N-PROLOG: AN EXTENSION OF PROLOG WITH HYPOTHETICAL IMPLICATION. II. LOGICAL FOUNDATIONS, AND NEGATION AS FAILURE

D. M. GABBAY

0. INTRODUCTION

This continuation paper investigates the logical properties of N-PROLOG and the way it relates to classical logic and the classical quantifiers. We shall also introduce negation as failure into N-PROLOG. We shall give practical examples of using logic to control the execution of programs in N-PROLOG and examine the validity of the thesis.

algorithm = logic + (control in) logic.

We shall see that success in the N-PROLOG computation of a goal G from the database $\mathbf{P}$ means logically that $\mathbf{P} \vdash G$ in intuitionistic logic. We will also introduce an additional computational rule called the *restart rule* (allowing one to replace, at any time of the computation, the current goal by the original goal).

Success of an original goal G from a database $\mathbf{P}$ through a computation in N-PROLOG with the restart rule means that $\mathbf{P} \vdash G$ in classical logic. The restart rule can be modified to yield many logics intermediate between classical and intuitionistic logic.

In my lecture notes [1], I use these ideas to present classical logic procedurally, in a PROLOG-like way. My paper with K. Broda and F. Kriwaczek [2] describes a theorem prover based on these ideas and compares this theorem prover with SL resolution and other forms of resolution. Modal and temporal logics can also be represented in N-PROLOG in certain ways, and especially applied to database deletion, temporal updating, and management. This will be dealt with in paper III of this series.

1. COMBINATORIAL PROPERTIES OF PROPOSITIONAL N-PROLOG

In this section we prove Lemma L3 of Section 3 of paper I of this series, namely, we show that: $\mathbf{P}?A = 1$ and $\mathbf{P}?(A \to B) = 1$ imply $\mathbf{P}?B = 1$. Then we use this lemma to

Address correspondence to D. M. Gabbay, Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, England.

THE JOURNAL OF LOGIC PROGRAMMING

0743-1066/85/$03.30

prove a first completeness theorem for N-PROLOG. We need some definitions of the notions of complexity and depth.

Definition D1 (An inductive definition of the complexity of N-clauses).

(a) We define the complexity of N-clauses by induction:

(a1) An atomic proposition is of complexity 1.

(a2) complexity($A \to q$) = complexity(A) + 1.

(a3) complexity($A1 \wedge A2$) = 1 + max(complexity(Ai)).

(b) Let **P** be a finite set of clauses. The complexity of **P** is defined as a function $f(n)$ on natural numbers giving for each n the number $f(n)$ of clauses in **P** of complexity n. Let $|f|$ be the first natural number m such that $[f(m) \neq 0)$ and $\forall m' > m\, f(m') = 0]$.

Since **P** is finite, such an $|f|$ exists.

(c) Given two finite sets of clauses **P1** and **P2**, let $f1$, $f2$ be their complexity functions.

Define an ordering $<$ on the functions as follows:

(1) $f1 < f2$ if $|f1| < |f2|$.

(2) If $|f1| = |f2| = m$, let $k \leq m$ be the largest natural number such that $f1(k) \neq f2(k)$. Then $f1 < f2$ if $f1(k) < f2(k)$.

Note that the ordering on $\{f\}$ is well founded.

Theorem T1. For any **P**, *any* $Ai \to xi$, *and any atom* q, *conditions* (a) *and* (b) *below imply condition* (c) *below*:

(a) $\mathbf{P} + \{Ai \to xi \mid i \leq r\}?q = 1$,

(b, i) $\mathbf{P} + Ai?xi = 1$,

(c) $\mathbf{P}?q = 1$.

PROOF. By induction on the depth m of the success tree of (a) (see definition D5 of Section 3 of paper I for the notion of a tree of a successful computation) and the complexity of the set $\{Ai \to xi\}$. Let ni be the complexity of $(Ai \to xi)$, and let f be the complexity function of $\{Ai \to xi\}$.

The induction is on the lexicographic ordering of the pairs (f, m).

Case 1: $m = 1$, ni arbitrary. In this case we must simply have $q \in \mathbf{P} + \{Ai \to xi\}$. So either $q \in \mathbf{P}$, or for some i, $Ai = 0$ (i.e. Ai does not exist) and $q = xi$. In either case $\mathbf{P}?q = 1$.

Case 2: $ni = 1$, m arbitrary.

Subcase 2a: q unifies with a $(\bigwedge_{j=1}^{k}(Bj \to yj) \to q) \in \mathbf{P}$. [Note: We shall not use the fact that $ni = 1$ in this subcase]. Since we are dealing with a success tree of (a), we get that for each j

(a, j) $[\mathbf{P} + Bj] + \{Ai \to xi\}\,?yj = 1.$

We also get (b, i, j) from (b) and lemma L1 of section 3 paper I:

(b, i, j) $[\mathbf{P} + Bj] + Ai\,?xi = 1,$

and the success tree of (a, j) has depth $m - 1$ and the success tree for (b, i, j) has depth ni; hence by the induction hypothesis

(c, j) $\mathbf{P} + Bj\,?yj = 1$

for each j, with a success tree (Oj, Tj, Vj) of depth $\leq m$. Thus $\mathbf{P}?q$ succeeds with a success tree of the form

$$\begin{tikzcd} & 0\mathbf{P}?q \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\ (O1, T1, V1)\cdots & (Oj, Tj, Vj)\cdots & (Ok, Tk, Vk) \end{tikzcd}$$

Subcase 2b: q unifies with $(Ai \to xi)$ for some i. Assume $i = 1$. Since for all i, $ni = 1$, we have $P + Ai\,?xi$ succeeds in one step. This means that $xi \in \mathbf{P}$. Since q unifies with xi, this means $xi = q$, and hence $P\,?q = 1$ with success tree of depth $ni \leq 1 - 1 + \max(ni)$.

Case 3a: $m > 1$ and some $ni > 1$ and q unifies with a $\bigwedge_{j=1}^{k}(Bj \to yj) \to q \in \mathbf{P}$. In this case the proof is the same as in subcase 2a; we have not used the fact that $ni = 1$ in the proof of subcase 2a.

Case 3b: $m > 1$, some $ni \geq 1$, and q unifies with $A1 \to x1$. This means that $x1 = q$ and $A = \bigwedge_{j=1}^{k}(Bj \to yj)$, i.e., the clause $A1 \to x1$ is $\bigwedge_{j=1}^{k}(Bj \to yj) \to q$.

The assumptions of the theorem are therefore:

(a) $\mathbf{P} + \bigwedge_{j=1}^{k}(Bj \to yj) \to q + \{Ai \to xi \mid i \geq 2\}\,?q = 1,$

(b, 1) $\mathbf{P} + \bigwedge_{j=1}^{k}(Bj \to yj)?q = 1,$

(b, i) $\mathbf{P} + Ai\,?xi = 1,\ i = 2, \ldots, r.$

(a) succeeds with a tree of depth m, and (b, i) have complexity ni. Since q unifies with $\bigwedge_{j=1}^{k}(Bj \to yj) \to q$ in (a) and the success tree of (a) follows the unification process, we get that the following holds with success trees of depth $m - 1$, for each $j = 1, \ldots, r$:

(a, j) $[\mathbf{P} + Bj] + \bigwedge_{j=1}^{k}(Bj \to yj) \to q + \{Ai \to xi \mid i \geq 2\}\,?yj = 1.$

The following also succeeds by Lemma L1 of Section 3, paper I:

(b, 1, j) $[\mathbf{P} + Bj] + \bigwedge_{j=1}^{k}(Bj \to yj)?q = 1$

(b, i, j) $[\mathbf{P} + Bj] + Ai\,?xi = 1$ for $i = 2, \ldots, r.$

Note that the complexity r remains unchanged.

By the induction hypothesis we get

$(c, j)\mathbf{P} + Bj\,?\,yj = 1$

We also have $(b, 1)$, namely

$$\mathbf{P} + \bigwedge_{j=1}^{k} (Bj \to yj)\,?\,q = 1$$

with complexities $mj \leq n1 - 1$.

We can now use the induction hypothesis for the theorem. The case (f, m) is reduced to the case (f', m'), where f' is the complexity of the set $\{Bj \to yj\}$. Now f' is smaller than f, since $f'(n1) = 0$ and $f(n1) \geq 1$, $n1$ being the complexity of $\bigwedge_{j=1}^{k}(Bj \to yj) \to q$.

From the induction hypothesis we get $P\,?\,q = 1$. This proves Theorem $T1$. □

Corollary T2. $\mathbf{P}\,?\,(A \to q) = 1$ *and* $\mathbf{P}\,?\,A + 1$ *implies* $\mathbf{P}\,?\,q = 1$.

PROOF. Let

(1) $A = \bigwedge_{j=1}^{m}(Bj \to yj)$.

We have

(a) $\mathbf{P} + \{Bj \to yj\}\,?\,q = 1$,

(b) $\mathbf{P} + Bj\,?\,yj = 1$ for each j,

and hence by Theorem T1 we get $P\,?\,q = 1$. □

Corollary T3.

(a) *For any* $\mathbf{P}, A, G$,

$\mathbf{P}\,?\,A = 1$ *and* $\mathbf{P} + A\,?\,G = 1$ *imply* $\mathbf{P}\,?\,G = 1$.

(b) $\mathbf{P}$ *need not be finite in* (a).

PROOF. (a): G has the form $B \to q$ and hence we have

$\mathbf{P}\,?\,A = 1$ and $\mathbf{P} + A + B\,?\,q = 1$,

implying by the previous corollary that $\mathbf{P} + B\,?\,q = 1$ and hence $\mathbf{P}\,?\,G = 1$.

(b): Since all computations are finite, there exists a large enough finite subset $\mathbf{P0} \subseteq \mathbf{P}$ such that $\mathbf{P0}\,?\,A = 1$ and $\mathbf{P0} + A\,?\,G = 1$, and thus $\mathbf{P0}\,?\,G = 1$, and hence $\mathbf{P}\,?\,G = 1$. □

We saw in paper I that N-PROLOG is sound for classical logic, namely, that $\mathbf{P}\,?\,A = 1$ implies $P \vdash A$ in classical logic. What about the converse? Here is what we can get:

Definition D2. Let A be a goal. Then the complement of A, denoted by Cop(A), is the following set of clauses:

$$\text{Cop}(A) = \{A \to x \mid x \text{ any atom of the language}\}$$

Lemma L1. For any database $\mathbf{P}$ *and goal* G *such that* $\mathbf{P} \supseteq \mathrm{Cop}(G)$ *and for any goal* A, *conditions* (a) *and* (b) *below imply condition* (c):

(a) $\mathbf{P} + A?G = 1$,

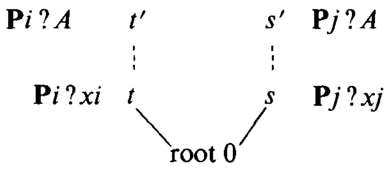
(b) $(\mathbf{P} \cup \mathrm{Cop}(A))?G = 1$,

(c) $\mathbf{P}?G = 1$.

PROOF. Since $\mathbf{P}\cup\mathrm{Cop}(A)?G = 1$ and computations are finite, only a finite number of the elements of $\mathrm{Cop}(A)$ are used in the computation. Assume then that

(b′) $\mathbf{P} + (A \to x1) + \cdots + (A \to xn)?G = 1$.

We now indicate how to construct a successful computation tree (the notion of computation tree was defined in Definition D5, Section 3, paper I) for $\mathbf{P}?G$, by induction on the number of nested uses of $(A \to xi)$ in the tree. Consider the successful computation tree for (b′): Go up the tree until you meet the *last* nodes in which any $(A \to xi)$ is used in the computation. If no such nodes exist, then clearly $P?G = 1$, since $A \to xi$ are not used. Otherwise we have the following situation:



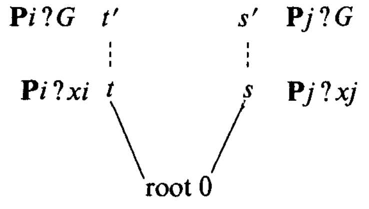
Recall xi, xj are atomic. At nodes t or at s the current subgoal is xi from the current database $\mathbf{P}i$.

$\mathbf{P}i$ contains $\mathbf{P} \cup \{A \to x1, \ldots, A \to xn\}$, the clause $A \to xi$ is used here for the last time, the next subgoal is $\mathbf{P}i?A$, and in the computation of $\mathbf{P}i?A$ *no* $(A \to xi)$ is used. This subgoal succeeds, of course, since we are dealing with a success tree. We thus have, since $(A \to xi)$ is no longer used,

$$\mathbf{P}i - \{A \to xi \mid i = 1, \ldots, n\}?A = 1.$$

Also, since $\mathbf{P} + A?G = 1$, we have $\mathbf{P}i - \{A \to xi \mid i = 1, \ldots, n\} + A?G = 1$. Hence by Corollary T3, we have $\mathbf{P}i - \{A \to xi \mid i = 1, \ldots, n\}?G = 1$, and so we have $\mathbf{P}i?G = 1$, *without using* $A \to xi$, for any i.

Therefore there exists a success tree Ti for G from Pi, without using $\{A \to xi\}$. We now eliminate the last use of $A \to xi$ in the success tree of (b′). We do this as in the diagram



Instead of unifying with $A \to xi$ taken from Pi, unify instead with $(G \to xi) \in \mathbf{P} \subseteq \mathbf{P}i$. [Recall that $\mathrm{Cop}(G) \subseteq P$.]

At the point t', ask the goal $\mathbf{P}i\,?G$ and follow the successful tree Ti. $\{A \to xi\}$ are not used any more. We have thus eliminated one nested use of $(A \to xi)$, and we can use the induction hypothesis and get that $\mathbf{P}?G = 1$. Thus Lemma L1 is proved. □

Theorem T3 (First completeness theorem for propositional N-PROLOG). *For any* **P** *and any A, (a) is equivalent to (b) below:*

(a) $\mathbf{P} \vdash A$ *in classical logic.*

(b) $(\mathbf{P} \cup Cop(A))?A = 1$ *in N-PROLOG.*

PROOF.

(1) Show (b) implies (a): Assume

$$(\mathbf{P} \cup \mathrm{Cop}(A))?A = 1.$$

Then by the soundness of N-PROLOG we get that $\mathbf{P} \cup \mathrm{Cop}(A) \vdash A$ in classical logic. Since the proof is finite, there is a finite set of the form

$$\{A \to x1, \ldots, A \to xn\}$$

such that

$$\mathbf{P} + (A \to x1) + \cdots + (A \to xn) \vdash A.$$

We show by induction on n that $\mathbf{P} \vdash A$.

Case $n = 1$:

$$\mathbf{P} + (A \to x1) \vdash A$$

Hence $\mathbf{P} \vdash (A \to x1) \to A$ by the deduction theorem. But since Pierce's law

$$((y \to x) \to y) \to y \text{ for any } x, y$$

is a tautology of classical logic, we get

$$\mathbf{P} \vdash ((A \to x1) \to A) \to A.$$

By modus ponens,

$$\mathbf{P} \vdash A.$$

Case $n > 1$: We assume

$$\mathbf{P} + (A \to x1) + \cdots + (A \to xn) \vdash A.$$

Hence

$$\mathbf{P} + (A \to x1) + \cdots + (A \to x(n-1)) \vdash (A \to xn) \to A$$

Again by Pierce's law

$$\mathbf{P} + (A \to x1) + \cdots + (A \to x(n-1)) \vdash A,$$

and by the induction hypothesis

$$\mathbf{P} \vdash A.$$

The above concludes the proof that (b) implies (a).

(2) Show that (a) implies (b): We prove that if

$$\mathbf{P} \cup \mathrm{Cop}(A)?A \neq 1$$

then $\mathbf{P} \nvdash A$ in classical logic. Let

$$\mathbf{P}0 = \mathbf{P} \cup \mathrm{Cop}(A).$$

We define a sequence of databases $\mathbf{P}n$, $n = 1,2,\ldots,$ as follows: Let $B1, B2, B3,\ldots$ be an enumeration of all goals of the language. Assume $\mathbf{P}(n-1)$ has been defined. We define $\mathbf{P}n$. If $\mathbf{P}(n-1) + Bn?A \neq 1$, let $\mathbf{P}n = P(n-1) + Bn$. [Remember if Bn is a conjunction, we add all the conjuncts separately to $\mathbf{P}(n-1)$.] Otherwise by Lemma L1,

$$\mathbf{P}(n-1) \cup \mathrm{Cop}(Bn)?A \neq 1.$$

So let

$$\mathbf{P}n = \mathbf{P}(n-1) \cup \mathrm{Cop}(Bn).$$

Let $\mathbf{P}^* = UnPn$. Clearly

$$\mathbf{P}^*?A \neq 1.$$

Define an assignment of truth values $\mathbf{h}$ on the atoms of the language by

$$\mathbf{h}(x) = \text{true iff } \mathbf{P}^*?x = 1.$$

Lemma L2. For any B

$$\mathbf{h}(B) = \text{true iff } \mathbf{P}^*?B = 1.$$

PROOF. By induction on B.

(a) For atoms this is the definition.

(b) The case of conjunction is immediate.

(c) We check the case of $(C \to q)$, q atomic.

(c1) If $\mathbf{P}^*?(C \to q) = 1$, then if $\mathbf{P}^*?C = 1$, then by Lemma L1 also $\mathbf{P}^*?q = 1$. This means, by the induction hypothesis, that if $\mathbf{h}(C) = \text{true}$ then also $\mathbf{h}(q) = \text{true}$, and hence $\mathbf{h}(C \to q) = \text{true}$.

(c2) If $\mathbf{P}^*?(C \to q) \neq 1$, then $\mathbf{P}^* + C?q \neq 1$. Hence by definition $\mathbf{h}(q) = \text{false}$, since certainly $\mathbf{P}^*?q \neq 1$.

If $\mathbf{h}(C) = \text{true}$, then we are finished, since this makes $\mathbf{h}(C \to q) = \text{false}$. We now show that indeed $\mathbf{h}(C) = \text{true}$ by showing that $\mathbf{h}(C) = \text{false}$ leads to a contradiction. Assume that $\mathbf{h}(C) = \text{false}$. Then by the induction hypothesis $\mathbf{P}^*?C \neq 1$. Thus certainly $C \notin \mathbf{P}^*$ (if $C \in P^*$ then $\mathbf{P}^*?C = 1$). We have that $C = Bn$, for some n in the enumeration of wffs, and so since $C \notin P^*$, by construction $\mathrm{Cop}(C) \subseteq P^*$, in particular $(C \to q) \in \mathbf{P}^*$. Thus $\mathbf{P}^*?(C \to q) = 1$, which contradicts the assumption of our case (c2). Thus case (c) is proved and lemma L2 is proved. □

We can now prove direction 2 of Theorem T3. Since $\mathbf{P}^*?A \neq 1$, we get $\mathbf{h}(A) = \text{false}$. Thus $\mathbf{h}$ is an assignment of truth values such that for any $B \in \mathbf{P}^*$, and certainly for any $B \in \mathbf{P} \cup \mathrm{Cop}(A)$, $\mathbf{h}(B) =$ true (since $B \in \mathbf{P}^*$ implies $P^* ?B = 1$) and $\mathbf{h}(A) =$ false. This means that $\mathbf{P} \cup \mathrm{Cop}(A) \vdash A$ in classical logic. Thus (a) of theorem T3 implies (b), as we showed that not (b) implies not (a).

This proves Theorem T3. □

2. PROPOSITIONAL N-PROLOG, INTUITIONISTIC PROPOSITIONAL LOGIC, AND CLASSICAL PROPOSITIONAL LOGIC

We must present classical logic in a form ready for comparison with our N-PROLOG. We begin with the propositional calculus and use formulation with the connectives $\wedge$, $\rightarrow$, and $\mathbf{f}$ ($\mathbf{f}$ for falsity). It is well known that $\wedge$ and $\vee$ can be defined using $\rightarrow$ and $\mathbf{f}$. In fact $\neg$ can also be defined using $\rightarrow$ and $\mathbf{f}$, but we are essentially taking $\wedge$ and $\rightarrow$ as primitives, and defining the intuitionistic resolution for this fragment first. We then add $\mathbf{f}$ as a simple extension.

Notice by contrast that N-PROLOG is really $\{\wedge, \rightarrow\}$ based, while the most widespread resolutions for classical logic are $\{\vee, \neg\}$ based.

Definition D1. Axioms for intuitionistic logic and for classical logic.

Axioms:

(n1) $A \rightarrow (B \rightarrow A)$.

(n2) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.

(n3) $A \wedge B \rightarrow A$,
$A \wedge B \rightarrow B$.

(n4) $A \rightarrow (B \rightarrow A \wedge B)$,

(n5) $((A \rightarrow B) \rightarrow A) \rightarrow A$ (Pierce's law).

(n6) $\mathbf{f} \rightarrow A$.

Rules:

$$\frac{A,\ A \rightarrow B}{B}$$

The positive intuitionistic propositional calculus $\mathbf{I}+$ (with negation and without disjunction) is defined using the axioms and rules except (n5). If we take all the axioms and rules, including (n5), we get the classical propositional calculus $\mathbf{C}$. In $\mathbf{I}+$, $\neg A$ is definable as $A \rightarrow \mathbf{f}$. $A \vee B$ however is not definable.

In $\mathbf{C}$, $A \vee B$ is definable as $(A \rightarrow B) \rightarrow B$, or equivalently as $(B \rightarrow A) \rightarrow A$, or equivalently as $((A \rightarrow \mathbf{f}) \wedge (B \rightarrow \mathbf{f})) \rightarrow \mathbf{f}$.

Theorem T1 (Deduction theorem). In both logics,

$$A, B \vdash C \textit{ iff } A \vdash B \rightarrow C.$$

Theorem T2 (Conjunction theorem). In both logics,

$$A \vdash B \wedge C \textit{ iff } A \vdash B \textit{ and } A \vdash C.$$

Theorem T3. In both logics,

$$\vdash (A \to (B \to C)) \leftrightarrow (A \wedge B \to C),$$

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

Definition D2.

(a) A *model* is any function $\mathbf{h}$ assigning values 0, 1 to the atoms. We require that $\mathbf{h}(\mathbf{f}) = 0$.

(b) Define $\leq$ on models by
$\mathbf{h} \leq \mathbf{h}'$ iff for all atoms x, $\mathbf{h}(x) \leq \mathbf{h}'(x)$.

(c) A kripke model is any partially ordered set $(T, \leq, \mathbf{h0})$ of models with $\mathbf{h0} \leq \mathbf{h}$ for all $\mathbf{h} \in T$ and $\mathbf{h0} \in T$.

(d) Given kripke model $(T, \leq, \mathbf{h0})$, define the function $\mathrm{Val}(\mathbf{h}, A)$, where $\mathbf{h} \in T$.

(d1) $\mathrm{Val}(\mathbf{h}, q) = 1$ if $\mathbf{h}(q) = 1$.

(d2) $\mathrm{Val}(\mathbf{h}, A \wedge B) = 1$ iff
$\mathrm{Val}(\mathbf{h}, A) = 1$ and $\mathrm{Val}(\mathbf{h}, B) = 1$.

(d3) $\mathrm{Val}(\mathbf{h}, A \to B) = 1$ iff
$\forall \mathbf{h}' \in T$ ($\mathbf{h} \leq \mathbf{h}'$ and $\mathrm{Val}(\mathbf{h}', A) = 1$ imply $\mathrm{Val}(\mathbf{h}', B) = 1$).

(d4) A is semantically valid in kripke models if for any $(T, \leq, \mathbf{h0})$ we have $\mathbf{h0}(A) = 1$.

(d5) A is a classical kripke tautology if for any kripke model of the form $(\{\mathbf{h}\}, \leq, \mathbf{h})$, $\mathbf{h}(A) = 1$.

Theorem T4 (Completeness theorem).

(a) $\vdash_{\mathrm{I}} A$ *iff A is valid in all kripke models.*

(b) $\vdash_{\mathrm{C}} A$ *iff A is a classical kripke tautology.*

The above semantics in kripke models is the temporal interpretation of the intuitionistic implication. We imagine a branching future, where there are several possibilities for what may happen. $A \to B$ is valid *now* if we *now* have a commitment that no matter what happens, if A becomes true, B will be true.

Thus imagine the following diagram:

$$\begin{tikzcd}
\mathbf{h1} & \mathbf{h2} & \mathbf{h3} \\
& \mathbf{h0} = \text{now} \arrow[ul, dash] \arrow[u, dash] \arrow[ur, dash] &
\end{tikzcd}$$

$\mathbf{h0}$ is now, the present. $\mathbf{h1}$, $\mathbf{h2}$, $\mathbf{h3}$ are alternative possible future events;

$$\mathbf{h0}(A \to B) = \text{true}$$

means that we know now that at any future event $\mathbf{h}i$, if A is true [i.e. $\mathbf{h1}(A) = 1$], then B is true [i.e. $\mathbf{h}i(B) = 1$].

For example, an insurance policy against fire ($A =$ "my house burns down") assures me that $B =$ "I get the house rebuilt". Thus $A \to B$ is what the insurance policy says and is true from the moment the policy is bought. In fact, the bank may write to me and say that my mortgage will be approved (M) provided I have such an insurance policy. Thus we have the true statement $(A \to B) \to M$ from the bank.

The reading of the computation rule

$$\mathbf{P}?(A \to B) \text{ iff } \mathbf{P} + A?B$$

is simply the "testing" of the commitment $A \to B$. We assume that A has happened and test for B. We test B from $\mathbf{P} + A$. This means that there is a further assumption in our temporal interpretation of $\to$. The assumption is that whatever is true now continues to be true in the future. This is the requirement $\mathbf{h} \leq \mathbf{h}'$ whenever $\mathbf{h}'$ is in the future of $\mathbf{h}$. This assumption is possible because we have no negation.

We shall deal with negation as failure in the next section and we shall see how problematic it is.

Definition D3 (Clauses).

(a) An atom or $\mathbf{f}$ is a clause.

(b) If Ai are clauses and q is an atom then $\bigwedge Ai \to q$ is a clause.

Theorem T5. For both logics, any wff A is equivalent to a conjunction of clauses A'.

Definition D4. The following are the quantifier rules which when added to $\mathbf{I}+$ *or* $\mathbf{C}$ *turn them into quantificational logic*:

(1) $\forall x A(x) \to A(y)$,

(2) $A(y) \to \exists x A(x)$,

(3) $\dfrac{A(x) \to B}{\exists x A(x) \to B}$,

(4) $\dfrac{B \to A(x)}{B \to \forall x A(x)}$,

where x is not free in B.

*Theorem T6 (Soundness of propositional N-*PROLOG *relative to* $\mathbf{I}+$*).*

(a) $\mathbf{P}?A = 1$ *implies* $\mathbf{P} \vdash_{\mathrm{I}+} A$.

(b) *If* $\mathbf{P} \subseteq \mathbf{P}'$ *and* $\mathbf{P}?A = 1$ *then* $\mathbf{P}'?A = 1$.

(c) *If* $A \in \mathbf{P}$ *then* $\mathbf{P}?A = 1$.

PROOF. By induction on the computation of A.

(a): We follow the rules of computation. The conjunction theorem for $\mathbf{I}+$ ensures the soundness of the rule for $\wedge$. The deduction theorem for $\mathbf{I}+$ ensures the soundness of the rule for $\to$.

The soundness of the rule for atoms is obtained by induction on the length of the successful computation of the goal. We have $\mathbf{P}?q = 1$ iff either:

(1) $q \in \mathbf{P}$, in which case clearly $\mathbf{P} \vdash q$; or

(2) for some $A \to q \in \mathbf{P}$, $\mathbf{P}?A = 1$, in which case, by the induction hypothesis on the length of computation, we have $\mathbf{P} \vdash A$, and since $A \to q \in \mathbf{P}$, we get $\mathbf{P} \vdash q$.

(b) and (c) are proved by induction on A and on the length of the successful computation.

*Theorem T7 (Completeness of Propositional N-*PROLOG). *If* $\mathbf{P}0 \vdash_{\mathrm{I}+} B0$ *then*

$\mathbf{P}0?B0 = 1$.

PROOF. Assume $\mathbf{P}0?B0 \neq 1$ (i.e., $B0$ does not succeed, either because it finitely fails or because it loops). We shall show that $\mathbf{P}0 \nvdash_{\mathrm{I}+} B0$.

Let S be the set of all finite databases $\mathbf{P}$. Define a function $\mathrm{Val}(\mathbf{P}, q)$ for $\mathbf{P} \in S$ and atomic q by letting

(a) $\mathrm{Val}(\mathbf{P}, q) = 1$ iff (def.) $\mathbf{P}?q = 1$. From Theorem T6 (soundness) we know that if $\mathrm{Val}(\mathbf{P}, q) = 1$ and $\mathbf{P} \subseteq \mathbf{P}'$ then $\mathrm{Val}(\mathbf{P}', q) = 1$.

Extend the definition of Val to any wff B by induction on the structure of B, as follows:

(b) $\mathrm{Val}(\mathbf{P}, B1 \wedge B2) = 1$ iff (def.) $\mathrm{Val}(\mathbf{P}, Bi) = 1$, $i = 1, 2$.

(c) $\mathrm{Val}(\mathbf{P}, A \to B) = 1$ iff (def.) for all $\mathbf{P}' \supseteq \mathbf{P}$, if $\mathrm{Val}(\mathbf{P}', A) = 1$ then $\mathrm{Val}(\mathbf{P}', B) = 1$.

Lemma L1. For any clause B and $\mathbf{P}$, *Val*$(\mathbf{P}, B) = 1$ *iff* $\mathbf{P}?B = 1$.

PROOF. By induction on B.

(a) For B atomic the theorem is the definition.

(b) $\mathrm{Val}(\mathbf{P}, B1 \wedge B2) = 1$ iff (def.) $\mathrm{Val}(\mathbf{P}, B1) = \mathrm{Val}(\mathbf{P}, B2) = 1$ iff (induction) $\mathbf{P}?B1 = 1$ and $\mathbf{P}?B2 = 1$ iff (def.) $\mathbf{P}?B1 \wedge B2 = 1$.

(c) B has the form $A \to q$.

(c1) Assume $\mathbf{P}?(A \to q) = 1$. We show that $\mathrm{Val}(\mathbf{P}, A \to q) = 1$. Let $\mathbf{P}' \supseteq \mathbf{P}$ and $\mathrm{Val}(\mathbf{P}', A) = 1$. By the induction hypothesis

$\mathbf{P}'?A = 1$.

Since $\mathbf{P}?A \to q = 1$ and $\mathbf{P}' \supseteq \mathbf{P}$, we also have

$\mathbf{P}'?A \to q = 1$

and so

$\mathbf{P}' + A?q = 1$.

We therefore have

$\mathbf{P}' + A?q = 1$,

$\mathbf{P}'?A = 1$,

and hence, by Theorem T5, $\mathbf{P}'?q = 1$ and therefore $\mathrm{Val}(\mathbf{P}', q) = 1$. We have thus shown that for any $\mathbf{P}' \supseteq \mathbf{P}$, if $\mathrm{Val}(\mathbf{P}', A) = 1$ then $\mathrm{Val}(\mathbf{P}', q) = 1$. This implies, by definition, that $\mathrm{Val}(\mathbf{P}, A \to q) = 1$.

(c2) Assume $\mathbf{P}?A \to q \neq 1$. Then

$\mathbf{P} + A?q \neq 1$;

hence $\mathrm{Val}(\mathbf{P} + A, q) \neq 1$. But $\mathbf{P} + A?A = 1$ by Theorem T1, and by the induction hypothesis $\mathrm{Val}(\mathbf{P} + A, A) = 1$. Hence we found a $\mathbf{P}' \supseteq \mathbf{P}$, namely, $\mathbf{P}' = \mathbf{P} + A \supseteq \mathbf{P}$, such that $\mathrm{Val}(\mathbf{P}', A) = 1$ and $\mathrm{Val}(\mathbf{P}', q) = 0$. Thus by definition $\mathrm{Val}(\mathbf{P}, A \to q) = 0$.

This completes the induction and proves Lemma L1. □

Corollary T8.

(a) $\mathrm{Val}(\mathbf{P0}, B0) = 0$.

(b) $\mathrm{Val}(\mathbf{P0}, A) = 1$ *for* $A \in \mathbf{P0}$.

PROOF. By Lemma L1, since $\mathbf{P0}?B0 \neq 1$ and $\mathbf{P0}?A = 1$ for $A \in \mathbf{P0}$. □

The above construction of $(S, \subseteq, \mathbf{P0})$ and of the function Val shows that if $\mathbf{P0}?B0 \neq 1$, then there exists a kripke model [namely $(S, \subseteq, \mathbf{P0})$] such that, by Corollary T8, **P0** is valid in the model and $B0$ is not valid in the model. Using the completeness theorem T4, we can deduce that $\mathbf{P0} \nvdash_{\mathrm{I}+} B0$. However, since Theorem T4 is not proved in this paper, we prove Lemma L2 below and show directly that $\mathbf{P0} \nvdash_{\mathrm{I}+} B0$.

Lemma L2. For any **P** *and* B, *if* $P \vdash_{\mathrm{I}+} B$ *then* $Val(\mathbf{P}, B) = 1$.

PROOF. By induction on the length of the **I** + proof of B from **P**. We have $\mathbf{P} \vdash_{\mathrm{I}+} B$ iff, by definition, there is a sequence of wffs $D0, D1, \ldots, Dn = B$ such that each Di is either from **P** or a substitution instance of an **I** + axiom or is obtained from Dj, Dk, $j, k < i$, by the rule of modus ponens.

If we verify (a), (b), (c) below, we get a proof of Lemma L2:

(a) If $D \in \mathbf{P}$ then $\mathrm{Val}(\mathbf{P}, D) = 1$

(b) If D is an instance of an axiom of $I+$ then $\mathrm{Val}(\mathbf{P}, D) = 1$.

(c) If $\mathrm{Val}(\mathbf{P}, D1) = 1$ and $\mathrm{Val}(P, D1 \to D2) = 1$ and $\mathrm{Val}(\mathbf{P}, D2) = 1$.

(a) follows from Corollary T8. (b) and (c) can be verified directly. □

PROOF OF THEOREM T7. We assume $\mathbf{P0}?B0 \neq 1$ and we have constructed the set S with $\mathbf{P0} \in S$ and $\mathrm{Val}(\mathbf{P0}, B0) = 0$. Then by Lemma L2 we cannot have that $P0 \vdash_{\mathrm{I}+} B0$.

REMARK R2. Theorems T6 and T7 show that our computation characterizes **I** + . $\mathbf{P}?A$ succeeds iff $\mathbf{P} \vdash_{\mathrm{I}+} A$. Later in this paper we shall characterize **C** +, the classical propositional logic without negation, by adding to N-PROLOG a special rule, called the *restart rule*.

Example E1. If $\mathbf{P} \vdash_{\mathrm{I}+} B$, then the computation $\mathbf{P}?B$ either loops or finitely fails. Here are some examples:

(a) $(q \to q)?q$ loops.

(b) $\varnothing\ ?q$ finitely fails.

(c) $(q \to a) \to q?q$ finitely fails.

Definition D5 (Propositional NR-PROLOG: propositional *N*-PROLOG with the restart rule). *Let NR-PROLOG be the extension of N-PROLOG with the following restart rule* **RS**:

RS: *In the course of the computation if a query of the form* $\mathbf{P}?q$, *q atomic, we may happen to be trying to succeed with a current subgoal* $?a$, *a atomic. If there are no clauses with heads a, it is permissible to continue the computation with the original goal* $?q$, *instead of the subgoal* $?a$, *and success of* $?q$ *will be considered a success of* $?a$.

The above rule is a strengthening of the computation procedures because in the course of the computation the database **P** increases, and so by replacing the current $?a$ by $?q$, we may now succeed.

To give a formal definition of a successful computation tree in NR-PROLOG of a goal $G0$ from a database $P0$, we add a clause to Definition D5 of Section 3 in paper I. (Definition D5 defines the notion of a successful computation tree in N-PROLOG.) The extra clause is (g3) for the current atomic subgoal q computed at node t from the current database **P**:

(g3) t has exactly one immediate successor s in the tree with $V(s) = (\mathbf{P}, G0)$.

Example E2.

(1) We saw that in N-PROLOG $(q \to a) \to q ? q$ fails. In NR-PROLOG the above query succeeds. Let us check the NR-computation:

$\{(q \to a) \to q\} ? q.$

We unify with the first clause and ask

$\{(q \to a) \to q, q\} ? a.$

We have no clauses with a as head; we are allowed by rule **RS** to ask q again. If q succeeds now it will be considered that $?a$ succeeded:

$\{(q \to a) \to q, q\} ? q.$

The query now succeeds. Hence the entire computation succeeds.

(2) $(q \to a) \to q ? q \wedge (q \to a)$ fails in NR-PROLOG. Let us try it. First try q:

$\{(q \to a) \to q\} ? q.$

q will succeed; now ask $(q \to a)$. We therefore add q and ask a; we have

$\{(q \to a) \to q, q\} ? a.$

There are no clauses with heads a. We can ask instead of a either $(q \to a)$ or the original query, namely, $q \wedge (q \to a)$. In either case we shall fail. Of course we must not keep on asking the original query again and again if we do not want to loop.

Theorem T9 (Soundness and completeness of Propositional NR-PROLOG). $P ? G = 1$ *in NR-PROLOG iff* $P \vdash_C G$.

PROOF.

(a) Assume $\mathbf{P} \vdash_C G$ and show $\mathbf{P}?G = 1$ in NR-PROLOG. By Theorem T3 we have $(\mathbf{P} \cup \mathrm{Cop}(G))?G = 1$ in N-PROLOG, where $\mathrm{Cop}(G) = \{G \to x \mid x$

atomic}. Therefore, there exists a successful computation tree $(T, \leq, 0, V)$ of G from $P \cup \mathrm{Cop}(G)$ in N-PROLOG. We now show that there exists a successful computation tree in NR-PROLOG of G from **P**. In fact we just modify the tree $(T, \leq, 0, V)$ itself. Let $V1$ be defined on $(T, \leq, 0)$ as follows:

If $V(t) = (P(t), G(t))$ then let $V1(t) = (\mathbf{P}(t) - \mathrm{Cop}(G), G(t))$.
We claim that $(T, \leq, 0, V1)$ is a successful computation tree of G from **P** in NR-PROLOG. By Definition D5, Section 3 of paper I, each node of the tree must satisfy one of the conditions of that definition. By taking $\mathrm{Cop}(G)$ out of the data we may be violating condition (g2) of the definition, namely, we may have

$$\begin{array}{ll} t & (\mathbf{P}(t), x),\ x \text{ atomic} \\ \vdots & \\ s & (\mathbf{P}(t), G) \end{array}$$

and the justification for this node is the fact that

$(G \rightarrow x) \in \mathrm{Cop}(G) \subseteq \mathbf{P}(t)$.

In the new tree $(G \rightarrow x)$ is taken out. However, the new tree is supposed to be a tree in NR-PROLOG. Since G is the original goal, the above node is justified by clause (g3) of Definition D5, namely, the restart rule.

(b) Assume $\mathbf{P}?G = 1$ in NR-PROLOG, and show that $\mathbf{P} \vdash_C G$. In this case we modify the computation tree of G from **P** in NR-PROLOG in the other direction. Assume $(T, \leq, 0, V1)$ is a successful computation tree of G from **P** in NR-PROLOG. Let V be defined as follows:

If $V1(t) = (\mathbf{P}(t), G(t))$ then let $V(t) = (\mathbf{P}(t) \cup \mathrm{Cop}(G), G(t))$. We claim $(T, \leq, 0, V)$ is a successful computation tree of G from **P** in N-PROLOG. We have to show what happens to nodes of the tree of the form:

$$\begin{array}{ll} t & (\mathbf{P}(t), x),\ x \text{ atomic} \\ \vdots & \\ s & (\mathbf{P}(t), G) \end{array}$$

which are justified by the restart rule in $(T, \leq, 0, V1)$. In N-PROLOG we do not have the restart rule, but since in $(t, \leq, 0, V)$ the nodes are

$$\begin{array}{ll} t & (\mathbf{P}(t) \cup \mathrm{Cop}(G), x) \\ \vdots & \\ s & (\mathbf{P}(t) \cup \mathrm{Cop}(G), G) \end{array}$$

the justification is the fact that $(G \rightarrow x) \in \mathrm{Cop}(G)$.

Theorem T9 is proved. □

3. THE COMPLETENESS OF QUANTIFICATIONAL QN-PROLOG

We want to study the soundness and completeness of quantificational QN-PROLOG relative to the intuitionistic predicate logic. We shall also formulate the restart rule for quantificational QN-PROLOG and study its soundness and completeness relative to the classical predicate logic. First recall Definition D4 of the previous section.

It lists the quantifier atoms to be added to the axioms of propositional intuitionistic or classical logic in order to obtain the corresponding predicate logic. There is nothing special about these axioms: they are the same standard quantifier axioms used for many logics, to pass from the propositional axiomatic formulation to the predicate axiomatic formulation.

The two predicate logics, however (i.e. intuitionistic and classical) have different quantificational properties. This is due to the effect of the different propositional properties of the systems. One such property is the so-called *existential property*. This property is of importance to us. We explain it by an example.

Example E1. Consider an Herbrand universe with the only constants a and b, and the unary predicate $\mathbf{P}(x)$. Then in classical predicate logic the following holds:

(a) $\vdash \exists x(P(a) \vee P(b) \rightarrow P(x))$,

but in intuitionistic predicate logic the above is *not* provable.

In classical logic, we can push the existential quantifier inside the formula and get the equivalent formula

(b) $\vdash P(a) \vee P(b) \rightarrow \exists x P(x)$.

Obviously (b) is provable. (b) is also provable in intuitionistic logic, but in intuitionistic logic (a) and (b) are *not* equivalent. (a) is stronger than (b); it proves (b).

Although (a) is provable in classical logic, there *does not* exist a substitution θ for x such that in classical logic

(a′) $P(a) \vee P(b) \rightarrow P(x)\theta$.

$\theta(x)$ can be either $x = a$ or $x = b$, and in neither case is (a′) provable. In intuitionistic logic, however, whenever $\vdash \exists x A(x)$, for some wff A, then there exists a θ into the Herbrand universe of A such that

$$\vdash A(x)\theta.$$

This is a serious, computationally meaningful difference between the two logics.

In intuitionistic logic it is also true that

$$\vdash A \vee B \text{ iff } \vdash A \text{ or } \vdash B$$

for A, B closed wffs.

Theorem T1 (Existential property). If $\vdash \exists x A(x)$ in intuitionistic logic, then for some θ over the Herbrand universe of A we have

$$\vdash A(x)\theta.$$

Note that the analog of this theorem for classical logic is a version of Herbrand's theorem, namely

Theorem T2 (Herbrand). If $\vdash \exists x A(x)$ in classical logic, then for some θi, $i = 1, \ldots, n$,

$$\vdash A\theta 1 \vee A\theta 2 \vee \cdots \vee A\theta n.$$

We are now in a position to study the soundness and completeness of QN-PROLOG. We need to use a previous theorem, Theorem T2, which we proved in Section

4 paper I. The theorem dealt with the connection between the propositional and quantificational computations in N-PROLOG. Given a database **P**, which we shall represent schematically as $\mathbf{P}(u, x)$, with u the universal variable (VAR 1) and x the free choice variable (VAR 2), and given a goal $G(x, z)$ (x, z both VAR 2 variables), then success in the computation of $\mathbf{P}?G$ in quantificational QN-PROLOG meant that there exists a θ such that $(\mathbf{P}\theta)^*?G\theta$ succeeds in propositional N-PROLOG. $(\mathbf{P}\theta)^*$ was defined in paper 1, Section 4 as the *propositional freeze* of $\mathbf{P}\theta$, namely, as the conjunction of all the results of the substitutions of all possible Herbrand terms for the VAR 1 variable in $\mathbf{P}\theta$. $(\mathbf{P}\theta)^*$ is really equivalent to $\forall u(\mathbf{P}\theta)$, where $\forall u$ symbolises schematically the universal closure (over VAR 1 variables) of $\mathbf{P}\theta$.

We can now prove:

Theorem T3 (Soundness and completeness of QN-PROLOG relative to the intuitionistic predicate logic). $\mathbf{P}?G$ *succeeds in QN-PROLOG if and only if in intuitionistic logic we have*

$$\vdash (\exists \text{VAR } 2)[(\forall \text{VAR } 1)\mathbf{P} \to G],$$

where $(\exists\text{VAR } 2)$ *indicates the existential closure over all* VAR 2 *variables, and* $(\forall \text{VAR } 1)$ *indicates the universal closure over all* VAR 1 *variables.*

PROOF. The $\to$ direction follows immediately. Assume that $\mathbf{P}?G = 1$ in QN-PROLOG. By Theorem T2 of Section 4, paper I, there exists a θ such that $G\theta$ succeeds in propositional N-PROLOG from $(\mathbf{P}\theta)^*$, the propositional freeze of $\mathbf{P}\theta$. By the completeness of N-PROLOG for the intuitionistic propositional logic, we get that $(\mathbf{P}\theta)^* \vdash G\theta$. By the quantifier rules of our logic we get $(\forall\text{VAR } 1)\mathbf{P}\theta \vdash G\theta$ and hence $\vdash (\exists\text{VAR } 2)[(\forall\text{VAR } 1)\mathbf{P} \to G]$.

For the other direction assume that

$$\vdash (\exists\text{VAR } 2)[(\forall\text{VAR } 1)\mathbf{P} \to G].$$

Then by Theorem T1, for some θ, $(\forall\text{VAR } 1)\mathbf{P}\theta \vdash G\theta$. Hence for the propositional freeze $(\mathbf{P}\theta)^*$,

$$(\mathbf{P}\theta)^* \vdash G\theta;$$

hence by completeness,

$$(\mathbf{P}\theta)^* ?G\theta = 1$$

and hence $\mathbf{P}?G = 1$ in *QN*-PROLOG

This concludes the proof of theorem T3. □

We now define the restart rule for QN-PROLOG and thus obtain QNR-PROLOG, namely, QN-PROLOG with restart.

Definition D1.

(a) The Restart rule for QN-PROLOG states that if in the course of the computation for the original goal $G(xi)$, where xi are all the VAR 2 variables of G, we reach an atomic head Q as the current subgoal, then we may replace the atom Q by the new current goal $G(\bar{x}i)$, which is a copy of the original goal with *completely new* (to the computation so far) VAR 2 variables $\bar{x}i$.

(b) Let $G(xi)$ be a goal with xi the VAR 2 variable of G. Then the complement of G, denoted by Cop(G), is defined by

$\text{Cop}(G) = \{G(ui) \to Q \mid Q$ is any atomic wff of the language with VAR 2 terms in it, and $G(ui)$ is a fixed copy of $G(xi)$ with VAR 1 variables ui replacing $xi\}$.

Theorem T4. For any database **P** *and any goal G on QN-PROLOG, the following three conditions are equivalent*:

(a) *In classical logic*

$$\vdash (\exists \text{VAR } 2)[(\forall \text{VAR } 1)\mathbf{P} \to G].$$

(b) $\mathbf{P} \cup \text{cop}(G)?G = 1$ *in* QN-PROLOG.

(c) $\mathbf{P}?G = 1$ *in* QNR-PROLOG (*i.e. in* QN-PROLOG *with the restart rule*).

PROOF. The proof that (c) is equivalent to (b) is similar to the proof of the same theorem in the propositional case. We concentrate now on the proof that (a) is equivalent to (b).

Assume that $\mathbf{P} \cup \text{cop}(G)?G = 1$. Then by the soundness of QN-PROLOG we get that for some θ

$$(\forall \text{VAR } 1)[P\theta \cup \text{cop}(G)\theta] \vdash G\theta$$

in intuitionistic logic, and hence certainly in classical logic. We claim that in classical logic, $\neg(\exists \text{VAR } 1)G \vdash \text{cop}(G)\theta$, where $(\exists \text{VAR } 1)\ G$ is obtained from G by replacing all VAR 2 variables of G with corresponding VAR 1 variables and existentially quantifying them. The above is true in classical logic because all elements in cop(G) have the form

$$G \to Q,$$

and certainly

$$\neg(\exists \text{VAR } 1)G \vdash G \to Q,$$

because $\neg(\exists \text{VAR } 1)G \wedge G$ is a contradiction. Thus we get

$$(\forall \text{VAR } 1)\mathbf{P}\theta \wedge \neg(\exists \text{VAR } 1)G \vdash G\theta;$$

hence

$$(\forall \text{VAR } 1)\mathbf{P}\theta \vdash (\exists \text{VAR } 1)G \vee G\theta;$$

hence

$$(\exists \text{VAR } 2)[(\forall \text{VAR } 1)\mathbf{P} \to G \vee (\exists \text{VAR } 1)G],$$

which is classically equivalent to

$$\vdash (\exists \text{VAR } 2)[(\forall \text{VAR } 1)\mathbf{P} \to G].$$

This proves that (b) implies (a).

We assume now that

$$\mathbf{P} \cup \mathrm{cop}(G)?G \neq 1$$

and show that in classical logic

$$\nvdash \exists \text{VAR } 2[(\forall \text{VAR } 1)\mathbf{P} \to G].$$

To show this we proceed in a way similar to what we have done in the propositional case. We prove a lemma like Lemma L1 of Section 1 of this paper and follow the construction of a model as in the proof of Theorem T3 of Section 1. Although the proofs need to be checked in detail, we don't think it is worth while doing so in this paper. □

4. NEGATION AS FAILURE IN N-PROLOG

It is possible, natural, and very useful to add negation as failure to N-PROLOG. One must do that carefully, however, with a full understanding of the logical nature of the negation involved. The addition of negation as failure to N-PROLOG is not a simple matter from the logical point of view, because N-PROLOG is a much more expressive language than PROLOG. We begin with two simple examples to illustrate the nature of the difficulty. In ordinary PROLOG, the theorem of Clark, Lloyd, et al. logically characterizes negation as failure. Formulated for the propositional case, it states that for any set **P** of data without negation, there exists a uniform way of extending **P** to a bigger set, called Com(**P**) (the completion of **P**), such that for any goal B the following holds:

$$B \text{ finitely fails from } \mathbf{P} \text{ iff } \mathrm{Com}(\mathbf{P}) \vdash \neg B.$$

The above shows that negation as failure in ordinary PROLOG has a sound logical meaning.

In fact, Keith Clark goes as far as to say that when we specify **P**, we are really specifying Com(**P**).

Example E1. Consider the following data **P** of N-PROLOG:

$$\mathbf{P} = \{(a \to b) \to a\}$$

and the goal $?a$. It is easy to verify that $\mathbf{P}?a$ finitely fails. However, in classical logic, we have $\mathbf{P} \vdash a$ [i.e., $((a \to b) \to a) \to a$ is a classical tautology]. Thus no $\mathrm{Com}(P) \supseteq \mathbf{P}$ can prove $\neg a$. Thus the analog of the Clark theorem cannot hold for N-PROLOG.

This example is not as disappointing as it may first seem. If we recall that success in a computation in N-PROLOG (i.e. $\mathbf{P}?a = 1$) means intuitionistic provability, then failure (or finite failure) implies intuitionistic unprovability. Thus that $\mathbf{P}?a$ finitely fails will imply $\mathbf{P} \nvdash a$ in intuitionistic logic. But $\mathbf{P} \vdash \neg a$ is much stronger in intuitionistic logic than $\mathbf{P} \nvdash a$, and it is quite possible and not surprising at all that we can have both conditions below true in intuitionistic logic:

$$\{(a \to b) \to a\} \nvdash a,$$

$$\{(a \to b) \to a] \cup \{\neg a\} \text{ is inconsistent.}$$

What we possibly need is another kind of completion. We do not expect $\mathrm{Com}((a \to b) \to a) \vdash \neg a$ in intuitionistic logic, but maybe in another logic, or maybe something like $\mathrm{Com}^*(\mathbf{P}) \vdash$ "a is *not* true now" in the temporal logic interpretation of $\to$.

Another problem associated with negation as failure appears already in ordinary PROLOG. Consider the following well-known example:

Example E2. Consider the database

$p \rightarrow p$,

$p \rightarrow r$,

$\neg p \rightarrow r$,

$\neg r \rightarrow q$,

$\mathbf{P}?q$ loops.

However,

$\mathrm{Com}(\mathbf{P}) \vdash \neg q$.

Examples of this sort are known to Clark and Lloyd and show that their theorem cannot be extended to allow negations in **P** without further modification.

The first possible modification which comes to mind is to avoid loops. Thus the simple-minded first approximation would be something like **P** (with a good loop checker) $?B$ finitely fails iff $\mathrm{Com}^*(P) \vdash \neg B$. Here we can allow $\neg$ in clauses in **P**, and allow for a good loop checker, and even allow for a possible different completion of **P**, denoted by Com*(**P**). This is not enough, unfortunately. Consider the following example:

Example E3. Take the database

$a \rightarrow C$,

$\neg C \rightarrow a$.

Let the goal be $?C$. The above database is logically equivalent to C.

Written in disjunctive clauses, the data become

$\neg a \vee C$,

$C \vee a$.

If we want to keep the PROLOG Horn clause flavor of the computation and not rewrite everything as resolution clauses, we must leave the data in their present syntactic form and try to make $?C$ succeed and make $?a$ fail via some loop checking means. This however is not possible. Since $?C$ can succeed only through $a \rightarrow C$, the two goals $?a$ and $?C$ either succeed together or fail together. Therefore, unless we *rewrite* the data or have a special loop checker or *add* data, we cannot force $\neg$ to behave like classical negation.

The above shows that negation as failure is a tricky sort of negation, and it is our task in this section to add negation as failure to N-PROLOG and to understand its logical nature.

Let us now introduce negation as failure formally into N-PROLOG. We deal, as we have done in previous sections, with propositional N-PROLOG. The quantifiers will be dealt with in a special section at the end of the paper.

The language contains, besides conjunction $\wedge$ and implication $\rightarrow$, the negation symbol $\neg$. As in ordinary PROLOG, we want to allow negations only in the body of clauses and not in heads. Thus a datum

$$b \wedge \neg a \rightarrow b$$

is acceptable, but not

$$q \rightarrow \neg C.$$

However, if we have a clause like

$$(\neg C \rightarrow a) \rightarrow b,$$

then when asked $?b$, we shall have to add $\neg C$ to the database, which is not allowed. We thus need a proper inductive definition of a goal-with-negation and a clause-with-negation.

Definition D1 (Clause and goals in the language with negation).

(a) *Any atomic q is both a clause and a goal.*

(b) *If A is a goal then $\neg A$ is a goal.*

(c) *If A and B are goals then $A \wedge B$ is a goal.*

(d) *If A is a goal and q atomic, then $A \rightarrow q$ is a clause.*

(e) *If Ai are clauses and q is atomic, then $\bigwedge Ai \rightarrow q$ is a goal.*

Definition D2 (Computation tree for propositional N-PROLOG with negation as failure). *A tree $(T, \leq, 0, V)$ is a (success or finite failure) computation tree of the goal $G0$ from $\mathbf{P}0$ iff the following conditions are satisfied:*

(a) $(T, \leq, 0)$ *is a tree with root* 0.

(b) *V is a labeling function. For each $t \in T$, $V(t)$ is $V(t) = (\mathbf{P}(t), G(t), x(t))$, where $\mathbf{P}(t)$ is a database, $G(t)$ a goal, and x is a number,* 0 *to* 1.

(c) $V(0) = (\mathbf{P}0, G0, x(0))$, *where* $x(0) = 1$ *for success and* $x(0) = 0$ *for finite failure.*

(d) *Let t be any node in the tree such that*

$$V(t) = (\mathbf{P}(t), G(t), x(t))$$

and assume that

$$G(t) = G1(t) \wedge G2(t).$$

Then the following holds:

(d1) $x(t) = 1.$

In this case t has exactly two immediately succeeding points in the tree, $s1$ *and* $s2$, *as shown:*

$$\begin{tikzcd}
s1 & & s2 \\
& t \arrow[ul, dash] \arrow[ur, dash] &
\end{tikzcd}$$

such that

$V(si) = (\mathbf{P}(t), Gi(t), 1)$ *for* $i = 1, 2,$ *and*

(d2) $x(t) = 0.$
In this case t has exactly one immediate succeeding point in the tree, as shown:

$$\begin{tikzcd} s \arrow[d, dash, dotted] \\ t \end{tikzcd}$$

and

$V(s) = (\mathbf{P}(t), Gi(t), 0)$

where i is either 1 *or* 2, *i.e.* $i \in \{1,2\}$.

(e) *Let t be a node such that*

$V(t) = (\mathbf{P}(t), \neg G(t), x(t))$

Then t has exactly one immediately succeeding point s in the tree as shown:

$$\begin{tikzcd} s \arrow[d, dash, dotted] \\ t \end{tikzcd}$$

and $V(s) = (\mathbf{P}(t), G(t), 1 - x(t))$.

(f) *Let t be a node such that* $V(t) = (\mathbf{P}(t), G(t) \to Q(t), x(t))$. *Then t has exactly one immediately succeeding point s in the tree as shown*:

$$\begin{tikzcd} s \arrow[d, dash, dotted] \\ t \end{tikzcd}$$

and $V(s) = (\mathbf{P}(t) + G(t), Q(t), x(t))$.

(g) *Assume that t is a node with*

$V(t) = (\mathbf{P}(t), q, x(t))$, *for q atomic.*

Then the following holds:

(g1) $x(t) = 1$ *and t is an endpoint of the tree. In this case,*
$q \in \mathbf{P}(t)$.

(g2) $x(t) = 0$ *and t is an endpoint of the tree. In this case,*
q is not head of any clause of $\mathbf{P}(t)$.

(g3) $x(t) = 0$ *and t is not an endpoint of the tree. Then for some* $m \geq 1$, *there exists exactly m immediately succeeding points si to t in the tree as shown*:

$$\begin{tikzcd} s1 & \cdots & sm \\ & t \arrow[ul, dash] \arrow[ur, dash] & \end{tikzcd}$$

and there exist exactly m clauses in $\mathbf{P}(t)$ *with heads q of the form*

$Bi \to q,\ i = 1, \ldots, m,$

such that for $i = 1, \ldots, m$,

$V(si) = (\mathbf{P}(t), Bi, 0)$.

(g4) *If* $x(t) = 1$ *and* t *is not an endpoint of the tree, then* t *has exactly one immediately succeeding point* s *in the tree*

$$\begin{tikzcd}
s \\
t \arrow[u, dash, dotted]
\end{tikzcd}$$

and $V(s) = (\mathbf{P}(t), B, 1)$.

Example E4. Let

(1) $(C \to a) \to C$,

(2) $(\neg b \wedge (b \to C)) \to a$,

and consider the goal G with

$(C \to a) \to a$.

This database and goal present us with a serious puzzle. The reason is that the following are all true:

(*1) $\mathbf{P}?(C \to a) = 1$,

(*2) $\mathbf{P}?(C \to a) \to a = 1$,

but

(*3) $\mathbf{P}?a = 0$.

Tree for (*1):

$$\begin{tikzcd}
(\mathbf{P} + C, b, 0) & & (\mathbf{P} + C + b, C, 1) \\
(\mathbf{P} + C, \neg b, 1) \arrow[u] & & (\mathbf{P} + C, b \to C, 1) \arrow[u] \\
& (\mathbf{P} + C, a, 1) \arrow[ul] \arrow[ur] & \\
& (\mathbf{P}, C \to a, 1) \arrow[u] &
\end{tikzcd}$$

Tree for (*2):

$$\begin{tikzcd}
(\mathbf{P} + (C \to a) + C, C, 1) \\
(\mathbf{P} + C \to a + C, a, 1) \arrow[u] \\
(\mathbf{P} + (C \to a), C \to a, 1) \arrow[u] \\
(\mathbf{P} + (C \to a), C, 1) \arrow[u] \\
(\mathbf{P} + (C \to a), a, 1) \arrow[u] \\
(\mathbf{P}, (C \to a) \to a, 1) \arrow[u]
\end{tikzcd}$$

Tree for (*3):

$$\begin{array}{ll}
 & (\mathbf{P}+b, C, b, 1) \\
\uparrow & \\
 & (\mathbf{P}+b+C, \neg b, 0) \\
\uparrow & \\
 & (\mathbf{P}+b+C, \neg b \wedge (C \rightarrow a), 0) \\
\uparrow & \\
 & (\mathbf{P}+b+C, a, 0) \\
\uparrow & \\
 & (\mathbf{P}+b, C \rightarrow a, 0) \\
\uparrow & \\
 & (\mathbf{P}+b, C, 0) \\
\uparrow & \\
 & (\mathbf{P}, b \rightarrow C, 0) \\
\uparrow & \\
 & (\mathbf{P}, \neg b \wedge (b \rightarrow C), 0) \\
\uparrow & \\
 & (\mathbf{P}, a, 0)
\end{array}$$

Example E5. Here is a very simple example:

(1) $(\neg b \rightarrow a)?a = 1,$

(2) $((\neg b \rightarrow a) + a)?b \rightarrow a = 1,$

(3) $(\neg b \rightarrow a)?b \rightarrow a = 0.$

Examples E4 and E5 are a problem, because they mean that negation as failure is *not* logical. They show that we can have a situation where

$$\mathbf{P}?A + 1,$$

$$\mathbf{P} + A?B = 1,$$

and

$$\mathbf{P}?B = 0.$$

Thus we cannot generate lemmas (namely A) and use them in further computations (namely B).

We now have to worry about two points. First, does the same happen in ordinary PROLOG with negation as failure? We know Clark's theorem is available for ordinary PROLOG, but we don't allow negations in the body. Lemma L1 below shows that this problem does not arise in ordinary PROLOG with negation allowed in clauses.

Second, do we have the correct and coherent notion of negation as failure for N-PROLOG?

We examine the second point later. Let us prove:

Lemma L1. Let **P** *be an ordinary PROLOG database, G be an ordinary PROLOG goal, and a atomic. Then* (a) *and* (b) *imply* (c) *below*:

(a) $\mathbf{P}?a = 1,$

(b) $\mathbf{P} + a?G = x,$

(c) $\mathbf{P}?G = x.$

PROOF. By induction on G and on the length of the computation of G.

(a) Let $G = G1 \wedge G2$.

If $x = 1$ then $\mathbf{P} + a?Gi = 1$, $i = 1,2$, and by the induction hypothesis $\mathbf{P}?Gi = 1$ and hence $\mathbf{P}?G = 1$.

If $x = 0$, then either $\mathbf{P} + a?G1 = 0$ or $\mathbf{P} + a?G2 = 0$. By the induction hypothesis $\mathbf{P}?G1 = 0$ or $\mathbf{P}?G2 = 0$, and hence $\mathbf{P}?G = 0$.

(b) If $G = \neg G'$, then the case reduces to the case of $G1$ and $x' = 1 - x$.

(c) If $G = b$, b atomic.

(c1) If $x = 1$ then $\mathbf{P} + a?b = 1$. If $b = a$ then $\mathbf{P}?b = 1$, which is what we want to show. If $b \neq a$, then for some clause $C \rightarrow b \in \mathbf{P}$, $\mathbf{P} + a?C = 1$. By the induction hypothesis, since C has a shorter computation tree, $\mathbf{P}?C = 1$ and hence $\mathbf{P}?b = 1$.

(c2) If $x = 0$, then clearly $a \neq b$, and either b is not the head of any clause in $\mathbf{P}$, in which case $\mathbf{P}?b = 0$, or for all clauses of the form $C \rightarrow b \in \mathbf{P}$, $\mathbf{P} + a?C = 0$. By the induction hypothesis $\mathbf{P}?C = 0$ and hence $\mathbf{P}?b = 0$.

This proves Lemma L1. □

We see that the source of the problem is the implication $\rightarrow$, which makes the database increase. It may have occurred to the reader that if we start a computation

$$\mathbf{P}?G$$

and in the course of the computation the database increases, we should still read any negation $\neg A$ as failure relative to the *original* database. Thus for example when we have

$$\neg b \rightarrow a?b \rightarrow a = 1,$$

we ask

$$(\neg b \rightarrow a) + b?a = 1$$

and then ask

$$(\neg b \rightarrow a) + b?\neg b = 1,$$

but now $\neg b$ is failure from the original database (namely $\neg b \rightarrow a$) and *not* the current database [namely $(\neg b \rightarrow a) + b$].

This approach seems to give $\neg A$ a meaning independent of the changes in the database and may even be free of our previous difficulties. However the notion is not coherent. We can ask

$$\mathbf{P}?A = 1;$$

we can also ask

$$\varnothing ?(\bigwedge\mathbf{P} \rightarrow A) = 1.$$

These two queries must be the same. However, all $\neg$ in $\mathbf{P}$ are evaluated relative to $\mathbf{P}$ in the first instance and relative to $\varnothing$ in the second. We lose coherence in our notion of negation.

We are thus forced to discover the "logic" of our present notion of negation. It is a very useful and practical notion and allows us to have control of our programs. If we can pin it down, we shall have a powerful tool at our disposal.

Another option open to us is to syntactically restrict the use of negation in such a way that it remains both useful and logical. ·

Let us look at some more examples and see how negation as failure works in N-PROLOG.

Example E6. We saw that we can name clauses in N-PROLOG by suffixing a special atom in front of the clause. Thus to name clause A we can write

$$(A \to \mathbf{name}) \to \mathbf{name}.$$

Consider the following database **P**:

(a) $(A \wedge q \to \mathbf{name}\ 1) \to \mathbf{name}\ 2,$

(b) $(B \to \mathbf{name}\ 2) \wedge \neg q \to \mathbf{name}\ 1.$

We can ask for two goals here:

$$G1 = ((A \wedge B) \to \mathbf{name}\ 1) \to \mathbf{name}\ 1,$$

$$G2 = ((A \wedge B) \to \mathbf{name}\ 2) \to \mathbf{name}\ 2.$$

$G1$ succeeds and $G2$ finitely fails. $G1$ says start the computation from clause (b), and $G2$ says start the computation from clause (a). The computation of $G1$ or of $G2$ does not depend on the operation of the performing machine. It is completely determined by the data and the goal.

Let us see now if the goal $G1$, which succeeds, would still succeed when $\neg$ is interpreted as classical negation. In other words, we check whether $P \vdash G1$. The answer is no, i.e., $\mathbf{P} \nvdash G1$.

Let

$$\mathbf{name}\ 1 = \text{false},$$

$$\mathbf{name}\ 2 = \text{true},$$

$$A = \text{false},$$

$$B = \text{false},$$

$$q = \text{true}.$$

The only way to understand the role of negation as failure in this program is to look at the temporal sequence of execution. If we ask $?\mathbf{name}\ 1$ first, we are starting with clause (b). If we ask $?\mathbf{name}\ 2$ first, then we are starting from clause (a).

We try to save the concept of negation by in effect delaying the computation of any negation $\neg q$ until the *largest* database is reached. This notion, which we have not made precise yet, may be coherent.

It makes no difference whether we ask $\mathbf{P}?G$ or $\varnothing\ ?\mathbf{P} \to G$. The largest database reached is always the same.

Our notion is therefore the following (intuitively defined). Negation as persistent failure is a binary notion, for the form $T1(G, q)$. Its meaning is that G succeeds and at no step of the computation does q succeed. Unfortunately it is not coherent, because $a?T1(a,q)$ succeeds, but $(q \to q) \to a?T1\ (a,q)$ fails, and $(q \to q) \to a$ is the same as a. This notion is similar in technical form to a loop checker. We compute the goal, and at no step of the computation do we repeat ourselves.

Let us examine again the reasons for our difficulty. Negation as failure is known to be nonmonotonic, even in the case of ordinary PROLOG. If $\mathbf{P}?a$ succeeds and the database is increased to $\mathbf{P}' \supseteq \mathbf{P}$, we do not expect $\mathbf{P}'?a$ also to succeed. It may succeed, or it may not. For example

$$a?\neg b$$

succeeds, but

$$\{a,b\}?\neg b$$

finitely fails.

In N-PROLOG the database changes all the time, and this is the source of our difficulty. The equation $\mathbf{P}?A = 1$ means that A succeeds from the database $\mathbf{P}$. The equation $\mathbf{P}+A?B=1$ means that B succeeds from the database $\mathbf{P}+A$. What knowledge do the two equations above give us about the success or failure of $(\mathbf{P}?B)$? Why should we expect B to succeed from $\mathbf{P}$?

We do want a comfortable notion of negation in N-PROLOG: a notion for which some reasonable form of the "modus ponens" lemma holds. To achieve that we can proceed in two alternative ways:

(a) Leave negation as failure as it is, in Definition D2, but make sure that we always have means of saying which goal succeeds from exactly which database, and thus find a way to deal with Examples E4 and E5.

(b) Adopt a stronger notion of negation, one which can survive changes in the database (as an extreme example, we can always use classical negation), and thus have proper modus ponens, and lemma generation.

We shall examine each of these alternatives. This paper will adopt the first alternative, namely, use the notion of negation as failure we already have. A variant of the second alternative is developed for ordinary PROLOG itself in our paper on Negation as Inconsistency [3].

Let us try and make sense of the first alternative, namely, of leaving negation as failure as it is in Definition D2. The problem is that when we see a $\neg q$ in the middle of a clause in a program $\mathbf{P}$, the meaning of $\neg q$ depends on the order of the execution of the program. As our database increases during the computation, $\neg q$ can mean:

(1) q fails from $\mathbf{P1}$, the current database,

and then if we encounter $\neg q$ again, possibly after some backtracking, $\neg q$ could mean:

(2) q fails from $\mathbf{P2}$ a different database, which happens to be current at the moment.

This problem does not arise in ordinary PROLOG, because the original database is fixed and does not change, and hence $\neg q$ has one fixed clear-cut meaning.

We can now formulate exactly the notion of lemma generation. Suppose $\mathbf{P}?A = 1$, i.e., A succeeds from $\mathbf{P}$. We want to add to $\mathbf{P}$ the lemma that A succeeds from $\mathbf{P}$, so that when we have a new computation, say of $\mathbf{P}?B$, and in the course of the new computation we encounter the subgoal $\mathbf{P}'?A$, we can use the lemma, and we do not have to recompute goal A.

The way we tried to add the lemma was to add A to $\mathbf{P}$. This is wrong. We know that $\mathbf{P}?A = 1$, but we *do not* know that e.g. $\mathbf{P} + q?A = 1$, because $\mathbf{P} + q$ is a larger database.

Suppose we ask $\mathbf{P}?B$. In the course of the computation of B from $\mathbf{P}$ we may encounter the subgoal $\mathbf{P}'?A$, where $\mathbf{P}' \supseteq P$ is a possibly larger database. If $\mathbf{P}' = \mathbf{P}$, then we know that A succeeds, and we can use the lemma and there is no need to continue the computation of A. But if $\mathbf{P}' = \mathbf{P} + q$, there is no guarantee that A succeeds, and we cannot use the lemma.

Yet, if we add A to $\mathbf{P}$, A will succeed from any extension $\mathbf{P}'$ containing $\mathbf{P} + A$. Thus the addition of A to $\mathbf{P}$ is equivalent to saying $\forall \mathbf{P}' \supseteq \mathbf{P}(\mathbf{P}'?A = 1)$, and is much stronger than the mere lemma $\mathbf{P}?A = 1$, which we want. Thus the possible lemmas should be either lemma L2 or Lemma L3 below:

Lemma L2.

$$\forall \mathbf{P}' \supseteq \mathbf{P}(\mathbf{P}'?A = 1) \textit{ and } \mathbf{P} + A?B = 1$$

$$\textit{implies } \mathbf{P}?B = 1.$$

Lemma L3.

$$\mathbf{P}?A = 1 \textit{ and } (\mathbf{P} + [\textit{fact that } \mathbf{P}?A = 1])?B = 1$$

$$\textit{implies } \mathbf{P}?B = 1.$$

How do we write an N-PROLOG clause which expresses the fact that $(\mathbf{P}?A = 1)$? We really want to say, whenever the current goal is A and the current database is $\mathbf{P}$, then A succeeds. To be able to express this fact we need a new predicate, which we call **NData** (X) (new data X).

Definition D3. Consider a computation for an original database $\mathbf{P}$ and goal G, i.e. for $\mathbf{P}?G$. Let **NData** (X) be the predicate with the intended meaning that the current addition to the original database is X (i.e. the current database is $\mathbf{P} + X$). Thus

$$\mathbf{P}?\mathbf{NDATA}\ (X) = 1$$

iff

$$X = \mathbf{P}' - \mathbf{P}.$$

If X is just a variable, then X will be instantiated as $\mathbf{P}' - \mathbf{P}$. If X is just a database, then the VAR 2 variables (the free choice variables) in $\mathbf{P}'$ and in X will be instantiated (via a substitution θ) in such a way as to allow, if possible, for $(\mathbf{P}' - \mathbf{P})\theta = X\theta$.

Let G be a goal. Assume that $G = \bigwedge_i (Ai \to ai)$, ai atomic.

$\mathbf{P}?G = 1$ is clearly equivalent to the conjunction on i of $\mathbf{P} + Ai?ai = 1$. Thus the information that G succeeds from $\mathbf{P}$ can be expressed by the following clauses, using **NData**:

$$\mathbf{NData}\ (Ai) \to ai.$$

Thus Lemma L3 becomes:

Lemma L3. (a) *and* (b) *below imply* (c):

(a) $\mathbf{P}?\bigwedge_i (Ai \to ai) = 1.$

(b) $(\mathbf{P} + \{\mathbf{NData}\ (Ai) \to ai\})?B = 1.$

(c) $\mathbf{P}?B = 1.$

Before we consider the proof, let us see what happens with Examples E4 and E5, which gave us all the trouble before. Let us see whether they violate the above Lemma L3.

Example E4. **P** is

(1) $(C \to a) \to C,$

(2) $(\neg b \wedge (b \to C)) \to a.$

We know that

$$\mathbf{P}?(C \to a) = 1.$$

We therefore want to add to **P** the clause

$$\mathbf{NData}\ (C) \to a.$$

Let us see whether $?a$ succeeds from this new database. If $?a$ indeed succeeds, then we still have a counterexample to our Lemma L3, since $\mathbf{P}?a$ is known to fail. Let us check then: The new database is **P1** given by

(1) $(C \to a) \to C,$

(2) $(\neg b \wedge (b \to C)) \to a,$

(3) $\mathbf{N\text{-}Data}\ (C) \to a.$

The goal is $?a$. The question is: Does $?a$ succeed from this database **P1**? Checking the computation, we find that $?a$ finitely fails from **P1**.

Below we have the failure tree of this computation. Notice that in this tree $(\mathbf{P1}, \mathbf{NData}\ (C), 0)$ succeeds (i.e. is an endpoint), because nothing was added to **P1** and hence $\mathbf{P1}?\mathbf{NData}\ (C) = 0$. On the other hand $(\mathbf{P1} + b + C, \mathbf{NData}\ (C), 0)$ succeeds because too much (more than exactly C) was added to **P1** and hence

$\mathbf{P1} + b + C\,?\,\mathbf{NData}\,(C) = 0$:

$$\begin{tikzcd}[row sep=small]
(\mathbf{P1} + b + C, \mathbf{NData}\,(C), 0): \arrow[dr, dash] & \\
& \left\{ \begin{array}{l} (\mathbf{P1} + b + C, b, 1) \\ (\mathbf{P1} + b + C, \neg b, 0) \\ (\mathbf{P1} + b + C, \neg b \wedge (b \to C), 0) \\ (\mathbf{P1} + b + C, a, 0) \\ (\mathbf{P1} + b, C \to a, 0) \\ (\mathbf{P1} + b, C, 0) \\ (\mathbf{P1}, b \to C, 0) \\ (\mathbf{P1}, \neg b \wedge (b \to C), 0) \end{array} \right. \\
(\mathbf{P1}, \mathbf{NData}\,(C), 0): \arrow[r, dash] & \{ (\mathbf{P1}, a, 0)
\end{tikzcd}$$

*Example E5**. We now check Example E5. This was

$$\neg b \to a\,?\,a = 1.$$

Hence we add to the database $\mathbf{NData}\,(\varnothing) \to a$ to form **P1**. We now check whether $\mathbf{P1}\,?\,b \to a = 1$.

The answer is no. Here is a failure tree:

$$\begin{tikzcd}[row sep=small]
& & (\mathbf{P1} + b, b, 1) \arrow[d, dash] \\
(\mathbf{P1} + b, \mathbf{NData}\,(\varnothing), 0) \arrow[dr, dash] & & (\mathbf{P1} + b, \neg b, 0) \arrow[dl, dash] \\
& (\mathbf{P1} + b, a, 0) \arrow[d, dash] & \\
& (\mathbf{P1}, b \to a, 0) &
\end{tikzcd}$$

We thus see that neither counterexample works. We can now try to prove Lemma L3.

Suppose we have a computation of ai from $\mathbf{P} + \mathbf{NData}\,(Ai) \to ai$. If we never use the additional clause, then clearly $\mathbf{P}\,?\,B = 1$.

Suppose we do use the additional clause. Consider any stage of the computation in which the clause was successfully used. The goal was $?\,ai$ at that stage. Hence we have at that moment the database and goal

$$\mathbf{P}'\,?\,ai = 1.$$

Hence we use the clause $\mathbf{NData}\,(Ai) \to ai$ and ask:

$$\mathbf{P}'\,?\,\mathbf{NData}\,(Ai) = 1.$$

This goal succeeds exactly if at the moment of use, the current database was $\mathbf{P}' = \mathbf{P} + Ai$. But then we know that $\mathbf{P} + Ai\,?\,ai = 1$, and hence we can replace the successful computation of $\mathbf{P}'\,?\,ai$ via the rule $\mathbf{Data}(Ai) \to ai$ by the actual successful computation of $\mathbf{P} + Ai\,?\,ai$. Thus $\mathbf{P}\,?\,b$ can succeed directly from **P**, without any use of the additional clauses $\mathbf{NData}\,(Ai) \to ai$, because their use can always be replaced.

We have thus managed to make negation as failure acceptable, at the cost of using the predicate $\mathbf{NData}\,(X)$. We must show that $\mathbf{NData}\,(X)$ is a logical predicate, i.e., we need a proper semantics for it. This is possible to do.

We shall have to talk more about the predicates **suspend**, and **restore**, mentioned in paper I, and also about **NData**. These are really *modal* operators, and they will be studied at length in paper III, when we use N-PROLOG for database updating, for expressing temporal phenomena in databases, and generally as a language for handling databases.

Before we conclude, let us say a few words about the second approach to negation, i.e. about *N-PROLOG with strong negation*. This approach is to make the notion of negation more absolute, more strong, and less dependent on the database. The obvious way to proceed is to list in the database itself which atoms are to be negated.

The closed world assumption assumes automatically that any atom q which is not the head of any clause is negated. We can relinquish this principle and say that we shall list explicitly the atomic q's to be negated. If the database increases, the list remains. Negation of atoms becomes a more positive notion. Thus

> $a?\neg b$ will fail because $\neg b$ is *not* listed in the database, $\{a, b\}?\neg b$ also fails,
>
> but
>
> $\{a, \neg b\}?\neg b$ succeeds.

We now ask what happens if we have the following query:

(*) $\{\neg b\}?b \to a.$

Following the rule for $\to$, we have to ask

(**) $\{\neg b, b\}?a.$

We have two alternatives:

(1) Decide that $\{\neg b, b\}$ can imply anything, as done in classical logic, and hence the original query succeeds.

(2) Decide that we do not allow inconsistent databases, and hence step (**) is not allowed, and hence the query fails.

The first alternative leads to some form of classical negation in N-PROLOG. This approach is studied for the case of ordinary PROLOG in our paper on Negation as Inconsistency [3]. The second alternative is also interesting and merits examination.

The next section discusses the use of metalanguage features within QN-PROLOG itself. Negation as failure will also be considered a part of QN-PROLOG, and the role of negation will become clearer within the metalanguage framework.

5. METALANGUAGE FEATURES OF QN-PROLOG

We saw in paper I of this series that the implication $\to$ of QN-PROLOG has a metalanguage meaning. If Demo(A, B) reads "B succeeds as a goal from the data A" then Demo(A, B) can be represented in QN-PROLOG by

Suspend $\wedge (A \to B) \wedge$ **Restore**.

We further saw how the properties of $\rightarrow$ can be utilized for metalanguage naming of clauses and control. We also used negation by failure for control purposes and expressed operations such as conditional deletion of clauses.

The purpose of this section is to use the metalanguage features of $\rightarrow$ in an organized and systematic way, similar in scope to the way that Gödel numbers and provability predicates are used in classical arithmetic. This will enable us to use negation as failure in a controlled logical way and thus will also help us solve some of the puzzles of the previous section.

In paper I, the clauses and goals of QN-PROLOG were defined. Data clauses can contain two types of variables: VAR 1 variables, read universally, and VAR 2 variables, playing the role of free-choice variables. The goals of QN-PROLOG contain only VAR 2 variables. The atomic predicates and the function symbols were arbitrary, nothing special, as in ordinary predicate logic.

For the purpose of encoding metalanguage features in QN-PROLOG, we must add to the language of QN-PROLOG special variables and predicates. These we describe now.

First we need to name clauses systematically. In previous examples, such as the ones in paper I, we use constants **name** 1, **name** 2, etc. We use now special predicate symbol $T(k, t)$, where k is a name variable and t is a control parameter, whose role will become clear later.

The most convenient field for names is the integers, and so for example k can range over the integers and integers can be linked to clauses via some system of numbering. We shall define one later.

We also need a special unary predicate called $\mathbf{Cancel}(k, t)$, whose field are names k and t. Its meaning is deletion. We also need a three place relation $g(x, y, z)$ yielding names z for each pair of names x and y. We also allow g to take (y, z) alone as arguments, if x is not important. The parameter t ranges over integers. It is a control counter, counting the sequential steps of the QN-PROLOG procedural computation. It is well known that sequences of natural numbers can'be coded as natural numbers. Let $S(t)$ be another special predicate saying that t is a sequence of numbers, and let $\mathbf{Length}(t)$ be the function giving the length of the sequence t. $T1(t, i)$ will give the ith number of the sequence t, for $i \leq \mathbf{Length}(t)$. We also write (t, i) for $T1(t, i)$.

The above coding is done using numbers, but any coding domain can be used, provided the appropriate functions for sequencing are available.

Before we go on, let us consider an example. Recall Example E6 of the previous section. Let us rewrite this example in our notation.

Example E1. Consider the following clauses:

(a) $(A \wedge \mathbf{Cancel}(2, \mathrm{t}) \rightarrow T(g(1,2), t+1) \wedge \neg\mathbf{Cancel}(1, t) \rightarrow T(1, t)).$

(b) $(B \rightarrow T(g(2,1), t+1)) \wedge \neg\mathbf{Cancel}(2, t) \rightarrow T(2, t).$

Here g has only two arguments.

The name of the first clause (a) is 1. The name of the second clause is 2. $T(1, t)$ is the head allowing us to use clause 1 at any time t. The computation checks whether clause 1 is canceled, and if not, it will add A to the database, cancel clause 2, and continue with the goal $T(2, t+1)$ [since $g(1,2)$ holds], which says: go to clause 2 and the time now is $t+1$.

We need one more clause, which specifies the goal. The goal is $A \wedge B$ in our example, and so we put

(c) $A \wedge B \rightarrow T(k, 0)$.

We ask for $T(1,0)$ if we want to compute from clause 1 at time 0, and we ask for goal $T(2,0)$ if we want to compute from clause 2 at time 0.

We have to add one more clause to all programs, namely,

$$\textbf{Cancel}(k, t) \rightarrow \textbf{Cancel}(k, t+1).$$

We now check whether the goal $T(2,0)$ succeeds. This means that we want at time 0 to compute the goal $A \wedge B$ from clause (b), whose name is 2.

Step 0: Unify with clause (b), add B, and ask for $T(1,1)$.

Step 1: Unify with clause (a), add A, and cancel clause (b); ask for $T(2,2)$.

Step 3: Unify with clause (c), and ask for $A \wedge B$ and succeed.

Let k code the sequence $(1,2,1,2)$. Thus what is provable is

$$\exists k[S(k) \wedge (\forall i \leq \textbf{Length}(k))\ g((k,i),(k,i+1)) \wedge$$
$$\wedge \text{ (universal closure of clauses)} \rightarrow T(1,0)],$$

which also gives meaning to any instance of negation as failure which may appear in the clauses of the program.

Definition D1. Add to the language of QN-PROLOG the special predicates **Length**(x), $T(x,y)$, $S(x)$, $T1(x,y)$ [also written (x,y)], **Cancel**(x,y), and the functions $g(x,y,z)$ and $g(x,z)$ (g is both three place and binary). Let x, y range over numbers.

(1) The notion of QN-clause and goals are defined as in paper 1 of this series.

(2) We define the notions of QN-ready-for-metalanguage clauses and goals, which we refer to as m-clauses and m-goals. This is a subset of the set of all clauses and goals.

(a) Any atomic predicate is a m-clause and m-goal. An m-goal is required to contain VAR 2 variables only. These clauses have arbitrary names.

(b) Let Ai be m-clauses with names ki. Here ki is a code name for Ai; it is a natural number. Ai contains certain variables from VAR 1 and VAR 2. Let k be a totally new name which has not been used. Let Ci be finite sets of m-clause names which have already been used. Then the following is an m-clause with name k:

$\bigwedge_i[Ai \wedge \bigwedge_{j \in Ci} \textbf{Cancel}(j) \rightarrow T(g(ki, k, z), t+1)]$
$\wedge \neg\textbf{Cancel}(k) \rightarrow T(k, t)$,

where t is totally new VAR 1 variable. The above clause says: At time t, you can use me, the clause named k, if I haven't been canceled. If you use me, then in parallel, add clauses Ai to the database and go to any clause named

z such that $g(ki, k, z)$ is in the database, but in this computation cancel every clause named in Ci.

(c) The set of clauses is obtained by performing the definition (b) a double infinity of times, $\omega \cdot \omega$, where the ground case is (a) above and where when we apply (b) we use all possible Ai, Ci available at that stage. We need to iterate $\omega \cdot \omega$ times to compensate for the fact that when we use a new name k we can refer only to clauses already defined. In the next ω-round we can refer to clauses which will be defined in the present ω-round.

(d) An m-goal is any conjunction of m-clauses containing VAR 2 variable only.

(e) An m-database (or m-program) is any set of m-clauses containing the following additional, possibly ordinary clauses:

Axiom 1. $\mathbf{Cancel}(x, t) \to \mathbf{Cancel}(x, t+1)$, x, t from VAR 1.

Axiom 2. Coding axioms relating to S, **Length**, $T1$, etc.

Axiom 3. A possibly empty selection of control information such as

$$T(k1, t+1) \to T(k, t).$$

Axiom 4. An additional set of data of the form $g(a, b, c)$ or of the form $\bigwedge_i g(ai, bi, ci) \to g(a, b, c)$.

Paper III of this series will deal mainly with the handling of time phenomena, i.e. temporal PROLOG.

REFERENCES[1]

1. Gabbay, D. M., *Elementary Logic, a Procedural Perspective*, Lecture notes, Dept. of Computing, Imperial College, London, Dec. 1984.
2. Broda, K., Gabbay, D. M., and Kriwaczek, F., A Goal Directed Theorem Prover for Predicate Logic Based on Conjunctions and Implications, Report, Dept. of Computing, Imperial College, London, May 1985.
3. Gabbay, D. M. and Sergot, M. J., Negation as Inconsistency I, Research Report DOC 84/7, Dept. of Computing, Imperial College, London, Dec. 1984.
4. Gabbay, D. M. and Reyle, U., N-PROLOG: An Extension of PROLOG with Hypothetical Implications. I. *J. Logic Programming* 1:319–355 (1985).

[1] In addition to the references of paper 1.