, 86] except that the final goal G_n is either a flounder, or a ground literal $\neg A$ is selected such that $P \cup \{\leftarrow A\}$ has a generalised flounder of rank k.

{This is similar to the definition of *eventual floundering* in Reynolds [17], except that a fixed selection rule is used there.}

2.2. Theorem. *If $P \cup \{G\}$ has an SLDNFS-refutation of rank k with answer θ then it has either an SLDNF-refutation of rank k with answer θ or a generalised flounder of rank k. If $P \cup \{G\}$ has a finitely failed SLDNFS-tree of rank k then either it has a finitely failed SLDNF-tree of rank k or a generalised flounder of rank k.*

Proof. By induction on k.

($k=0$): SLDNF is the same as SLDNFS.

(*Induction step*): Suppose we have an SLDNFS-refutation of $P \cup \{G\}$ of rank $k+1$. The proof will be by induction on the length n of this. If the length is 0 then G is the empty goal and the refutation is an SLDNF-refutation. If the length is $n+1$ use the switching lemma for SLDNFS-refutation [19, Lemma 7.2] to choose a positive or ground negative literal of G first if there is one (if such a literal is present it must be chosen at some stage because all literals are eventually removed). If there is not one then G is a flounder so we have a generalised flounder of $P \cup \{G\}$ of rank 0. If we choose a positive literal first, the result follows by the induction hypothesis (on n). If we choose a ground negative literal $\neg A$ first then, by the induction hypothesis on k, either $P \cup \{\leftarrow A\}$ has a finitely failed SLDNF tree of rank k or a generalised flounder or rank k. In the first case, by the induction hypothesis on n, $P \cup \{G\}$ has either an SLDNF-refutation or generalised flounder of rank $k+1$. In the second case we get at once a generalised flounder of rank $k+1$ for $P \cup \{G\}$.

Now suppose we have a finitely failed SLDNFS-tree for $P \cup \{G\}$ of rank $k+1$. The proof will be by induction on the height h of this. If $h=1$ then G is a leaf. If the selected literal of G is a positive and no clause of P unifies with it then we have a finitely failed SLDNF-tree of rank 0. If the selected atom of G is a ground negative literal $\neg A$ then by the induction hypothesis on k there is either an SLDNF-refutation of $P \cup \{\leftarrow A\}$ or a generalised flounder of this of rank k. In the first case we get a finitely failed SLDNF-tree of rank $k+1$ for $P \cup \{G\}$, in the second a generalised flounder of rank $k+1$.

For height $h+1$, if the finite selected literal is positive then all children have finitely failed SLDNFS-trees of rank $k+1$, so by the induction hypothesis on h, either one has a generalised flounder of rank $k+1$ and so does G or all have finitely failed SLDNF-trees of rank $k+1$ and so does G. If the first selected literal is $\neg A$, then A must be ground and there is a finitely failed SLDNFS-tree for $P \cup \{\leftarrow A\}$ of rank k, so by the induction hypothesis on k there is either a finitely failed SLDNF-tree or generalised flounder of rank k for $P \cup \{\leftarrow A\}$, so there is either a finitely failed SLDNF-tree or generalised flounder of rank $k+1$ for $P \cup \{G\}$. □

The converse of Theorem 2.2 does not hold; SLDNFS-resolution does not prevent all floundering, e.g. for the program $p(x) \leftarrow p(x)$ the query $?\text{-}\neg p(x)$ flounders under SLDNF-resolution. But although this is technically a flounder the lack of result is not caused by the presence of a variable—the same thing happens if $p(x)$ is replaced by p— but by an infinite branch.

The alternatives in Theorem 2.2 are not mutually exclusive. By Lemma 2.1 $P \cup \{G\}$ cannot have both an SLDNF-refutation and a finitely failed SLDNF-tree but if P is $p, p \leftarrow \neg q(x)$ then $P \cup \{\leftarrow p\}$ has both a refutation and a flounder, and if we form P' by adding the clause $r \leftarrow \neg p$ to P then $P' \cup \{\leftarrow r\}$ has both a finitely failed tree and a generalised flounder.

2.3. Theorem. *SLDNFS-resolution is sound for comp(P) for both success or failure.*

Proof. The argument was given informally in Section 1. For a full proof take the corresponding Theorems 15.6 and 15.4 in [13] for SLDNF-resolution and check that the argument still works for SLDNFS-resolution. Theorem 5.4 below contains a stronger result, that the soundness holds with the classical 2-valued derivability relation replaced by a 3-valued and intuitionistic one. □

3. The default operator $\bar{T}_\omega$

We now proceed to describe the construction of a default operator $\bar{T}_\omega$ giving a semantics for which SLDNFS-resolution is sound and complete. The idea behind the construction is that, until it is selected, a negative literal $\neg p(\underline{t})$ behaves like an instance $\bar{p}(\underline{t})$ of a new predicate $\bar{p}$ unrelated to p. (Here $\underline{t}$ denotes a tuple $t_1, \ldots, t_n$ of terms.) So we start by replacing $\neg p$ by $\bar{p}$. We then arrange for an instance $\bar{p}(\underline{t})$ to be true or false when and only when the falsity or truth of $\neg p(\underline{t})$ is established by an NF step. This is done by an iterative construction paralleling the rank of the SLDNFS-refutation or finitely failed tree. We add $\bar{p}(\underline{t})$ when $\neg p(\underline{t})$ is a consequence of the previous stage, and we add $\neg \bar{p}(\underline{t}_0)$ when $\underline{t}_0$ is ground and $p(\underline{t}_0)$ is a consequence of the previous stage. The precise definitions are as follows:

Given a program P and a language L (we do not, as in [13], take the language to be implicitly defined by the program) we enlarge L to $\bar{L}$ by adding for each

predicate p in L a new predicate $\bar{p}$ of the same arity as p. We then replace all negative literals $\neg p(\underline{t})$ in bodies of clauses by $\bar{p}(\underline{t})$ where $\bar{p}$ is a new predicate. If we were to start by forming the completion of this program then all the new predicates $\bar{p}$ would be defined to be everywhere false since they do not occur in the heads of any clauses. So we now add clauses $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$ for each new predicate $\bar{p}$. Let $\bar{P}$ be the resulting definite Horn clause program. (The new clauses have the effect that the completed definition of $\bar{p}$ in $comp(\bar{P})$ is the tautology $\bar{p}(\underline{x}) \leftrightarrow \bar{p}(\underline{x})$.) Now define

$$\bar{T}_0(P) = comp(\bar{P}),$$

$$\bar{T}_{n+1}(P) = \bar{T}_n(P) \cup \bar{P}_{n+1}(P) \cup \bar{N}_{n+1}(P),$$
$$\text{where } \bar{P}_{n+1}(P) = \{\bar{p}(\underline{t}) : \bar{T}_n(P) \vdash \neg p(\underline{t})\},$$

$$\bar{N}_{n+1}(P) = \{\neg \bar{p}(\underline{t}_0) : \underline{t}_0 \text{ ground and } \bar{T}_n(P) \vdash p(\underline{t}_0)\},$$

$$\bar{T}_\omega(P) = \bigcup_{n=0}^{\infty} \bar{T}_n(P).$$

Then SLDNFS-resolution is sound and complete for $\bar{T}_\omega(P)$ for success, i.e. *?-Q succeeds from P with answer including θ using SLDNFS-resolution iff* $\bar{T}_\omega(P) \vdash Q\theta$. It is also sound for failure, and complete for failure for positive queries, i.e. those which contain only positive literals:

- *If ?-Q fails from P using SLDNFS-resolution then* $\bar{T}_\omega(P) \vdash \neg Q$.
- *If ?-Q is a positive query and* $\bar{T}_\omega(P) \vdash \neg Q$ *then ?-Q fails from P using SLDNFS-resolution.*

This restriction on the completeness results for failure is due to the fact that none of the extensions of SLD-resolution considered is capable of using the law of contradiction, that $p \wedge \neg p$ is false, so that a query $?\text{-}p, \neg p$ which should always fail, will only do so if p succeeds or fails. So to get completeness for failure for any (2-valued) semantics we will have to impose some condition on the query. Kunen's condition [11] that the program is strict with respect to the query, seems to be the appropriate one. For Kunen's 3-valued semantics [10] this is not necessary because the law of contradiction fails in 3-valued logic, since p can be undefined, in which case $p \wedge \neg p$ is not false but undefined.

We give formal statements of these results later (Theorems 3.2, 3.4, 3.5) after a series of preliminary lemmas.

$\bar{T}_\omega(P)$ is in a sense a weak version of $comp(P)$, because

$$comp(P) \cup \bigcup_p \forall \underline{x}(\bar{p}(\underline{x}) \leftrightarrow \neg p(\underline{x})) \vdash \bar{T}_\omega(P),$$

but $\bar{T}_\omega(P) \vdash (\bar{p}(\underline{t}) \leftrightarrow \neg p(\underline{t}))$ only for certain specific $p(\underline{t})$, namely those for which this could be established by an NF step. See also Theorems 5.6, 5.8.

Note that P is not always a consequence of $\bar{T}_\omega(P)$. This is inevitable because we are aiming for completeness, and SLDNF-resolution is not always complete for P, e.g. if P is $p \leftarrow \neg p$ then p is a consequence of P but $?\text{-}p$ does not succeed.

3.1. Lemma (soundness). *If $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation of rank k with answer θ then $\bar{T}_k(P) \vdash Q\theta$, and if it has a finitely failed SLDNFS-tree of rank k then $\bar{T}_k(P) \vdash \neg Q$.*

Proof. We prove this by induction on the rank k, but to do this we need to prove a more general statement. Let us use $\bar{Q}$ to denote (ambiguously) any query obtained from Q by replacing zero or more occurrences of any atom $\neg p(\underline{t})$ by $\bar{p}(\underline{t})$. When *all* such occurrences are replaced we denote the result by $\bar{\bar{Q}}$. Our more general induction hypothesis is obtained by replacing Q by $\bar{Q}$ in the conclusions $\bar{T}_k(P) \vdash Q\theta$ and $\bar{T}_k(P) \vdash \neg Q$ of the lemma. We now prove this is true for k on the assumption it is true for all ranks less than k.

(*Refutation*): The proof is by induction on the length n of the refutation. If $n = 0$ the result is trivial. For the inductive step suppose Q is Q_1, L, Q_2 where the selected literal L is positive and unifies with the head of a clause $A \leftarrow Q'$ of P with mgu θ_1, so that the next goal is $\leftarrow (Q_1, Q', Q_2)\theta_1$. By the induction hypothesis on n, $\bar{T}_k(P) \vdash (\bar{Q}_1 \wedge \bar{\bar{Q}}' \wedge \bar{Q}_2)\theta$. But there is a clause $A \leftarrow \bar{\bar{Q}}'$ in $\bar{P}$, and $\bar{T}_k(P) \vdash comp(\bar{P}) \vdash \bar{P}$, so $\bar{T}_k(P) \vdash (\bar{Q}_1 \wedge L \wedge \bar{Q}_2)\theta$ as required. Now suppose the selected literal L is negative, say $L = \neg p(\underline{t})$, and the first step is an NF step, replacing $\leftarrow Q$ by $\leftarrow (Q_1, Q_2)\theta_1$, justified by the existence of a finitely failed tree of rank $k' < k$ for $P \cup \{\leftarrow p(\underline{t}\theta_1)\}$. By the induction hypothesis on k, $\bar{T}_{k'}(P) \vdash \neg p(\underline{t}\theta_1)$ so $\bar{p}(\underline{t}\theta_1) \in \bar{T}_{k'+1}(P)$, hence $\bar{T}_k \vdash \bar{p}(\underline{t}\theta_1)$. By the induction hypothesis on n, $\bar{T}_k(P) \vdash (\bar{Q}_1 \wedge \bar{Q}_2)\theta_1 \ldots \theta_{n+1}$ where $\theta_2, \ldots, \theta_{n+1}$ are the subsequent substitutions. Hence

$$\bar{T}_k(P) \vdash (\bar{Q}_1 \wedge \overline{\neg p(\underline{t})} \wedge \bar{Q}_2)\theta_1 \ldots \theta_{n+1}$$

whether $\overline{\neg p(\underline{t})}$ is chosen to be $\bar{p}(\underline{t})$ or $\neg p(\underline{t})$.

(*Finitely failed tree*): The proof is by induction on the height h of the tree. If the height is 1 and the selected literal L is positive then L unifies with the head of no clause in P. So it does not unify with the head of any clause in $\bar{P}$, so by [13, Lemma 15.3(a)], $comp(\bar{P}) \vdash \neg L$ hence $comp(\bar{P}) \vdash \neg \bar{Q}$, i.e. $\bar{T}_0(P) \vdash \neg \bar{Q}$, hence $\bar{T}_k(P) \vdash \neg \bar{Q}$. Suppose then that the selected literal is a ground literal $\neg p(\underline{t}_0)$ and there is a rank $k' < k$ refutation of $P \cup \{\leftarrow p(\underline{t}_0)\}$. By the induction hypothesis on k, $\bar{T}_{k'}(P) \vdash p(\underline{t}_0) \in \bar{T}_{k'+1}(P)$, so $\bar{T}_k(P) \vdash \neg \bar{Q}$, whether $\overline{\neg p(\underline{t}_0)}$ is chosen to be $\neg p(\underline{t}_0)$ or $\bar{p}(\underline{t}_0)$.

If the height is $h+1$, if Q is Q_1, L, Q_2 with L the selected positive literal then there is a child goal for each clause $A \leftarrow Q'$ of P such that A unifies with L. If θ is an mgu of A and L, this child goal will be $\leftarrow (Q_1, Q', Q_2)\theta$. By the induction hypothesis on height, $\bar{T}_k(P) \vdash \neg(\bar{Q}_1 \wedge \bar{\bar{Q}}' \wedge \bar{Q}_2)\theta$. Now $\bar{P}$ has a clause $A \leftarrow \bar{\bar{Q}}'$, so for $\bar{P} \cup \{\leftarrow \bar{Q}\}$ the corresponding child goal is $\leftarrow \{\bar{Q}_1, \bar{\bar{Q}}', Q_2)\theta$. By [13, Lemma 15.3(b)], $comp(\bar{P})$ implies that the goal $\leftarrow \bar{Q}$ is equivalent to the conjunction of child goals, and so does $\bar{T}_k(P)$ since $\bar{T}_k(P) \vdash comp(\bar{P})$. Hence $\bar{T}_k(P) \vdash \neg \bar{Q}$. If Q is Q_1, L, Q_2 and the selected literal L is a ground negative literal $\neg p(\underline{t}_0)$ where there is a finitely failed tree of rank $k' < k$ for $P \cup \{\leftarrow p(\underline{t}_0)\}$, then the next goal is $\leftarrow Q_1, Q_2$. By the induction hypothesis on k, $\bar{T}_{k'+1}(P) \vdash \neg(\bar{Q}_1 \wedge \bar{Q}_2)$, so $\bar{T}_k(P) \vdash \neg(\bar{Q}_1 \wedge \bar{L} \wedge \bar{Q}_2)$. {In fact as mentioned above there is no need to consider steps of this kind.} □

3.2. Theorem (soundness). *If $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation with answer θ then $\bar{T}_\omega(P) \vdash Q\theta$, and if it has a finitely failed SLDNFS-tree then $\bar{T}_\omega(P) \vdash \neg Q$.*

Proof. Immediate consequence of Lemma 3.1. □

3.3. Lemma (completeness). *If $A_1, \ldots, A_r$ are atoms (of the original language L) and*

(a) *$\bar{T}_k(P) \vdash A_1\theta \wedge \ldots \wedge A_r\theta$ then $P \cup \{\leftarrow A_1, \ldots, A_r\}$ has a SLDNFS-refutation of rank k with answer including θ;*

(b) *if $\bar{T}_k(P) \vdash \neg(A_1 \wedge \ldots \wedge A_r)$ then $P \cup (\leftarrow A_1, \ldots, A_r\}$ has a finitely failed SLDNFS-tree of rank k.*

Proof. By induction on k. Note that (b) implies $\bar{T}_k(p)$ is consistent since the empty goal does not have a finitely failed tree.

($k = 0$) (a): We are given $comp(\bar{P}) \vdash A_1\theta \wedge \cdots \wedge A_r\theta$. By the completeness of SLD-resolution for definite Horn programs, $\bar{P} \cup \{\leftarrow A_1, \ldots, A_r\}$ has an SLD-refutation with answer including θ. The new predicates $\bar{p}$ cannot occur in this because all literals are eventually removed, and the $\bar{p}$ could only loop, since the only clause of $\bar{P}$ with $\bar{p}$ in its head is $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$. So it is an SLDNFS-refutation of rank 0 for $P \cup \{A_1, \ldots, A_r\}$.

(b): We are given $comp(\bar{P}) \vdash \neg(A_1 \wedge \cdots \wedge A_r)$. By the completeness of NF for definite Horn programs [13, Theorem 16.1, p.95], $\bar{P} \cup \{\leftarrow A_1, \ldots, A_r\}$ has a finitely failed SLD-tree. If we take such a tree with a minimum number of goals then none of the new predicates $\bar{p}$ can be selected because, as noted above, they can only loop. So replacing all $\bar{p}$ by $\neg p$ gives a finitely failed SLDNFS-tree of rank 0 for $P \cup \{A_1, \ldots, A_r\}$.

(*Induction step*): For both (a) and (b) we use the following preliminary step:

$$\bar{T}_{k+1}(P) = comp(\bar{P}) \cup \bar{P}_{k+1}(P) \cup \bar{N}_{k+1}(P),$$

so if Q is a logical consequence of $\bar{T}_{k+1}(P)$, then by the compactness theorem it is a logical consequence of $comp(\bar{P}) \cup P^f \cup \bar{N}_{k+1}(P)$, where P^f is a finite subset of $\bar{P}_{k+1}(P)$. Let $\tilde{P} = \bar{P} \cup P^f$. {$P^f$ and $\tilde{P}$ depend on Q as well as P but we do not need to show this dependence on Q}. Then $comp(\tilde{P}) \equiv comp(\bar{P}) \cup P^f$. This is because the only difference between $comp(\bar{P})$ and $comp(\tilde{P})$ is in the definition of the $\bar{p}$ which in $\bar{P}$ is $\bar{p}(\underline{x}) \leftrightarrow \bar{p}(\underline{x})$, and in $\tilde{P}$ is

$$\bar{p}(\underline{x}) \leftrightarrow (\bar{p}(\underline{x}) \vee \exists \underline{y}_1(\underline{x} = \underline{t}_1) \vee \cdots \vee \exists \underline{y}_r(\underline{x} = \underline{t}_r)),$$

where $\bar{p}(\underline{t}_1), \ldots, \bar{p}(\underline{t}_r)$, with free variables $\underline{y}_1, \ldots, \underline{y}_r$ respectively, are the instances of $\bar{p}$ in P^f. This is easily seen to be equivalent to $\bar{p}(\underline{t}_1) \wedge \cdots \wedge \bar{p}(\underline{t}_r)$. So $comp(\tilde{P}) \cup \bar{N}_{k+1}(P) \vdash Q$.

(a). Let Q' be obtained from $Q = A_1\theta \wedge \cdots \wedge A_r\theta$ by replacing its variables by new constants. Then $comp(\tilde{P}) \cup \bar{N}_{k+1}(P) \vdash Q'$. Consider the least fixed point model $lfp(T_{\tilde{P}})$ of $comp(\tilde{P})$ (using the Herbrand universe including the new constants). This is also a model for $\bar{N}_{k+1}(P)$, otherwise it has some ground $\bar{p}(\underline{t}_0)$ true where

$\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$, i.e. $\bar{T}_k(P) \vdash p(\underline{t}_0)$. But $\bar{p}(\underline{t}_0)$ can only be in the least fixed point model if it matches the head of some clause of $\tilde{P}$, i.e. (since $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$ gives nothing new), some $\bar{p}(\underline{t}) \in P^f$, in which case $\bar{T}_k(P) \vdash \neg p(\underline{t})$ and $\bar{T}_k(P)$ would be inconsistent, which, by the induction hypothesis, it is not. So Q' is true in $lfp(T_{\tilde{P}})$, hence in all models of $\tilde{P}$, since $lfp(T_{\tilde{P}})$ is a generic model of $\tilde{P}$. Since the constants in Q' are new, Q is true in all models of $\tilde{P}$. By the completeness of SLD-resolution for definite Horn programs, there is an SLD-refutation of $\tilde{P} \cup \{\leftarrow A_1, \ldots, A_r\}$ with answer including θ. Take one of minimal length and replace all $\bar{p}$ by $\neg p$. The resolution steps in this are also resolution steps for $P \cup \{\leftarrow A_1, \ldots, A_r\}$ apart from the steps in the original refutation where some $\bar{p}(\underline{t}_1)$ was unified with an element $\bar{p}(\underline{t})$ of P^f, with mgu φ, say. But then $\bar{T}_k(P) \vdash \neg p(\underline{t})$, so $\bar{T}_k(P) \vdash \neg p(\underline{t}\varphi)$, so by the induction hypothesis there is a finitely failed SLDNFS-tree of rank k for $p(\underline{t}\varphi)$. So in the new refutation the $\neg p(\underline{t}_1)$ which replaces $\bar{p}(\underline{t}_1)$ can be eliminated by an NF step with substitution φ.

(b): We now choose P^f and $\tilde{P}$ as above so that

$$comp(\tilde{P}) \cup \bar{N}_{k+1}(P) \vdash \neg(A_1 \wedge \cdots \wedge A_r).$$

Take an SLD-tree for $\tilde{P} \cup \{\leftarrow A_1, \ldots, A_r\}$ using any fair rule and modify it by failing instantly each selected literal $\bar{p}(\underline{t}_0)$ such that $\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$. We show first that this results in a finitely failed tree. This is a generalisation of the completeness of NF for definition Horn programs [13, Theorem 16.1, pp. 95, 96], which is the special case $\bar{N}_{k+1}(P) = \emptyset$) and is proved by a refinement of that proof. Suppose the modified tree is not finitely failed. Then it has an infinite branch on which no $\bar{p}(\underline{t}_0)$ such that $\neg\bar{p}(t_0) \in \bar{N}_{k+1}(P)$ appears, because, since the selection rule is fair, if it appeared it would eventually be selected, and if it were selected it would be failed. This branch is also an infinite branch of the original tree. Take such a branch and the model I for $comp(\tilde{P}) \cup \{\exists(A_1 \wedge \cdots \wedge A_r)\}$ associated with it in the proof of [13, Theorem 16.1]. If we can show I is also a model for $\bar{N}_{k+1}(P)$ we will have the desired contradiction. So we must show that all $\neg\bar{p}(\underline{t}_0)$ in $\bar{N}_{k+1}(P)$ are true in I, i.e. that $\bar{p}([\underline{t}_0])$ is not in I. Now $I_0 \subseteq T^J_{\tilde{P}}(I_0)$ and it is easily verified that $T^J_{\tilde{P}}$ is, like $T_{\tilde{P}}$, continuous, so there is a least fixed point model $(T^J_{\tilde{P}})^\omega(I_0)$ which we may take for I. Since none of the $\bar{p}(\underline{t}_0)$ such that $\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$ appear in the branch, none of these $\bar{p}([\underline{t}_0])$ are in I_0. To show they do not belong to I, it will therefore be enough to show that they do not belong to $(T^J_{\tilde{P}})^{n+1}(I_0) - (T^J_{\tilde{P}})^n(I_0)$ for any n. For $\bar{p}([\underline{t}_0])$ to belong to this set, $\bar{p}(\underline{t}_0)$ would have to unify with the head of a clause of $\tilde{P}$ other than $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$, i.e. with some $\bar{p}(\underline{t})$ in P^f, i.e. in $\bar{P}_{k+1}(P)$, so that $\bar{T}_k(P) \vdash \neg p(\underline{t})$. Since $\underline{t}_0$ is ground we have $\underline{t}\varphi = \underline{t}_0$ for some φ hence $\bar{T}_k(P) \vdash \neg p(\underline{t}_0)$. But $\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$ so $\bar{T}_k(P) \vdash p(\underline{t}_0)$, hence $\bar{T}_k(P)$ is inconsistent, contrary to the induction hypothesis.

We now have a finitely failed tree in which the $\bar{p}(\underline{t}_0)$ such that $\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$ are failed instantly. By an easy induction on its depth we may replace it with a similar tree in which no literal $\bar{p}(\underline{t})$ is selected for unification, because $\tilde{P}$ contains a clause $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$, so that if $\bar{p}(\underline{t})$ were selected, one of the child goals would be

identical with the parent. Now replace $\bar{p}$ by $\neg p$ and we get the required finitely failed SLDNFS-tree for $P \cup \{\leftarrow A_1, \ldots, A_r\}$, since the instant failing of $\bar{p}(\underline{t}_0)$ when $\neg\bar{p}(\underline{t}_0) \in \bar{N}_{k+1}(P)$, becomes a failing of $\neg p(\underline{t}_0)$ which is justified by an NF step since $\bar{T}_k(P) \vdash p(\underline{t}_0)$, so by the induction hypothesis $P \cup \{\leftarrow p(\underline{t}_0)\}$ has an SLDNFS-refutation of rank k. □

3.4. Theorem (completeness for failure). *If Q is a positive query and $\bar{T}_\omega(P) \vdash \neg Q$ then $P \cup \{\leftarrow Q\}$ has a finitely failed SLDNFS-tree.*

Proof. Immediate consequence of Lemma 3.3(b) and the compactness theorem. □

3.5. Theorem (completeness for success). *If $\bar{T}_\omega(P) \vdash Q\theta$ then $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation with answer including θ.*

Proof. We shall prove that if $\bar{T}_k(P) \vdash Q\theta$ then $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation of rank $k+1$ with answer including θ, from which the theorem follows by compactness. Suppose Q is $A_1, \ldots, A_r, \neg B_1, \ldots, \neg B_s$, where $A_1, \ldots, A_r$, $B_1, \ldots, B_s$ are atoms. Then $\bar{T}_k(P) \vdash A_1\theta \wedge \cdots \wedge A_r\theta$, so by Lemma 3.2(a), $P \cup \{A_1, \ldots, A_r\}$ has an SLDNFS-refutation of rank k with answer θ' including θ, i.e. such that $\theta = \theta'\varphi$ for some φ. We may obviously follow a similar refutation starting with the query $\leftarrow Q$, ending with a goal $\leftarrow \neg B_1\theta', \ldots, \neg B_s\theta'$. Now $\bar{T}_k(P) \vdash \neg B_1\theta$ so by Lemma 3.2(b), $P \cup \{\leftarrow B_1\theta\}$ has a finitely failed SLDNFS-tree of rank k. Use of the NF rule with substitution φ now allows the elimination of $\neg B_1\theta'\varphi$ and the replacement of the rest of the goal by $\leftarrow \neg B_2\theta, \ldots, \neg B_s\theta$. We can now make similar steps to eliminate the remaining literals, using the NF rule with the identity substitution. □

4. Least Herbrand, fixpoint, generic model of $\bar{T}_\omega(P)$; consistency of $CWA(\bar{T}_\omega(P))$

Because it has new predicates $\bar{p}$ not totally identified with $\neg p$, the set of axioms $\bar{T}_\omega(P)$ has models with some of the properties of models of definite Horn clause programs.

4.1. Theorem. (i) *$\bar{T}_\omega(P)$ has a least Herbrand model M_0.*

(ii) *M_0 is obtainable as a least fixed point of $T_{\bar{P}}$.*

(iii) *M_0 is a generic model of $\bar{T}_\omega(P)$.*

(iv) *M_0 satisfies $\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$ for each predicate p.*

(v) *$CWA(\bar{T}_\omega(P))$ is consistent.*

Proof. (i), (ii): $T_{\bar{P}}$ here denotes the usual operator associated with the definite program $\bar{P}$ [13, p. 37]. Let $\bar{P}_\omega = \bigcup_{n=1}^{\infty} \bar{P}_n(P)$ and $\bar{P}_\omega^g$ be the set of ground elements of $\bar{P}_\omega$. Then $\bar{P}_\omega^g \subseteq T_{\bar{P}}(\bar{P}_\omega^g)$ because $\bar{P}$ has clauses $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$. So $M_0 = T_{\bar{P}}^\omega(\bar{P}_\omega^g)$ is the least fixpoint of $T_{\bar{P}}$ which contains $\bar{P}_\omega^g$. This is a model for $comp(\bar{P}) \cup \bar{P}_\omega$ and is a least Herbrand model for it (and for $\bar{P} \cup \bar{P}_\omega$), because to be an Herbrand model for $\bar{P}$ it must be closed under $T_{\bar{P}}$, and to be a model for $\bar{P}_\omega$ it must contain $\bar{P}_\omega^g$. It is also a model for $\bar{N}_\omega = \bigcup_{n=1}^{\infty} \bar{N}_n(P)$, otherwise there is some $\neg\bar{p}(\underline{t}_0)$ in some

$\bar{N}_n(P)$ such that $\bar{p}(\underline{t}_0)$ is in $T_{\bar{P}}^m(\bar{P}_\omega^g)$ for some m. By an easy induction $\bar{p}(\underline{t}_0) \in \bar{P}_\omega^g$ ($T_{\bar{P}}$ does not add any more instances of $\bar{p}$, since the only clause involving it is $\bar{p}(\underline{x}) \leftarrow \bar{p}(\underline{x})$). So there is some $\bar{p}(\underline{t})$ in some $\bar{P}_k(P)$ such that $\bar{p}(\underline{t}_0) = \bar{p}(\underline{t}\theta)$. But then $\bar{T}_r(P)$ would be inconsistent for $r \geqslant k, n$, contrary to Lemma 3.3(b).

(iii): If a ground atom A is true in M_0, it must be true in all models of $\bar{T}_\omega(P)$ because $\bar{P}_\omega^g$ must be true in all models, and a model must be closed under $T_{\bar{P}}$. Note that M_0 is also a generic model of $comp(\bar{P}) \cup \bar{P}_\omega$ and $\bar{P} \cup \bar{P}_\omega$.

(iv): If M_0 does not satisfy $\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$ then there is some ground term $\underline{t}_0$ such that both $p(\underline{t}_0)$ and $\bar{p}(\underline{t}_0)$ are true in M_0. As in (ii), the fact that $\bar{p}(\underline{t}_0)$ is in M_0 implies that, for some k, $\bar{T}_k(P) \vdash \neg p(\underline{t}_0)$, so $\bar{T}_\omega(P) \vdash \neg p(\underline{t}_0)$; since M_0 is a model for $\bar{T}_\omega(P)$ it cannot satisfy $p(\underline{t}_0)$.

(v): This follows from (iii); see [15, 20]. □

A 2-valued model M of $\bar{T}_\omega(P)$ which, like M_0, also satisfies $\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$ for all predicates can be associated with a 3-valued interpretation of the original language, by defining $p(\underline{a})$ to be true if it is true in M, false if $\bar{p}(\underline{a})$ is true in M, undefined otherwise. It would be interesting to compare such 3-valued models with Kunen's [10] 3-valued models of $comp(P)$. Theorem 5.5 below is a first step in this direction.

Adding $\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$ to $\bar{T}_\omega(P)$ does not affect the completeness of SLDNF-resolution for positive queries (for success), but does for arbitrary queries, as shown in the following theorem.

4.2. Theorem. (i) *If Q is a positive query and*

$$\bar{T}_\omega(P) \cup \bigcup_p \neg(p(\underline{x}) \wedge \bar{p}(\underline{x})) \vdash Q$$

then $\bar{T}_\omega(P) \vdash Q$.

(ii) *There is a program P and ground atom A such that*

$$\bar{T}_\omega(P) \cup \bigcup_p \neg(p(\underline{x}) \wedge \bar{p}(\underline{x})) \vdash \neg A$$

but $\bar{T}_\omega(P) \not\vdash \neg A$.

Proof. (i): It is enough to prove for Q an atom and that follows from Theorem 4.1 (iii), (iv).

(ii): Take P to be

$$s \leftarrow p, \neg p$$

$$p \leftarrow p$$

and A to be s. □

Note that the program P is not strict with respect to the query ?-s.

5. Weaker consequence relations

We prove below that the soundness result of Theorem 3.1 remains true if the consequence relation $\vdash$ is replaced by $\models_3$ (3-valued consequence relation) or $\vdash_I$ (intuitionistic derivability) or by a relation $\vdash_{3I}$ defined below which is based on inferences which are sound in both 3-valued and intuitionistic logic. Obviously the completeness results also hold with $\vdash_{3I}$ in place of $\vdash$. {Since the difficulty in getting completeness for failure for nonpositive queries referred to above arose from the law of contradiction, $\neg(p \wedge \neg p)$, which is not 3-valid, it is possible that using $\vdash_{3I}$ one might get completeness for $\bar{T}_\omega(P)$ for failure for all queries}. Our definition of $\vdash_{3I}$ requires a large number of axioms and rules of inference. In particular a lot of separate rules are needed for negation, because the method of introducing negation in intuitionistic logic, by

$$\frac{\Gamma, A: B, \qquad \Gamma, A: \neg B}{\Gamma: \neg A}$$

is not sound in 3-valued logic, since A may be undefined, and the rules for negation in 3-valued logic, e.g.

$$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$

are not intuitionistically sound.

Following Fitting [8] and Kunen [10] we use Kleene's truth tables for $\wedge$, $\vee$, $\neg$, $\supset$, $\exists$, $\forall$; these give a formula the value **t** (similarly **f**) iff all possible ways of putting in **t** or **f** for the various occurrences of the third truth value **u** lead to a value of **t** (similarly **f**) in ordinary 2-valued logic. However in forming the completion of a program P, the equivalence $\leftrightarrow$ used in forming the completed definition of each predicate p, viz. $p(\underline{x}) \leftrightarrow E_1 \vee \cdots \vee E_j$ is to have the Lukasiewicz truth table, i.e. $p \leftrightarrow q$ has value **t** if p, q have the same truth value, **f** otherwise. And we write the program clauses of P as $A_1 \wedge \cdots \wedge A_n \rightarrow A$ where $p \rightarrow q$ is the 2-valued connective "if p is true then q is true" which has the value **t** if q is **t** or p is **u** or **f**, the value **f** otherwise. If we used the Kleene implication $\supset$ here, we would not have P as a 3-valued consequence of $comp(P)$ since if p and q are both **u** then $p \leftrightarrow q$ is **t** but $p \supset q$ is **u**. And it would not permit a full range of 3-valued interpretations, since the assertion of $A_1 \wedge \cdots \wedge A_n \supset A$ excludes the possibility that all of $A_1, \ldots, A_n, A$ are **u**. Many axioms and rules of inference for $\leftrightarrow$ are needed because it is the principal connective of $comp(P)$. Note that $(p \rightarrow q) \wedge (\neg q \rightarrow \neg p)$ is a 3-valued consequence of $p \supset q$, but not vice versa.

Although our consequence relation $\vdash_{3I}$ is weaker than the usual 3-valued one, since it is also intuitionistically sound, our 3-valued logic is stronger than the system of Ebbinghaus [7] in one respect, namely all terms are considered to be defined and the equality relation is 2-valued.

We shall write $C_1, \cdots, C_n \models_3 D$ to mean that all 3-valued interpretations and assignments of variables which give all of $C_1, \ldots, C_n$ the value **t**, also give D the value **t**. It is easily verified that the consequence relation $\vdash_{3I}$ defined below is both 3-valued and intuitionistically sound, i.e.

$$\textit{If } C_1, \ldots, C_n \vdash_{3I} D \textit{ then } C_1, \ldots, C_n \models_3 D \textit{ and } C_1, \ldots, C_n \vdash_I D.$$

(In the intuitionistic case both $\rightarrow$, $\supset$ are considered to be implication, and $\leftrightarrow$ to be equivalence.)

We introduce the axioms and rules of inference $\vdash_{3I}$ as they are needed for the various parts of the soundness proof. The reader may be interested to compare these rules of inference with those of Bruffaerts and Henin in [3]. They give a Prolog meta-interpreter which produces actual proof trees justifying, on the basis of *comp*(P), the success or failure of queries. Their rules of proof are "macro" as opposed to our "micro" rules, because their object is mainly to provide a proof which is a useful explanation to the user. So they do not go into the derivations from CET but incorporate the result of Lemma 5.2(b) in a single rule:

$$\frac{\forall \neg Q\theta}{\forall \neg(s_1 = t_1 \wedge \cdots \wedge s_n = t_n \wedge Q)}$$

where θ is an mgu of $p(s_1, \ldots, s_n)$, $p(t_1, \ldots, t_n)$. It takes sixteen of our rules to derive this. Similarly they do not prove that *comp*(P) $\vdash P$ which needs seven of our rules, and since they are dealing with the Prolog selection rule they do not need any commutation rule for $\wedge$. As a result they need only fourteen rules instead of our twenty-nine. Both systems try and state rules as weakly as possible with the aim of obtaining a logic for drawing consequences from *comp*(P) corresponding closely to SLDNF-resolution. For example we both state the associative law for $\wedge$ in the form

$$\neg(A \wedge (B \wedge C)) \vdash \neg((A \wedge B) \wedge C)$$

rather than the more obvious but stronger form

$$A \wedge (B \wedge C) \leftrightarrow (A \wedge B) \wedge C.$$

Because its rules parallel SLDNF-resolution more closely, the system of [3] succeeds in this aim better than we do and is able to make do with even weaker rules. For example where we need $\neg A \vdash \neg(A \wedge B)$, $\neg B \vdash \neg(A \wedge B)$, [3] needs only the special cases $\mathbf{t} \vdash \neg(\mathbf{f} \wedge B)$ and $\neg B \vdash \neg(\mathbf{t} \wedge B)$.

Inspection of the rules of inference used in Theorems 5.4 below shows that the soundness of SLDNF- (and SLDNFS-)resolution for *comp*(P) with respect to these rules holds for the weaker version of *comp*(P) where the connective $\leftrightarrow$ used in forming the completed definitions of predicates is replaced by $\leftrightarrows$ where $p \leftrightarrows q$ is

$$(q \rightarrow p) \wedge (\neg q \rightarrow \neg p).$$

This is weaker than $p \leftrightarrow q$ in intuitionistic logic since $p \rightarrow q$ is not an intuitionistic consequence of $\neg q \rightarrow \neg p$, and also in 3-valued logic since if p is **t** and q is **u** then $p \leftrightarrow q$ is **t** but $p \leftrightarrow q$ is **f**. The same conclusion is also evident for the rules in [3]. And it is intuitively rather obvious because in SLDNF-resolution the direction of proof is always from body to head not from head to body. In the case of success it is "body therefore head"; in the case of failure it is "not body therefore not head".

In view of this Bruffaerts and Henin have suggested to us that when dealing with weak systems of inference it would be simpler to formulate the definition of *comp*(P) in this way. Instead of taking the completed definition of a predicate p to be the usual

$$\forall \underline{x}\left(p(\underline{x}) \leftrightarrow \bigvee_{j=1}^{k} \exists \underline{y}_j(\underline{x} = \underline{t}_j \wedge M_j)\right)$$

you would form *comp*(P) by adding to P the "only-if" half of this definition for each predicate p. This could be expressed in one of three different ways:

$$\forall \underline{x}\left(\neg p(\underline{x}) \leftarrow \neg \bigvee_{j=1}^{k} \exists \underline{y}_j(\underline{x} = \underline{t}_j \wedge \underline{M}_j)\right), \tag{1}^*$$

$$\forall \underline{x}\left(\neg p(\underline{x}) \leftarrow \bigwedge_{j=1}^{k} \neg \exists \underline{y}_j(\underline{x} = \underline{t}_j \wedge \underline{M}_j)\right), \tag{2}^*$$

$$\forall \underline{x}\left(\neg p(\underline{x}) \leftarrow \bigwedge_{j=1}^{k} \forall \underline{y}_j \neg(\underline{x} = \underline{t}_j \wedge \underline{M}_j)\right). \tag{3}^*$$

These versions of *comp*(P) are all equivalent to the usual *comp*(P) in classical logic, but in 3-valued and intuitionistic logic they are equivalent to the *comp*(P) defined using $\dot{\leftrightarrow}$ instead of $\leftrightarrow$. Since they use different connectives they permit different simplifications to the rules (1)-(29) used below to establish the soundness Theorem 5.4. None of them use $\leftrightarrow$, so the rules for that, (6), (7), (14), (15) of Lemma 5.2 below, are replaced by corresponding rules for $\rightarrow$, namely

$$(6)\ \frac{A \leftrightarrow B, \quad B \leftrightarrow C}{A \leftrightarrow C} \quad \text{by} \quad (25)\ \frac{A \rightarrow B, \quad B \rightarrow C}{A \rightarrow C},$$

$$(7)\ \frac{A \leftrightarrow A'}{A \wedge B \leftrightarrow A' \wedge B} \quad \text{by} \quad (7)'\ \frac{\neg A \rightarrow \neg A'}{\neg(A \wedge B) \rightarrow \neg(A' \wedge B)},$$

$$(14)\ \frac{A \supset (B \leftrightarrow C)}{A \supset (\neg B \leftrightarrow \neg C)} \quad \text{by} \quad (14)'\ \frac{A \supset (B \rightarrow C)}{A \supset (\neg\neg B \rightarrow \neg\neg C)},$$

$$(15)\ \frac{A \supset (B \leftrightarrow C)}{A \wedge B \leftrightarrow A \wedge C} \quad \text{by} \quad (15)'\ \frac{A \supset (\neg B \rightarrow \neg C)}{\neg(A \wedge B) \rightarrow \neg A \wedge C)}$$

and (18) is discarded. The CET axiom $\underline{x}=\underline{y}\supset(p(\underline{x})\leftrightarrow p(\underline{y}))$ is replaced by

$$\underline{x}=\underline{y}\supset(p(\underline{x})\rightarrow p(\underline{y}))\quad\text{and}\quad x=\underline{y}\supset(\neg p(x)\rightarrow\neg p(\underline{y})).$$

The last conclusion of Lemma 5.2 takes the form $\neg C_k\theta\rightarrow\neg C_k$ (with the notation used in the proof of Lemma 5.2). In the rest of the proof rules (26), (27), (28) used to prove $comp(P)\vdash P$ are no longer needed, since that is now trivial. If (1)* is used, the rest of the proof is unchanged, except for the replacement of (7) by (7)′ as above. The use of (2)* permits the elimination of $\vee$, i.e. the deletion of rules (21), (22), but the simplest adequate replacement rule we could find which is both 3-valued and intuitionistically valid is a generalisation of (7)′:

$$\frac{\neg A_1\wedge\cdots\wedge\neg A_r\rightarrow\neg B}{\neg(A_r\wedge C)\wedge\cdots\wedge\neg(A_r\wedge C)\rightarrow\neg(B\wedge C)}. \qquad (22)'$$

The use of (3)* also permits the elimination of $\exists$, i.e. the deletion of rules (23), (24) as well, at the cost of adding the following infinitary version of (22)′:

$$\frac{\forall x\neg A_1\wedge\cdots\wedge\forall x\neg A_r\rightarrow\neg B}{\forall x\neg(A_1\wedge C)\wedge\cdots\wedge\forall x\neg(A_r\wedge C)\rightarrow\neg(B\wedge C)}\quad x\text{ not free in }C. \qquad (22)''$$

5.1. Lemma. *SLD-resolution is sound for P for success, i.e. if $P\cup\{\leftarrow Q\}$ has an SLD-refutation with answer θ then $P\vdash\forall Q\theta$, for any consequence relation $\vdash$ containing the axiom*

$$\mathbf{t} \qquad (1)$$

and rules

$$\frac{A, A\rightarrow B}{B} \qquad (2)$$

$$\frac{A,\ B}{A\wedge B} \qquad (3)$$

$$\frac{\forall xA(x)}{A(t)} \qquad (4)$$

$$\frac{\Gamma\vdash A(y)}{\Gamma\vdash\forall xA(x)}\quad \textit{if } y \textit{ not free in } \Gamma. \qquad (5)$$

Proof. We are assuming that program clauses $A\leftarrow$ with empty bodies are written $\mathbf{t}\rightarrow A$. If we wrote them simply as A, and excluded the empty query, then we would

not need the axiom t. In (4) the usual conditions on t are required, i.e. no free occurrences of x in $A(x)$ must be bound by a variable in t.

The proof in [13 Theorem 7.1 p. 43] would need the inverse of rule (3) as well. To avoid this, prove by induction on N that if $P \cup \{\leftarrow A_1, \ldots, A_n\}$ has a refutation of length N with answer θ then $P \vdash A_i\theta$ for $i = 1, \ldots, n$. Then use (3) to put these together to give $P \vdash (A_1 \wedge \cdots \wedge A_n)\theta$ (with any desired bracketing), and (5) to universally quantify. □

5.2. Lemma. *Let* CET *be Clark's Equational Theory defined below, and let* $\vdash$ *be a consequence relation containing the axioms and rules* (1), (4), (5) *of Lemma* 5.1 *and* (6)–(16) *below. Let* $p(s_1, \ldots, s_n)$ *and* $p(t_1, \ldots, t_n)$ *be atoms.*

(a) *If* $p(s_1, \ldots, s_n)$ *and* $p(t_1, \ldots, t_n)$ *are not unifiable then*

$$\text{CET} \vdash \forall\neg(s_1 = t_1 \wedge \cdots \wedge s_n = t_n).$$

(b) *If* $p(s_1, \ldots, s_n)$ *and* $p(t_1, \ldots, t_n)$ *are unifiable with mgu* $\theta = \{x_1/r_1, \ldots, x_k/r_k\}$ *given by the unification algorithm then*

$$\text{CET} \vdash \forall((s_1 = t_1 \wedge \cdots \wedge s_n = t_n) \supset (x_1 = r_1 \wedge \cdots \wedge x_k = r_k)),$$

and if $L_1, \ldots, L_k$ *are literals and* $\underline{s} = \underline{t}$ *stands for* $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$, *then*

$$\text{CET} \vdash \forall((\ldots(\underline{s} = \underline{t} \wedge L_1) \wedge \ldots) \wedge L_k \leftrightarrow (\ldots(\underline{s} = \underline{t} \wedge L_1\theta) \wedge \ldots) \wedge L_k\theta)$$

***Axioms of CET**: The universal closures of*

$$x = x,$$

$$x = y \supset y = x,$$

$$x = y \wedge y = z \supset x = z,$$

$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \supset f(x_1, \ldots, x_n) = f(y_1, \ldots, x_n),$$

$$f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

for each function symbol f,

$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \supset (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n))$$

for each predicate symbol p,

$f(x_1, \ldots, y_n) \neq g(y_1, \ldots, y_m)$ *for each pair of distinct function symbols* f, g,

$x \neq t(x)$ *for each term* $t(x)$ *different from* x *in which* x *occurs.*

New axioms and rules for $\vdash$:

$$\frac{A \leftrightarrow B, \quad B \leftrightarrow C}{A \leftrightarrow C} \tag{6}$$

$$\frac{A \leftrightarrow A'}{A \wedge B \leftrightarrow A' \wedge B} \tag{7}$$

$$\frac{A \supset B, \quad B \supset C}{A \supset C} \tag{8}$$

$$\frac{B}{A \supset B} \tag{9}$$

$$\frac{A \supset A', \quad B \supset B'}{B \wedge A \supset B' \wedge A'} \tag{10}$$

$$\frac{A \supset A}{A \supset A \wedge A} \tag{11}$$

$$\frac{A \supset A}{A \wedge \mathbf{t} \supset A}, \quad \frac{A \supset A}{\mathbf{t} \wedge A \supset A} \tag{12}$$

$$\frac{A \supset B, \quad \neg B}{\neg A} \tag{13}$$

$$\frac{A \supset (B \leftrightarrow C)}{A \supset (\neg B \leftrightarrow \neg C)} \tag{14}$$

$$\frac{A \supset (B \leftrightarrow C)}{A \wedge B \leftrightarrow A \wedge C} \tag{15}$$

$$\frac{\neg A}{\neg (A \wedge B)}. \tag{16}$$

Proof. The first two statements are [13, Lemma 15.2, p. 90], but it is easier to use the proof of [12, 1] which proceeds by replacing equations, and can be carried out with the given axioms and rules. Note that the use of $\supset$ in the axioms of CET is appropriate since equality is 2-valued. Indeed the second and third axioms give $x = y \supset x = y$ which implies that $x = y$ is either **t** or **f**; but note that our axioms and rules, because of their intuitionist soundness, do not allow us to deduce $x = y \vee x \neq y$. The axiom involving p must not be strengthened by replacing $\leftrightarrow$ by $\supset$ since that would force a 2-valued interpretation of the atom $p(x_1, \ldots, x_n)$. The first statement of (b) holds also with the $\supset$-sign reversed, but that is not needed for the soundness proof.

The proof starts by using (1), (8)–(12), to establish derived rules

$$\frac{A \supset B, \quad B \supset B}{A \wedge C \supset B}, \quad \frac{A \supset B, \quad B \supset B}{C \wedge A \supset B}, \tag{D_1}$$

$$\frac{A \supset A, A \supset B, A \supset C}{A \supset B \wedge C}. \tag{D_2}$$

These would be sound (both intuitionistically and 3-valued) without the premisses $B \supset B$, $A \supset A$ (which in 3-valued logic say that B, A are either t or f, not u) but the weaker versions here are all we need because, as noted above, the symmetry and transititivity axioms of CET give $A \supset A$ when A is an equation and (10) then gives this when A is a conjunction of equations, which is all we are concerned with. From $(\mathbf{D}_1)$, $(\mathbf{D}_2)$ we can prove that

$$E_1 \wedge E_2 \supset E_2 \wedge E_1 \quad \text{and} \quad (E_1 \wedge E_2) \wedge E_3 \supset E_1 \wedge (E_2 \wedge E_3)$$

if E_1, E_2, E_3 are equations.

The procedure in [1] starts with the conjunction of equations C_0: $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ and after a finite sequence of replacement steps either ends with the unification equations $x_1 = r_1 \wedge \cdots \wedge x_k = r_k$ or ends in failure. We show that each replacement step applied to a conjunction of equations C gives a conjunction C' such that $C \supset C'$ is derivable (from CET and our rules). The transitivity of $\supset$ (rule (8)) then gives the first part of (b) if unification is successful. Failure occurs when the final conjunction C_n contains an equation $E: f(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$ with f different from g, or $x = t$ where t is different from x but contains x. The CET axioms and (4) yield $\neg E$, then (16) and transitivity and commutativity of $\wedge$ give $\neg C_n$ and (13) gives $\neg C_0$. In the replacement steps, the fact that we have $C \supset C$ and (10) means that we need only consider the actual equations which are replaced. The crucial case is where the conjunction C contains an equation $x = t$ with x not in t. Then C' is obtained by replacing x by t in every other equations. To show that $C \supset C'$ here, you first prove by induction on the depth of t_1, using (9), (10) and $(\mathbf{D}_2)$, that $x = t \supset t_1 = t_1(x/t)$. Then show, using this and the symmetry and transitivity of $=$, that

$$x = t \wedge t_1 = t_2 \supset x = t \wedge t_1(x/t) = t_2(x/t),$$

and use commutativity and associativity of $\wedge$ and (8) and (10) to extend this to replacement of x by t in any conjunction.

We now have (a) and the first part of (b). The second part of (b) will follow by (5) if we can derive

$$(\ldots(\underline{s} = \underline{t} \wedge L_1) \wedge \ldots) \wedge L_k \leftrightarrow (\ldots(\underline{s} = \underline{t} \wedge L_1\theta) \wedge \ldots) \wedge L_k\theta,$$

from CET, for literals $L_1, \ldots, L_k$, where $\underline{s} = \underline{t}$ abbreviates $s_1 = t_1 \wedge \cdots \wedge a_n = t_n$. We have just shown $\underline{s} = \underline{t} \supset \underline{x} = \underline{r}$, and

$$\underline{x} = \underline{r} \supset t_1 = t_1\theta \wedge \cdots \wedge t_k = t_k\theta$$

follows easily from $x = t \supset t_1 = t_1(x/t)$ using $(\mathbf{D}_2)$ and the fact that θ is idempotent, i.e. $\underline{r}$ does not contain any of the variables $\underline{x}$. A CET axiom gives, by (4),

$$t_1 = t_1\theta \wedge \cdots \wedge t_k = t_k\theta \supset (p(t_1, \ldots, t_k) \leftrightarrow p(t_1\theta, \ldots, t_k\theta)),$$

so using (8) we get

$$\underline{s} = \underline{t} \supset (p(t_1, \ldots, t_n) \leftrightarrow p(t_1\theta, \ldots, t_k\theta)).$$

Now (14) gives

$$\underline{s} = \underline{t} \supset (\neg p(t_1, \ldots, t_k) \leftrightarrow \neg p(t_1\theta, \ldots, t_k\theta)),$$

so (15) gives

$$\underline{s} = \underline{t} \wedge L_1 \leftrightarrow \underline{s} = \underline{t} \wedge L_1\theta \quad \text{for a single literal } L_1.$$

Now prove by induction on k that

$$(\ldots(\underline{s} = \underline{t} \wedge L_1) \wedge \cdots) \wedge L_k \leftrightarrow (\ldots(\underline{s} = \underline{t} \wedge L_1\theta) \wedge \cdots) \wedge L_k\theta,$$

or, in short $C_k \leftrightarrow C_k\theta$. By induction on k using $(\mathbf{D}_1)$ and (8) we have $C_k \supset \underline{s} = \underline{t}$, also $\underline{s} = \underline{t} \supset (L_{k+1} \leftrightarrow L_{k+1}\theta)$ as proved above, so by (8), $C_k \supset (L_{k+1} \leftrightarrow L_{k+1}\theta)$, and by (15)

$$C_k \wedge L_{k+1} \leftrightarrow C_k \wedge L_{k+1}\theta.$$

The induction hypothesis and (7), (6) then give $C_{k+1} \leftrightarrow C_{k+1}\theta$. □

5.3. Lemma. *Suppose an SLDNF-resolution step involving the selection of a positive literal of the goal* $\leftarrow Q$ *using the program* P *is made. Then if* $\vdash$ *includes the axioms and rules* (1)-(16) *of Lemmas* 5.1, 5.2 *and* (17)-(24) *below, then we have*

(a) *if there are no derived goals then* $comp(P) \vdash \forall\neg Q$,

(b) *if the set of derived goals* $\{\leftarrow Q_1, \ldots, \leftarrow Q_r\}$ *is non-empty then* $comp(P)$, $\forall\neg Q_1, \ldots, \forall\neg Q_r \vdash \forall\neg Q$.

New axioms and rules for $\vdash$:

$$\frac{A, \quad A \supset B}{B} \tag{17}$$

$$\frac{A \leftrightarrow B}{B \rightarrow A} \tag{18}$$

$$\frac{\neg(A \wedge B)}{\neg(B \wedge A)} \tag{19}$$

$$\frac{\neg(A \wedge (B \wedge C))}{\neg((A \wedge B) \wedge C)} \tag{20}$$

$$\frac{\neg A, \quad \neg B}{\neg(A \vee B)} \tag{21}$$

$$\frac{\neg(A \wedge B), \quad \neg(A \wedge C)}{\neg(A \wedge (B \vee C))} \tag{22}$$

$$\frac{\forall x \neg A(x)}{\neg \exists x A(x)} \tag{23}$$

$$\frac{\neg \exists x(A \wedge B)}{\neg((\exists x A) \wedge B)} \quad x \text{ not free in } B. \tag{24}$$

Proof. {This version of (b) is all we need for our soundness results since we are only interested in finitely failed trees. If one wished to use incomplete derivation trees for other purposes, e.g. partial evaluation, one might want the usual form of (b), i.e.

$$comp(P) \vdash (\forall \neg Q \leftrightarrow \forall \neg Q_1 \wedge \cdots \wedge \forall \neg Q_r).$$

This can be obtained at the cost of strengthening some of our rules, e.g. replacing $\forall x \neg A \vdash \neg \exists x A$ by $\forall x \neg A \leftrightarrow \neg \exists x A$.}

We need two derived rules. From (9), (14), (17) and (1) we get

$$\frac{A \leftrightarrow B}{\neg A \leftrightarrow \neg B} \qquad (\mathbf{D}_3)$$

and from this and (18) we get

$$\frac{A \leftrightarrow B, \quad \neg B}{\neg A}. \qquad (\mathbf{D}_4)$$

Let $p(\underline{s})$ be the selected positive literal. The completed definition of p in $comp(P)$ is

$$\forall \underline{x}\left(p(\underline{x}) \leftrightarrow \bigvee_{j=1}^{k} D_j\right)$$

where D_j is $\exists \underline{y}_j(\underline{x} = \underline{t}_j \wedge \underline{M}_j)$, where $\underline{M}_j$ is some conjunction of literals and the $\underline{x}$ are variables occurring in no $\underline{M}_j$. The first step is to use rule (4) to get

$$p(\underline{s}) \leftrightarrow \bigvee_{j=1}^{k} D_j'$$

where D_j' is $\exists \underline{y}_j(\underline{s} = \underline{t}_j \wedge \underline{M}_j)$. This is only legitimate if the variables $\underline{y}_j$ do not occur in $\underline{s}$. However our premisses and conclusion in Lemma 5.3 are the universal closures of $\neg Q_1, \ldots, \neg Q_r, \neg Q$ so we may use rules (4), (5) to change the variables in Q so that they are distinct from those in each clause $p(\underline{t}_j) \leftarrow \underline{M}_j$ used in forming $comp(P)$. And since the use of any variant of a program clause which has no variables in common with the current goal gives derived goals which are variants of each other, we may suppose that $p(\underline{t}_j) \leftarrow \underline{M}_j$ is also the same program clause variant as is used in the resolution step.

(a): Since there is no derived goal, $p(\underline{s})$ does not unify with the head $p(\underline{t}_j)$ of any clause of P so by Lemma 5.2(a) we can derive $\neg(\underline{s} = \underline{t}_j)$ from CET and hence from $comp(P)$. Rule (16) now gives $\neg(\underline{s} = \underline{t}_j \wedge \underline{M}_j)$; (23) and (5) give $\neg \exists \underline{y}_j(\underline{s} = \underline{t}_j \wedge \underline{M}_j)$, i.e. $\neg D_j'$; (21) gives $\neg \bigvee_{j=1}^{k} D_j'$; then $\mathbf{D}_4$ gives $\neg p(\underline{s})$. Finally (16), (19) and (5) give $\forall \neg Q$,

(b): Suppose Q is $L_1 \wedge \cdots \wedge L_i \wedge \cdots \wedge L_n$ with L_i the selected literal. If $p(\underline{s})$ unifies with $p(\underline{t}_j)$ with mgu θ_j then the corresponding derived goal Q_j is

$$(L_1 \wedge \cdots \wedge L_{i-1} \wedge \underline{M}_j \wedge L_{i+1} \wedge \cdots \wedge L_n)\theta_j.$$

We are given $\forall\neg Q_j$, so by (4), (19) and (20) we get $\neg(\underline{M}_j\theta_j \wedge \underline{L}\theta_j)$ where $\underline{L}$ is

$$L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n.$$

Rule (16) gives $\neg(\underline{s} = \underline{t}_j \wedge \underline{M}_j\theta_j \wedge \underline{L}\theta_j)$ and (20) and Lemma 5.2(b) and $\mathbf{D}_4$ give

$$\neg(\underline{s} = \underline{t}_j \wedge \underline{M}_j \wedge \underline{L}).$$

If $p(\underline{s})$ does not unify with $p(\underline{t}_j)$ we also have this, as in (a). Now using (24), (23) and (5) we get

$$\neg(\exists \underline{y}_j(\underline{s} = \underline{t}_j \wedge \underline{M}_j) \wedge \underline{L})$$

i.e. $\neg(D'_j \wedge \underline{L})$. Using (22) we get $\neg(\bigvee_{j=1}^{k} D'_j \wedge L)$ and since $p(\underline{s}) \leftrightarrow \bigvee_{j=1}^{k} D'_j$, (7) and $\mathbf{D}_4$ give $\neg(p(\underline{s}) \wedge L)$ and then (19), (20) and (5) give $\forall\neg Q$. □

5.4. Theorem. *Let $\vdash_{3I}$ be the consequence relation defined by the axioms and rules* (1)–(24) *listed in Lemmas* 5.1, 5.2, 5.3 *and* (25)–(29) *below. Then SLDNFS-resolution is sound with respect to $\vdash_{3I}$ for both comp(P) and $\bar{T}_\omega(P)$, i.e. if $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation with answer θ then $comp(P) \vdash_{3I} \forall Q\theta$ and $\bar{T}_\omega(P) \vdash_{3I} \forall Q\theta$, and if $P \cup \{\leftarrow Q\}$ has a finitely failed SLDNFS-tree then $comp(P) \vdash_{3I} \forall\neg Q$ and $\bar{T}_\omega(P) \vdash_{3I} \forall\neg Q$. Furthermore the same $\bar{T}_\omega(P)$ is obtained if the $\vdash$ in its definition is replaced by $\vdash_{3I}$.*

New rules for $\vdash_{3I}$:

$$\frac{A \rightarrow B, \quad B \rightarrow C}{A \rightarrow C} \tag{25}$$

$$\frac{A}{B \rightarrow A \wedge B} \tag{26}$$

$$A(t) \rightarrow \exists x A(x) \tag{27}$$

$$A \rightarrow A \vee b, \qquad A \rightarrow B \vee A \tag{28}$$

$$\frac{A}{\neg\neg A}. \tag{29}$$

Proof. {In (27) no free occurrence of x in $A(x)$ must be bound by a variable in t.} The first step is to show $comp(P) \vdash_{3I} P$. With the notation of the last lemma, the completed definition of p in $comp(P)$ is

$$\forall \underline{x}\left(p(\underline{x}) \leftrightarrow \bigvee_{j=1}^{k} \exists \underline{y}_j(\underline{x} = \underline{t}_j \wedge \underline{M}_j)\right).$$

Let $\underline{t}'_j$, $\underline{M}'_j$ be the result of replacing $\underline{y}_j$ by different distinct variables $\underline{y}'_j$ in $\underline{t}_j$, $\underline{M}_j$. Then rule (4) gives

$$p(\underline{t}'_j) \leftrightarrow \bigvee_{j=1}^{k} \exists \underline{y}_j(\underline{t}'_j = \underline{t}_j \wedge \underline{M}_j).$$

We have $\underline{t}'_j = \underline{t}'_j$, so by (26) $\underline{M}'_j \to \underline{t}'_j = \underline{t}'_j \wedge \underline{M}'_j$, so repeated use of (27) and (25) gives

$$\underline{M}'_j \to \exists \underline{y}_j(\underline{t}'_j = \underline{t}_j \wedge \underline{M}_j).$$

Using (28) and (25) we get

$$\underline{M}'_j \to \bigvee_{j=1}^{k} \exists \underline{y}_j(\underline{t}'_j = \underline{t}_j \wedge \underline{M}_j)$$

and (18) and (25) gives $\underline{M}'_j \to p(\underline{t}'_j)$, and using rules (5) and (4) to change variables in the closure we get $\forall(\underline{M}_j \to p(\underline{t}_j))$, which is the appropriate clause of P.

The rest of the proof consists of checking that $\vdash_{3I}$ is strong enough to support the arguments in the proof of Lemma 3.1 above and a similar argument with $\bar{T}_k(P)$ replaced throughout by $comp(P)$. In the case of $\bar{T}_\omega(P)$ the use of $\vdash$ in its definition is replaced by $\vdash_{3I}$, giving soundness for what we will call $\bar{T}^{3I}_\omega(P)$. Finally we prove by induction on k that $\bar{T}^{3I}_k(P) = \bar{T}_k(P)$. This is obvious for $k = 0$. For the induction step we have, by the induction hypothesis, $\bar{T}^{3I}_k(P) \vdash_{3I} \neg p(\underline{t})$ is the same as $\bar{T}_k(P) \vdash_{3I} \neg p(\underline{t})$, which implies $\bar{T}_k(P) \vdash \neg p(\underline{t})$ which, by Lemma 3.3(b), implies that $P \cup \{\leftarrow p(\underline{t})\}$ has a finitely failed SLDNFS-tree of rank k. By the soundness result proved as the first part of the present theorem this implies $\bar{T}^{3I}_k(P) \vdash_{3I} \neg p(\underline{t})$. So $\bar{P}^{3I}_{k+1}(P) = \bar{P}_{k+1}(P)$ and similarly $\bar{N}^{3I}_{k+1}(P) = \bar{N}_{k+1}(P)$, so $\bar{T}^{3I}_{k+1}(P) = \bar{T}_{k+1}(P)$. □

Instead of going through the argument separately for $comp(P)$ you can, at the cost of adding a few more rules, use Theorem 5.6 below, which depends on the next lemma.

5.5. Lemma. *If $\vdash_{3I}$ is extended by adding rules* (30)-(35) *below, then*

$$comp(P) \cup \bigcup_p \forall(\bar{p}(\underline{x}) \leftrightarrow \neg p(\underline{x})) \vdash_{3I} \bar{T}_\omega(P).$$

New rules for $\vdash_{3I}$

$$\frac{A \supset (B \leftrightarrow C), \quad A \supset (C \leftrightarrow D)}{A \supset (B \leftrightarrow D)} \tag{30}$$

$$\frac{A \leftrightarrow A', \quad B \leftrightarrow B'}{A \wedge B \leftrightarrow A' \wedge B'} \tag{31}$$

$$\frac{A \leftrightarrow A', \quad B \leftrightarrow B'}{A \vee B \leftrightarrow A' \vee B'} \tag{32}$$

$$\exists x(A \wedge B) \leftrightarrow \exists x A \wedge B \quad x \text{ not free in } B \tag{33}$$

$$\frac{A}{B \leftrightarrow A \wedge B} \tag{34}$$

$$\frac{A \leftrightarrow A'}{\exists x A \leftrightarrow \exists x A'} \tag{35}$$

$$\frac{A \leftrightarrow B}{B \leftrightarrow A}. \tag{36}$$

Proof. Note that (30), (31), (33), (34) are strengthened versions of (6), (7), (24), (26). Rule (36) excludes the asymmetric interpretation of $\leftrightarrow$ as $\dot{\leftrightarrow}$ referred to before Lemma 5.1, but it is not needed if the hypothesis is strengthened by adding $\bigcup_p \forall(\neg p(\underline{x}) \leftrightarrow \bar{p}(\underline{x}))$. Show by induction on k that

$$comp(P) \cup \bigcup_p \forall(\bar{p}(\underline{x}) \leftrightarrow \neg p(\underline{x})) \vdash_{3l} \bar{T}_k^{3l}(P),$$

then use the fact that $\bar{T}_\omega^{3l}(P) = \bar{T}_\omega(P)$. The inductive step is easy; the new rules are needed to replace $\neg p$ by $\bar{p}$ in $comp(P)$ and so obtain $comp(\bar{P})$. □

5.6. Theorem. *If φ is any sentence of the original language L then $\bar{T}_\omega(P) \vdash \varphi$ implies $comp(P) \vdash \varphi$. Similarly with $\vdash_1$, $\models_3$ or $\vdash_{3l}$ in place of $\vdash$. (Provided $\vdash_{3l}$ is extended by* (30)–(35) *and by $A \leftrightarrow A$.)*

Proof. By Lemma 5.5, if $\bar{T}_\omega(P) \vdash \varphi$ then

$$comp(P) \cup \bigcup_p \forall(\bar{p}(\underline{x}) \leftrightarrow \neg p(\underline{x})) \vdash \varphi.$$

If we replace $\bar{p}(\underline{t})$ by $\neg p(\underline{t})$ everywhere in this proof of φ we get a proof of φ from

$$comp(P) \cup \bigcup_p \forall(\neg p(\underline{x}) \leftrightarrow \neg p(\underline{x})),$$

i.e. from $comp(P)$. Similarly for $\vdash_1, \vdash_{3l}$. For $\models_3$, either replace it with a sound and complete 3-valued derivability relation $\vdash_3$ and argue similarly, or use the fact that any 3-valued model for $comp(P)$ over the language L can be extended to a model of

$$comp(P) \cup \bigcup_p \forall(\bar{p}(\underline{x}) \leftrightarrow \neg p(\underline{x}))$$

by defining $\bar{p}(\underline{a}) \leftrightarrow \neg p(\underline{a})$ for all predicates p and constant $\underline{a}$. □

For definite Horn clause programs, $\bar{T}_\omega(P)$ adds no positive information, as shown in this theorem.

5.7. Theorem. *If P is a definite Horn clause program and φ is a positive sentence then $\bar{T}_\omega(P) \vdash \varphi$ implies $P \vdash \varphi$.*

Proof. By Theorem 5.6 and the corresponding result for $comp(P)$. □

5.8. Theorem. *A 3-valued model of $comp(P)$ can be extended to a 3-valued model of*

$$\bar{T}_\omega(P) \cup \bigcup_p \forall\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$$

by defining $\bar{p}(\underline{a}) \leftrightarrow \neg p(\underline{a})$ for all predicates p and constant $\underline{a}$. By replacing **u** *by* **f** *this gives a 2-valued model of $\bar{T}_\omega(P) \cup \bigcup_p \forall\neg(p(\underline{x}) \wedge \bar{p}(\underline{x}))$.*

Proof. The first part is an immediate consequence of Lemma 5.5. The second part depends on the fact that replacing **u** by **f** does not destroy the truth of $comp(\bar{P})$ (basically because negation does not occur), and it does not destroy the truth of the $\bar{p}(\underline{t})$ in the $\bar{P}_n(P)$ or of the $\neg\bar{p}(\underline{t}_0)$ in the $\bar{N}_n(P)$, since these atoms are already respectively **t** and **f**, not **u**. □

The fact established in Theorem 5.4 that SLDNF-resolution is sound for $comp(P)$ with respect to the consequence relation $\vdash_{3I}$ gives some reason why it is not always complete with respect to the 3-valued consequence relation $\models_3$. Let us take the example given in [10], of the program P:

$$p(x) \leftarrow isc(x)$$

$$p(x) \leftarrow nonc(x)$$

$$isc(c)$$

$$nonc(x) \leftarrow not\ isc(x).$$

The completion of P is equivalent to

$$p(x) \leftrightarrow x = c \vee x \neq c$$

$$isc(x) \leftrightarrow x = c$$

$$nonc(x) \leftrightarrow x \neq c.$$

Now $comp(P) \models_3 \forall x p(x)$ (since $=$ is taken as 2-valued) but ?-$p(x)$ does not succeed with answer the identity. This is explained by the fact that $comp(P) \not\vdash_{3I} \forall x p(x)$.

However this is not the only reason for the incompleteness of SLDNF-resolution for $comp(P)$ with respect to $\models_3$, because it is not complete with respect to $\vdash_{3I}$ either. If P is taken to be

$$q \leftarrow \neg p(x)$$

$$p(x)$$

then $comp(P) \vdash_{3I} \neg q$, but ?-$\neg q$ does not succeed under SLDNF-resolution, or even under SLDNFS-resolution. To get ?-$\neg q$ to succeed, i.e. to get ?-q to fail we need to further extend the NF rule as in Section 6 below, to allow ?-$\neg p(x)$ to fail because ?-$p(x)$ succeeds with answer the identity.

6. Symmetric variants of $\bar{T}_\omega$

The definition of $\bar{T}_\omega$ is asymmetric; in forming $\bar{N}_{n+1}(P)$ we consider only ground atoms $p(\underline{t}_0)$, but in forming $\bar{P}_{n+1}(P)$ we allow all atoms $p(\underline{t})$. Let us consider $\bar{T}^-_\omega(P)$ which is obtained by restricting to ground atoms in forming $\bar{P}_{n+1}(P)$ also, and $\bar{T}^+_\omega(P)$ which is obtained by allowing non-ground atoms in the definition of $\bar{N}_{n+1}(P)$ as well as $\bar{P}_{n+1}(P)$.

Examination of the proofs of Theorems 3.1 and 5.4 shows that SLDNF-resolution is sound, for success and failure, and with respect to $\vdash_{3I}$ as well as $\vdash$, for $\bar{T}^{-}_{\omega}(P)$. This is expected, because the only reason for allowing non-ground atoms in the definition of $\bar{P}_{n+1}(P)$ was to get soundness for the extended NF rule. Since $\bar{T}^{-}_{\omega}(P)$ is weaker than $\bar{T}_{\omega}(P)$, the completeness Theorems 3.4 and 3.5 still hold, although for SLDNF-resolution the conclusions must be weakened to admit the possibility of a generalised flounder. Since this is the best one can get using $\bar{T}_{\omega}(P)$ it is arguable that for SLDNF-resolution, $\bar{T}^{-}_{\omega}(P)$ is a more appropriate semantics than $\bar{T}_{\omega}(P)$. It is also appropriate for SLDNFS-resolution if one restricts to ground queries, for Theorem 3.1 holds in the form "If $P \cup \{\leftarrow Q\}$ has an SLDNFS-refutation with answer θ and $Q\theta\sigma$ is ground then $\bar{T}^{-}_{\omega}(P) \vdash Q\theta\sigma$, and if it has a finitely failed SLDNFS-tree then $\bar{T}^{-}_{\omega}(P) \vdash \neg Q\sigma$, if $Q\sigma$ is ground."

To prove this replace "$Q\theta$", "$\neg Q$" in the proof of Theorem 3.1 by "all ground instances of $Q\theta$ (or $\neg Q$)".

$\bar{T}^{+}_{\omega}(P)$ corresponds to the further extension of SLDNFS-resolution obtained by allowing the NF rule to be used in the form:

If A succeeds with answer the identity then $\neg A$ fails.

This is justified by the validity of $\forall A \Rightarrow \neg\exists\neg A$, which shows that if SLDNF-resolution is sound for some $H(P)$ for both success and failure, then it will remain so if NF is extended in this way. The results proved above for SLDNFS-resolution and $\bar{T}_{\omega}(P)$ all hold for this further extended form with respect to $\bar{T}^{+}_{\omega}(P)$.

Note that for this further extension of SLDNF-resolution the analogue of Theorem 2.2 fails. Taking P to be

$$q \leftarrow q, \neg p(x)$$

$$p(x)$$

the query $?\text{-}\neg q$ succeeds under this extended form of resolution, but under SLDNF-resolution it neither succeeds nor has a generalised flounder, but an infinite derivation tree.

We do not know whether this further extension of SLDNFS-resolution is complete for $comp(P)$ with respect to $\vdash_{3I}$ as defined by rules (1)-(31) above. But it is not complete if $\vdash_{3I}$ is extended so as to allow the derivation from CET of

$$\neg(\neg\neg(x = a) \wedge \neg(x = a))$$

which is valid intuitionistically, and is also a valid 3-valued consequence of CET, since that implies $x = a \supset x = a$ which means that $x = a$ is **t** or **f** not **u**. This is shown by the program

$$s \leftarrow \neg p(x) \wedge \neg q(x)$$

$$p(x) \leftarrow \neg q(x)$$

$$q(a)$$

because we then have $comp(P) \vdash \neg s$, but $\neg s$ does not succeed under this extension of SLFNFS-resolution.

7. Conclusion

The definition of $\bar{T}_\omega(P)$ is not sufficiently simple for the programmer to grasp at once when he writes down the program P. This is also true of the default operator $SYNTH(P)$ for which Reynolds [17] has recently obtained some completeness results. And it is actually true in general of $comp(P)$. In simple cases the meaning of $comp(P)$ can be simply read off from P, but in the presence of recursion and negation this is by no means true. Indeed using the "extended syntax" of Lloyd and Topor [13, Chapter 4] given any sentence φ of first-order logic one can construct a P for which $comp(P)$ is essentially equivalent to φ. The fact that to get soundness and completeness one has to distort the original program P to such an extent suggests that negation as failure really is a very complicated device, which cannot be given a simple declarative meaning. Perhaps the best way to proceed is to look for further generalisations of natural conditions like allowed and semi-strict which give completeness for $comp(P)$. The present completeness results for $\bar{T}_\omega(P)$ may possibly be of technical use in such a development.

Acknowledgment

I should like to thank the Forschungszentrum für Informatik of the Eidgenössische Technische Hochschule in Zürich for providing support and ideal conditions for the writing of the paper.

I am grateful to Gerhard Jäger for helpful discussions and to Robert Stärk for pointing out mistakes in, and making corrections to, an earlier version.

References

[1] K.R. Apt, Introduction to logic programming, Technical Report TR-87-35, Department of Computer Science, University of Texas at Austin, 1987.

[2] E. Börger, Unsolvable decision problems for prolog programs, in: G. Goos and J. Hartmanis, eds., *Computation Theory and Logic*, Lecture Notes in Computer Science, Vol. 270 (Springer, Berlin, 1987).

[3] A. Bruffaerts and E. Henin, Proof trees for negation as failure or yet another prolog meta-interpreter, Philips Research Laboratory Report R524, January 1988.

[4] L. Cavedon and J.W. Lloyd, A completeness theorem for SLDNF-resolution, Technical Report CS-87-06, University of Bristol, 1987.

[5] D. Chan, Constructive negation based on the completed database, Preprint, European Computer Industry Research Centre, Arabellastrasse 17, 8000 Munchen 81, Fed. Rep. Germany, 1987.

[6] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (Plenum Press, New York, 1978) 293-322.

[7] H-D. Ebbinghaus, Über eine Prädikatenlogik mit partiell definierten Prädikaten und Funktionen, *Arch. Math. Logik* **12** (1969) 39-53.

[8] M.R. Fitting, A Kripke-Kleene semantics for logic programs, *J. Logic Programming* **2** (1985) 295-312.

[9] G. Jäger, Non-monotonic reasoning by axiomatic extensions, in: *Abstracts 8th Internat. Cong. Logic Methodology and Philosophy of Science*, Moscow (1987) 6.

[10] K. Künen, Negation in logic programming, *J. Logic Programming* **4** (1987) 289-308.

[11] K. Künen, Signed data dependencies in logic programs, Computer Sciences Technical Report No. 719, University of Wisconsin, Madison, October 1987.

[12] J-L. Lassez, M.J. Maher and K. Marriott, Unification re-visited, Technical Report RC12394 (No. 55630), IBM T.J. Watson Research Laboratory, 1986.

[13] J.W. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 1987, 2nd ed.).

[14] J.W. Lloyd and J.C. Shepherdson, Partial evaluation in logic programming. Technical Report CS-87-09, University of Bristol, 1987.

[15] J.A. Makowsky, Why Horn formulas matter in computer science; initial structures and generic examples, in: H. Ehrig et al., eds., *Mathematical Foundations of Software Development, Proc. Internat. Joint Conf. on Theory and Practice of Software Development (TAPSOFT)*, Lecture Notes in Computer Science, Vol. 185 (Springer, Berlin, 1985) 374-387. Revised version, Preprint, 1986.

[16] T.C. Przymusinksi, On the declarative and procedural semantics of logic programs, Preprint, 1987.

[17] M. Reynolds, A completeness result for logic programming, *J. Logic Programming*, to appear.

[18] J.C. Shepherdson, Negation as failure: A comparison of Clark's completed data base and Reiter's closed world assumption, *J. Logic Programming* **1** (1984) 51-81.

[19] J.C. Shepherdson, Negation as failure II, *J. Logic Programming* **3** (1985) 185-202.

[20] J.C. Shepherdson, Negation in logic programming, in: J. Minker, ed., *Introduction to Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988).

[21] J.C. Shepherdson, Unsolvable problems for SLDNF-resolution, Technical Report PM-88-02, University of Bristol, 1988 also: *J. Logic Programming*, to appear.