Expressing Database Queries with Intuitionistic Logic

Anthony J. Bonner
L. Thorne McCarty
Kumar Vadaparty

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

Abstract

This paper develops a declarative language with intuitionistic semantics which expresses exactly the generic database queries. Syntactically, the language is an extension of Datalog (function-free Horn logic) which allows rules themselves to appear in the bodies of other rules. Such rules are called *embedded implications*. Several researchers have studied restricted versions of these rules, but in their full incarnation, universal quantifiers may appear in the premises, as in the rule $A \leftarrow \forall_x[B(x) \leftarrow C(x)]$. This paper focuses on these embedded universal quantifiers. It is shown, for instance, that such quantifiers give the logic the ability to create new constant symbols hypothetically during inference. This, in turn, allows the logic to simulate unbounded counters and arbitrary Turing machines. In addition, when the logic is augmented with negation-as-failure, it becomes expressively complete, that is, it can express any database query which is typed and generic. Similar results exist for other languages, but the novelty of this work is that expressive completeness is achieved with a declarative language based on a well-established semantics, i.e., intuitionistic semantics. Indeed, the paper presents a simple proof of the intuitionistic completeness of the logic.

Appears in *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 831–850. MIT press. Conference held in Cleveland, Ohio, October 16-20 1989.

This paper is available at the following URL:
`ftp://db.toronto.edu/pub/bonner/papers/hypotheticals/naclp89.ps.gz`

1 Introduction

This paper investigates a subset of intuitionistic logic which expresses exactly the generic database queries. Syntactically, it is an extension of Horn logic which allows rules themselves to appear in the bodies of other rules. Thus rules of the form $A \leftarrow (B \leftarrow C)$ are allowed. Such rules are called *embedded implications.*

Several researchers in the logic-programming and database communities have pointed out the utility of embedded implications, and have developed inference systems for them. Miller, for instance, has shown that such rules can structure the runtime environment of a logic program [18]; and Warren and Manchanda have proposed such logics for reasoning about database updates [20, 14]. The legal domain has inspired much work into embedded implications. Gabbay, for instance, has reported a need to augment Prolog with such rules in order to encode the British Nationality Act [11], and McCarty has used such rules as the foundation of a general "Language for Legal Discourse" [17].

Most research so far has focussed on a subset of embedded implications in which universal quantifiers do not appear in the bodies of rules. Thus rules of the form $A(x) \leftarrow [B(x) \leftarrow C(x)]$ have been considered, while rules of the form $A \leftarrow \forall_x[B(x) \leftarrow C(x)]$ have not, one exception being the work of McCarty [15, 16]. An embedded implication in which universal quantifiers do *not* appear in the rule body shall be said to have *restricted universal quantification.* This paper examines the unrestricted case. In addition, to maintain a database emphasis, attention is focussed on the function-free case. Thus, we are considering the most general extension of Datalog in which rules themselves may appear in the bodies of other rules.

Interesting theoretical results have been established for embedded implications in the restricted case: Gabbay has shown that such rules have an intuitionistic semantics [10], and Miller has developed a fixpoint semantics for the predicate case [18]. Bonner has shown that query evaluation in such systems is data-complete for PSPACE in the function-free case [4]. Furthermore, when this logic is augmented with negation-as-failure, any database query whose graph is in PSPACE can be expressed [4]. Syntactic restrictions have also been identified which express the database queries at each level in the polynomial time hierarchy [6]. Furthermore, there is a precise sense in which some low-complexity queries can be expressed with intuitionistic embedded implications but not with classical logic, even full first-order classical logic [3]. Theoretical research into the restricted case thus appears well developed. In contrast, research into the unrestricted case is still in progress. McCarty has developed a fixpoint semantics for embedded implications based on intuitionistic logic [15] as well as a practical proof procedure [16]. Nadathur and Miller have also investigated the unrestricted case, using it as the basis for an extension of Prolog called λProlog [19]. But the complexity and expressiblity of unrestricted embedded implications have not yet been addressed. This is the main subject of the present paper.

When interpreted intuitionistically, the presence of universal quantifiers in rule bodies gives embedded implications the ability to create new constant symbols during inference. These constant symbols are hypothetical. That is, from the point of view of a top-down proof procedure, constant symbols are created temporarily in order to prove a subgoal, and are destroyed automatically after the subgoal is completed. Semantically, this amounts to a hypothetical expansion of the data domain. Computationally, the ability to create new constant symbols during inference gives the logic the power to implement counters of unbounded range and thus to simulate arbitrary Turing machines. In contrast, when embedded implications do not have this ability, they can only implement counters of polynomial range and can only simulate PSPACE-machines [4].

Finally, when embedded implications are augmented with negation-as-failure, they acquire the ability to express non-monotonic queries. A central result of this paper is that the augmented language is expressively complete; that is, it can express any database query which is typed and generic. Furthermore, with simple syntactic restrictions, the logic expresses exactly these queries, and no more. Similar results exist for other languages [7, 1, 2], but the novelty of the present work is that expressive completeness is achieved with a declarative language based on a well-established semantics, i.e., intuitionistic semantics. Indeed, the paper presents a simple proof of the intuitionistic completeness of the logic.

2 Examples

This section gives several examples of queries and rules. The examples are centered on a rulebased system which describes university policy.

In these examples, the atomic formula $take(s, c)$ intuitively means that student s has taken course c, and $grad(s)$ means that s is eligible for graduation. Conceptually, the rulebased system has two parts: a database DB containing facts such as $take(tony, cs250)$; and a rulebase R containing rules such as

$$grad(s) \leftarrow take(s, his101),\ take(s, eng201).$$

The notation $R + DB \vdash \phi$ means that the formula ϕ can be inferred from the rulebase R and the database DB. In the examples, each query is described in three ways: (i) informally in English, (ii) formally at the meta-level, and (iii) formally at the object-level.

Example 1. Consider the query, "If Tony took $cs452$, would he be eligible to graduate?" That is, if $take(tony, cs452)$ were added to the database, could we infer $grad(tony)$? This query can be formalized at the meta-level as follows:

$$R + DB + take(tony, cs452) \vdash grad(tony) \tag{1}$$

At the object level, the expression $grad(tony) \leftarrow take(tony, cs452)$ represents this query. That is, $R + DB \vdash grad(tony) \leftarrow take(tony, cs452)$ iff meta-level condition (1) is satisfied.

Example 2. "Retrieve those students who could graduate if they took one more course." i.e., at the meta-level, we want those s such that

$$\exists_c \; [R + DB + take(s, c) \vdash grad(s)]$$

The expression $\psi(s) = \exists_c[grad(s) \leftarrow take(s, c)]$ represents this query at the object-level. That is, for each value of s, $R + DB \vdash \psi(s)$ iff the meta-level condition is satisfied.

Queries such as these can be used in the premises of rules. These rules turn our query language into a logic for building rulebases.

Example 3. Consider the following university policy:

> "A student qualifies for a degree in *math and physics* if he is within one course of a degree in *math* and within one course of a degree in *physics*."

This policy can be represented as two rules:

$$within1(s, d) \leftarrow \exists_c[grad(s, d) \leftarrow take(s, c)].$$
$$grad(s, mathphys) \leftarrow within1(s, math), \; within1(s, phys).$$

Here, $grad(s, d)$ means that student s is eligible for a degree in discipline d, and $within1(s, d)$ means that s is within one course of a degree in d. Note that the premise of the first rule is a query similar to the one in example 2. [3] gives a precise sense in which such rules cannot be expressed in Datalog.

The next two examples demonstrate the use of universal quantification in queries and rule bodies.

Example 4. "Retrieve those departments for which any student can graduate by taking just one course." That is, at the meta level, we want those departments d such that

$$\forall_s \exists_c \, [R + DB + take(s, c) \vdash grad(s, d)]$$

Or, using the notation of the previous example, we want those departments d for which any student is within one course of graduation. i.e.,

$$\forall_s \, [R + DB \vdash within1(s, d)] \tag{2}$$

At the object level, this is represented by the expression $\forall_s \; within1(s, d)$. That is, for each value of d, $R + DB \vdash \forall_s \; within1(s, d)$ iff condition 2 is satisfied.

Example 5. The following rule defines a department to be "easy" if any student can obtain a degree from the department by taking history 100 and english 100.

$$easy(d) \leftarrow \forall_s[grad(s, d) \leftarrow take(s, his100), take(s, eng100)]$$

The last two examples illustrate a subtle point about our use of universal quantification: Quantification occurs not just over those students currently mentioned in the database, but over all possible students. In example 4, for instance, we are asking if a hypothetical student who has taken no courses can graduate by taking just one. The fact that every student in the database may have already taken several courses is irrelevant. For this reason, the idea of hypothetical objects is central to our interpretation of universal quantification and gives it an intuitionistic semantics. Proof theoretically, the creation of hypothetical objects means creating new constant symbols to represent them. The next section describes an inference system which does exactly this.

3 Inference

This section defines a logical inference system for embedded implications. Such systems have been developed by several researchers [11, 10, 14, 16, 18]. The one presented in this section is different from most in that it allows universal quantifiers in the body of a rule. In the logic programming context, such rules were originally treated by McCarty [15, 16]. In particular, [15] develops an intuitionistic fixpoint semantics for embedded implications, and [16] provides a practical proof-procedure.

The inference system below assumes the existence of three infinite, enumerable sets: a set of variables $x, y, z, ...$, a set of constant symbols $a, b, c, ...$, and a set of predicate symbols $A, B, C, ...$. Formulas called embedded implications are constructed from these symbols as follows.

Definition 1

- *An atomic formula is an embedded implication.*
- *If A is atomic and if $\phi_1, ..., \phi_k$ are embedded implications, then $A \leftarrow \phi_1 ... \phi_k$ is an embedded implication.*
- *If $\phi(x)$ is an embedded implication, then $\forall_x \phi(x)$ is an embedded implication.*

Embedded implications thus include formulas of the form A, $A \leftarrow B$, $A \leftarrow (B \leftarrow C)$, $A \leftarrow (B \leftarrow (C \leftarrow D))$, etc. They also include formulas with embedded universal-quantifiers, such as $A \leftarrow \forall_x B(x)$. They do not, however, include formulas with explicit disjunction or existential quantification.

We define a rulebase to be a finite set of embedded implications.[1] All free variables in a rulebase are assumed to be universally quantified at the top level.

Definition 2 *For any rulebase R and any constant symbol b,*

1. $R \vdash \phi$ *if* $\phi \in R$
2. $R \vdash \phi(b)$ *if* $R \vdash \forall_x \phi(x)$
3. $R \vdash \phi$ *if* $R \vdash \phi \leftarrow \phi_1 ... \phi_n$ *and* $R \vdash \phi_i$ *for each i.* *(Modus Ponens)*
4. $R \vdash \phi \leftarrow \phi_1 ... \phi_k$ *if* $R + \{\phi_1, ..., \phi_k\} \vdash \phi$
5. $R \vdash \forall_x \phi(x)$ *if* $R \vdash \phi(b)$ *provided b is new, i.e., does not appear in R or $\phi(x)$.*

The first three of these rules describe inference in definite Horn logic (e.g., Prolog and Datalog), and the last two extend this to an inference system for embedded implications. The fourth rule is commonly known as the deduction theorem. In the fifth rule, b does not appear in the rulebase, and so it is not given special treatment by any of the rules. Thus, if $R \vdash \phi(b)$ is derivable

[1] Besides being a practical assumption, finiteness also simplifies the theoretical development. The results in this paper can, however, be generalized to infinite rulesets in a straightforward way.

when b is new, then it should be derivable for any b. Section 4 gives a precise semantic meaning to this idea.

The rest of this section provides some lemmas and examples illustrating the basic properties of this inference system. The first lemma is an immediate consequence of rules 1, 2 and 3, and is used implicitly throughout the paper.

Lemma 1 *$R \vdash \phi$ if there is a rule $\phi' \leftarrow \phi_1 ... \phi_n$ in R and a substitution θ such that $\phi = \phi'\theta$ and $R \vdash \phi_i\theta$ for each i.*

Lemma 2 *Inference is monotonic. i.e., if $R \vdash \phi$ and $R \subseteq R'$ then $R' \vdash \phi$.*

The following two results are central to the completeness proof of section 4.3. Indeed, any inference system which has rule 1 and which satisfies these two properties is complete for intuitionistic semantics. The proofs are straightforward.

Lemma 3 *Letting b range over the universe of constant symbols,*

$$R \vdash \forall_x \psi(x) \quad \textit{iff} \quad \forall_b\ [R \vdash \psi(b)]$$

Lemma 4 *Letting Φ range over the set of all rulebases,*

$$R \vdash A \leftarrow \phi_1 ... \phi_n \quad \textit{iff} \quad \forall_\Phi\ [R + \Phi \vdash \phi_1\ \cdots\ R + \Phi \vdash \phi_n\ \Rightarrow\ R + \Phi \vdash A]$$

Example 6. Suppose R consists of the following three rules:

$$E \leftarrow \forall_x [A(x) \leftarrow C(x)]$$
$$A(x) \leftarrow B(x)$$
$$B(x) \leftarrow C(x)$$

Then $R \vdash E$, as the following top-down argument shows:

	$R \vdash E$	
if	$R \vdash \forall_x [A(x) \leftarrow C(x)]$	by lemma 1.
if	$R \vdash A(b) \leftarrow C(b)$	where b is new, by rule 5.
if	$R + C(b) \vdash A(b)$	by rule 4.
if	$R + C(b) \vdash B(b)$	by lemma 1, using the rule $A(x) \leftarrow B(x)$.
if	$R + C(b) \vdash C(b)$	by lemma 1, using the rule $B(x) \leftarrow C(x)$.

But the last line is trivially true by inference rule 1.

In this example, the constant symbol b is created during inference. In particular, it is created when rule 5 is invoked in a top-down mode. The ability to create constants in this way is crucial to the inference system.

Indeed, it is the basis of the top-down proof procedure given in [16]. In section 5, this ability is used to simulate unbounded counters and arbitrary Turing machines. Without the ability to create new constant symbols, it would only be possible to simulate Turing machines that use a polynomial amount of space, i.e., PSPACE-machines [4].

Example 7. Suppose the rulebase R contains rules defining a predicate D plus the following three rules:

$$\begin{aligned} &A \leftarrow \forall_x B(x) \\ &B(x) \leftarrow \forall_y [B(y) \leftarrow NEXT(x,y)] \\ &B(x) \leftarrow D. \end{aligned}$$

Then for any $n \geq 1$, $R \vdash A$ if

$$R + NEXT(b_1, b_2) + NEXT(b_2, b_3) + ... + NEXT(b_{n-1}, b_n) \ \vdash \ D$$

where $b_1, ..., b_n$ are new and distinct constant symbols. The proof is straightforward and is left to the reader.

This example uses recursion to generate a counter of unbounded range. The rules insert the counter into the rulebase before trying to prove D. In this way, the rules for proving D (not shown) can assume that a counter is available to them.

4 Semantics

The inference system defined in section 3 is a subset of intuitionistic logic. This section first defines intuitionistic semantics and then shows that the inference system is sound and complete with respect to it.

Intuitionistic logic can be viewed as a subset of classical logic. That is, every intuitionistic theorem is also a classical theorem, but not vice-versa. For instance, the five inference rules of our system are sound with respect to classical semantics, but not complete. To see this, consider the following simple rulebase

$$R \ = \ \{A \leftarrow (B \leftarrow C), \quad D \leftarrow A, \quad D \leftarrow C\}$$

If this rulebase is interpreted classically, then $R \models D$. In particular, from the classical definition of implication, the first rule in R is equivalent to $(A \vee \sim B) \wedge (A \vee C)$ and the last two are equivalent to $D \leftarrow (A \vee C)$. Thus $R \models D$ classically. It is not hard to see, however, that the expression $R \vdash D$ cannot be derived from inference rules 1-5. Thus, this inference system is not classical.

We now show that it is intuitionistic; that is, the inference system is both sound and complete with respect to intuitionistic semantics.

4.1 Intuitionistic Logic

This section defines the semantics of first-order intuitionistic logic in the function-free case. A more extensive treatment may be found in [9]. Recall that the syntax of the logic is first-order and that it includes three infinte sets: a set of variables $x, y, z, ...$, a set of constant symbols $a, b, c, ...$, and a set of predicate symbols $A, B, C, ...$ Each of these sets is called a *universe*.

Definition 3 *An intuitionistic structure is a quadruple $M = \langle S, \leq, \pi, dom \rangle$, where*

- *S is a non-empty set,*
- *$\leq$ is a partial order on S,*
- *π is a mapping from elements of S to sets of ground atomic formulas, and*
- *dom is a mapping from elements of S to sets of constant symbols.*

Furthermore, for any element s in S, $dom(s)$ must contain all the constant symbols appearing in $\pi(s)$; and for any two elements s_1 and s_2 in S, if $s_1 \leq s_2$ then $\pi(s_1) \subseteq \pi(s_2)$. Finally, each element, s, of S is called a substate of M, $dom(s)$ is called the domain of s, and $\cup_{s \in S} dom(s)$ is called the domain of M.

Truth in an intuitionistic structure is defined relative to its substates. That is, one can ask whether a formula ψ is true at a particular substate s of some intuitionistic structure M, written $s, M \models \psi$. In general, this expression is defined to be false if ψ contains any constant symbols not in $dom(s)$. The following definitions make these ideas precise.

Definition 4 *If dom is a set of constant symbols, then $\mathcal{F}(dom)$ denotes the set of first-order formulas containing only those constant symbols in dom. If s is a substate, then $\mathcal{F}(s)$ means $\mathcal{F}[dom(s)]$.*

Definition 5 *(Satisfaction) Suppose M is an intuitionistic structure and s is a substate of M. Then,*

$$
\begin{array}{lll}
s, M \models A & \textit{iff} & A \in \pi(s) \quad \textit{when } A \textit{ is atomic.} \\
s, M \models \psi_1 \wedge \psi_2 & \textit{iff} & s, M \models \psi_1 \quad \textit{and} \quad s, M \models \psi_2 \\
& \textit{and} & \psi_1 \wedge \psi_2 \textit{ is in } \mathcal{F}(s). \\
s, M \models \psi_1 \vee \psi_2 & \textit{iff} & s, M \models \psi_1 \quad \textit{or} \quad s, M \models \psi_2 \\
& \textit{and} & \psi_1 \vee \psi_2 \textit{ is in } \mathcal{F}(s). \\
s, M \models \sim \psi & \textit{iff} & r, M \not\models \psi \ \textit{ for all } r \geq s \\
& \textit{and} & \sim \psi \textit{ is in } \mathcal{F}(s). \\
s, M \models \psi_2 \leftarrow \psi_1 & \textit{iff} & r, M \models \psi_1 \ \Rightarrow \ r, M \models \psi_2 \ \textit{ for all } r \geq s \\
& \textit{and} & \psi_1 \leftarrow \psi_2 \textit{ is in } \mathcal{F}(s). \\
s, M \models \forall_x \psi(x) & \textit{iff} & r, M \models \psi(b) \ \textit{ for all } r \geq s \textit{ and all } b \in \textit{dom}(r) \\
& \textit{and} & \forall_x \psi(x) \textit{ is in } \mathcal{F}(s). \\
s, M \models \exists_x \psi(x) & \textit{iff} & s, M \models \psi(b) \ \textit{ for some } b \in \textit{dom}(s) \\
& \textit{and} & \exists_x \psi(x) \textit{ is in } \mathcal{F}(s).
\end{array}
$$

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

Lemma 5 $s, M \models \psi$ *iff* $r, M \models \psi$ *for all* $r \geq s$.

Definition 6 *(Models)* $M \models \psi$ *iff* $s, M \models \psi$ *for all substates* s *of* M *such that* $\psi \in \mathcal{F}(s)$*. If* $M \models \psi$*, then we say that* M *is a model of* ψ*.*

Definition 7 *(Validity) A formula* ψ *is valid iff* $M \models \psi$ *for all intuitionistic structures* M*.*

Definition 8 *(Entailment) Suppose* ψ_1 *and* ψ_2 *are formulas. Then* $\psi_1 \models \psi_2$ *iff the formula* $\psi_2 \leftarrow \psi_1$ *is valid.*

The following basic result about universal quantification is an immediate consequence of the above definitions.

Lemma 6 $M \models \forall_x \psi(x)$ *iff* $s, M \models \psi(b)$ *for all* $b \in \textit{dom}(s)$ *and all substates,* s*, of* M *such that* $\psi(x) \in \mathcal{F}(s)$

4.2 Soundness

The inference system defined in section 3 is sound with respect to intuitionistic semantics. That is,

Theorem 1 *If* $R \vdash \phi$ *then* $R \models \phi$ *for any embedded implication* ϕ*.*

To prove this theorem, it is sufficient to show that each of the five rules of inference is intuitionistically valid. In fact, the five inference rules are special cases of the following five intuitionistic theorems, respectively.

Lemma 7 *The following statements are intuitionistically valid.* α, β *and* γ *are arbitrary closed formulas, and* $\beta(x)$ *is any formula with free variable* x*.*

1. $\alpha \wedge \beta \models \alpha$
2. *If* $\alpha \models \forall_x \beta(x)$ *then* $\alpha \models \beta(b)$ *for any constant symbol* b*.*
3. *If* $\alpha \models \gamma \leftarrow \beta$ *and* $\alpha \models \beta$, *then* $\alpha \models \gamma$ *(Modus Ponens).*
4. $\alpha \models \gamma \leftarrow \beta$ *iff* $\alpha \wedge \beta \models \gamma$ *(Deduction Theorem).*
5. *If* $\alpha \models \beta(b)$ *then* $\alpha \models \forall_x \beta(x)$ *provided* b *is new, i.e., does not appear in* α *or* $\beta(x)$*.*

The proofs of these five statements are not difficult. The first four follow in a straightforward way from the definitions of intuitionistic satisfaction and entailment. The fifth statement is closely related to theorems given in [9] and can be proved along similar lines.

4.3 Completeness

This section establishes the completeness of the inference system defined above using a construction inspired by the Henkin constructions of modal logic [12]. Given a set R of embedded implications, we define an intuitionistic structure M_R called the *canonical model* of R. This structure, defined proof-theoretically, provides the necessary link between inference and semantics.

Definition 9 *The canonical model of a rulebase* R *is an intuitionistic structure* $M_R = \langle S, \leq, \pi, dom \rangle$, *where*

$$S = \{\Phi \mid \Phi \text{ is a finite rulebase}\}$$
$$dom(\Phi) = \mathcal{U}, \text{ the universe of constant symbols}$$
$$\pi(\Phi) = \{A \mid R + \Phi \vdash A \text{ and } A \text{ is atomic}\}$$
$$\Phi_1 \leq \Phi_2 \text{ iff } \Phi_1 \subseteq \Phi_2$$

Notice in particular that the domain of each substate is the same. Not only does this lead to a simple proof of completeness, it also has interesting semantic consequences for embedded implications, as we shall see.

Without loss of generality, we an assume that $\mathcal{U}$ contains infinitely more constant symbols than appear in R, even if R is infinite. Thus, $\mathcal{U}$ always contains a constant symbol not in $R + \Phi$, since Φ is finite. This assumption is neccessary in order to invoke invoke lemma 3 in the proof of the following theorem. This theorem is the main result about canonical models.

Theorem 2 *If ϕ is an embedded implication, then $\Phi, M_R \models \phi$ iff $R + \Phi \vdash \phi$ for any substate Φ of M_R.*

Proof: (by induction on the structure of ϕ.)

Basis: When ϕ is atomic, the lemma follows immediately from definition 9.

Induction: There are two cases, depending on the form of ϕ.

(i) Suppose $\phi = A \leftarrow \phi_1 ... \phi_n$ where A is a ground atomic formula and each ϕ_i is an embedded implication. Then,

$$\begin{array}{ll} & R + \Phi \vdash A \leftarrow \phi_1 ... \phi_n \\ \textit{iff} & \forall_{\Phi' \geq \Phi} \, [R + \Phi' \vdash \phi_1 \;\cdots\; R + \Phi' \vdash \phi_n \;\Rightarrow\; R + \Phi' \vdash A] \qquad \text{by lemma 4.} \\ \textit{iff} & \forall_{\Phi' \geq \Phi} \, [\Phi', M_R \models \phi_1 \;\cdots\; \Phi', M_R \models \phi_n \;\Rightarrow\; \Phi', M_R \models A] \\ & \quad \text{by induction hypothesis.} \\ \textit{iff} & \forall_{\Phi' \geq \Phi} \, [\Phi', M_R \models \phi_1 \wedge ... \wedge \phi_n \;\Rightarrow\; \Phi', M_R \models A] \\ \textit{iff} & \Phi, M_R \models A \leftarrow \phi_1 ... \phi_n \qquad \text{by definition.} \end{array}$$

(ii) Suppose $\phi = \forall_x \psi(x)$. Then,

$$\begin{array}{lll} & R + \Phi \vdash \forall_x \psi(x) & \\ \textit{iff} & \forall_{a \in \mathcal{U}} \; R + \Phi \vdash \psi(a) & \text{by lemma 3.} \\ \textit{iff} & \forall_{a \in \mathcal{U}} \; \Phi, M_R \models \psi(a) & \text{by induction hypothesis.} \\ \textit{iff} & \forall_{\Phi' \geq \Phi} \forall_{a \in \mathcal{U}} \; \Phi', M_R \models \psi(a) & \text{by lemma 5.} \\ \textit{iff} & \forall_{\Phi' \geq \Phi} \forall_{a \in dom(\Phi')} \; \Phi', M_R \models \psi(a) & \text{since } dom(\Phi') = \mathcal{U} \text{ for every } \Phi'. \\ \textit{iff} & \Phi, M_R \models \forall_x \psi(x) & \text{by definition.} \end{array}$$

QED

Completeness follows almost immediately from this result. We first note that for any substate Φ of M_R, the domain of Φ is the entire universe of constant symbols. Thus any formula ϕ is in $\mathcal{F}(\Phi)$. The following lemma is thus an immediate consequence of definition 6.

Lemma 8 *If ϕ is a formula, then $M_R \models \phi$ iff $\Phi, M_R \models \phi$ for all substates Φ.*

Corollary 1 *If ϕ is an embedded implication, then $M_R \models \phi$ iff $R \vdash \phi$.*

Proof:

	$R \vdash \phi$	
iff	$R + \{\} \vdash \phi$	
iff	$\{\}, M_R \models \phi$	by theorem 2.
iff	$\forall_{\Phi \geq \{\}}\ \Phi, M_R \models \phi$	by lemma 5.
iff	$\Phi, M_R \models \phi$	for all substates Φ of M_R.
iff	$M_R \models \phi$	by lemma 8.

QED

Thus $M_R \models \phi$ for each rule ϕ in R, since $R \vdash \phi$ by inference rule 1. Hence, treating R as a conjuction of formulas, we get:

Corollary 2 *$M_R \models R$, that is, M_R is a model of R.*

Corollary 3 *(Completeness) If $R \models \phi$ then $R \vdash \phi$ for any embedded implication ϕ.*

Proof: Note that by two results above, $\Phi, M_R \models R$ for every substate Φ. Thus,

if	$R \models \phi$	
then	$M \models \phi \leftarrow R$	for any intuitionistic structure M, by definition 8.
	$M_R \models \phi \leftarrow R$	since M_R is an intuitionistic structure.
	$\Phi, M_R \models \phi \leftarrow R$	for any substate Φ, by lemma 8.
	$\Phi, M_R \models \phi$	since $\Phi, M_R \models R$
	$R + \Phi \vdash \phi$	by theorem 2.
	$R + \{\} \vdash \phi$	using $\Phi = \{\}$.
	$R \vdash \phi$	

QED

In general, the domain of an intuitionistic structure M may vary from substate to substate, but as pointed out earlier, the canonical model M_R has a constant domain. This has interesting semantic consequences for embedded implications. In general, $R \models \phi$ iff $M \models R \leftarrow \phi$ for every intuitionistic structure M. However, from the above results, it follows that if R and ϕ are (sets of) embedded implications, then $R \models \phi$ iff $M_R \models \phi$. Because M_R is a model of R with constant domain, we obtain the following:

Lemma 9 *If R is a rulebase and ϕ is an embedded implication, then* $R \models \phi$ *iff* $M \models \phi$ *for all models M of R with constant domain.*

Thus, in deciding whether R entails ϕ, it is possible to focus only on those intuitionistic structures which (i) are models of R, and (ii) have constant domain. Neither of these properties is true of intuitionistic entailment in general. Indeed, the possibility of variable domains is the source of some interesting differences between classical and intuitionistic logic [9]. Many of these differences do not appear when reasoning only with embedded implications.

5 Semi-Decidability

This section shows that the inference system defined in section 3 is semi-decidable. That is, the problem of determining whether the expression $R \vdash \phi$ can be derived is in re but is non-recursive. The upper bound (being in re) is easy, and most of the section is devoted to showing non-recursiveness. The main idea is to construct a rulebase which encodes the transition function of an arbitrary Turing machine. Besides establishing undecidability, these encodings are the basis of the expressibility results in the next section. The encodings themselves are an adaptation of those used in [4] to encode PSPACE machines.

To show that inference is in re, it is sufficient to note the following: (i) that the set of all possible rulebases can be enumerated, as can the set of all embedded implications, and (ii) that the set $\{R \vdash \phi \mid R \vdash \phi \textit{ is derivable}\}$ can be enumerated as in any inference system, by applying the inference rules in a bottom-up fashion to all rulebases R and all embedded implications ϕ.

To show that inference is non-recursive, suppose that M is a Turing machine and $\overline{s}$ is an input string. We encode M as a rulebase $R(M)$ and $\overline{s}$ as a set of ground atomic formulas, that is, as a database $DB(\overline{s})$. These encodings are constructed so that

$$R(M) + DB(\overline{s}) \vdash ACCEPT \quad \textit{iff} \quad M \textit{ accepts } \overline{s}, \tag{3}$$

where $ACCEPT$ is a zero-ary predicate symbol. Non-recursiveness follows by choosing M to be a machine that accepts a non-recursive language.

It is not necessary to use the full syntax of the logic to build $R(M)$. In fact, we shall use rules satisfying the following syntactic restrictions:

- Implication is nested to a depth of at most two. Thus, $A \leftarrow (B \leftarrow C)$ is allowed, but the rule $A \leftarrow (B \leftarrow (C \leftarrow D))$ is not.

- Any variable appearing in the head of a rule must also appear in an atomic premise in the body. Thus $A(x) \leftarrow B(x), [C(x) \leftarrow D(x)]$ is allowed, but the rule $A(x) \leftarrow [C(x) \leftarrow D(x)]$ is not, nor is the rule $A(x, y) \leftarrow B(x), [C(x) \leftarrow D(x)]$.

- The rules do not contain any constant symbols. All constant symbols are placed in the database.

These restrictions, and especially the last two, are central to the expressibility results in section 6.

The rest of this section constructs the rulebase $R(M)$ which encodes the computations of Turing machine M and which satisfies formula (3).

5.1 Representing the Turing Machine

Without loss of generality, we assume that M is a Turing machine with a single, semi-infinite tape. That is, the tape starts at some point and extends infinitely far to the right. To represent positions on the tape as well as points in time, we use a binary predicate $NEXT(i, j)$ to encode a counter. If the input to the Turing machine is a string of length n, then the following formulas are placed in the database $DB(\overline{s})$ to initialize the counter:

$$NEXT(a_1, a_2), \quad NEXT(a_2, a_3) \ ... \ NEXT(a_{n-1}, a_n),$$

where $a_1...a_n$ are distinct constant symbols. During inference, new constant symbols will be created and the range of the counter extended (as in example 7) in order to represent new points in time and new positions on the tape. Indeed, the range of the counter will be increased by 1 for each step of the machine's computation. In this way arbitrary amounts of time and tape can be represented.

Given such a counter, the Turing machine M can be represented by the following predicates:

- $HEAD^q(j, t)$: at time t, the control head is in state q and is scanning tape cell j.

- $CELL^c(j, t)$: at time t, tape cell j contains symbol c.

By defining distinct predicates $HEAD^q$ and $CELL^c$ for each control state q and tape symbol c, the rulebase $R(M)$ is kept constant free. For each value of the time variable t, these predicates represent an instantaneous description of the Turing machine (an id).

The same counter is used to represent both time and tape; however, they have different starting points. In particular, a_1 denotes the first position on the tape, and a_n denotes the first point in time. More generally, as the range of the counter increases during inference, the highest value of the counter always denotes the current time. This property is a technical convenience which simplifies the overall encodings. The different starting points for tape and time are represented by the following two formulas, which are added to the database $DB(\overline{s})$:

$$FIRST_{tape}(a_1), \quad FIRST_{time}(a_n).$$

With this in mind, we add the following formulas to the database $DB(\overline{s})$ to represent the initial *id* of the machine:

$$\begin{aligned} &CELL^{s_1}(a_1, a_n), \quad CELL^{s_2}(a_2, a_n) \ldots CELL^{s_n}(a_n, a_n), \\ &HEAD^{q_0}(a_1, a_n), \end{aligned}$$

where s_i is the i^{th} character of the input string $\overline{s}$. The last formula says that the control head is initially in state q_0 and is scanning the first tape cell.

5.2 Simulating the Computation

During inference, the rulebase $R(M)$ simulates the computations of the Turing machine M. Inference rules 4 and 5 are central to this simulation, and they play distinct roles.

Rule 5 creates new constant symbols, that is, constants which do not already appear in the rulebase. These new symbols are used to extend the range of the time and tape counters. As their range is extended, more machine *id*'s can be represented.

Rule 4 adds formulas to a rulebase. This capability is used to record the computation path of the machine. A computation path is a sequence of successive *id*'s. As the machine steps from one *id* to the next, a set of atomic formulas is added to the rulebase to represent the new *id*. Inference halts when an accepting *id* is generated, that is, an *id* in which the control head is in an accepting state.

Specifically, let $DB(t)$ denote the set of atomic formulas representing the machine *id* at time t. If $t_0, t_1, ..., t_m$ denote the first m instants of time, then the set $DB(t_0) + DB(t_1) + \cdots + DB(t_m)$ represents the first m steps of the computation path. With this in mind, we define a predicate $ACC(t)$ with the following property:

$$\begin{aligned} &R(M) + DB(t_0) + \cdots + DB(t_m) \ \vdash \ ACC(t_m) \\ &\quad \textit{iff} \quad DB(t_m) \ \textit{represents an accepting id} \\ &\quad \textit{or} \quad R(M) + DB(t_0) + \cdots + DB(t_m) + DB(t_{m+1}) \ \vdash \ ACC(t_{m+1}), \end{aligned} \tag{4}$$

where $DB(t_0) = DB(\overline{s})$ denotes the initial *id* of the machine.

The operation of the rulebase $R(M)$ is easiest to understand if we imagine that the formula $ACC(t_0)$ is derived in a top-down manner. That is, in trying to infer $ACC(t_0)$, the rules in $R(M)$ first examine the database $DB(t_0)$ to determine whether it represents an accepting *id*. If it does not, then a new constant symbol t_1 is created, and a new database $DB(t_1)$ is generated and added to the rulebase. The process then repeats. In trying to infer $ACC(t_1)$, the rules examine $DB(t_1)$ to determine whether it represents an accepting *id*. If it does not, then a new constant symbol t_2 is created, and a new database $DB(t_2)$ is generated and added to the rulebase. This process is repeated until an accepting *id* is generated.

If the machine M accepts its input, then an accepting *id* will eventually be reached. In this case, the computation halts and the atomic formula $ACC(t_0)$ is derived from $R(M) + DB(t_0)$. If the input is not accepted, then an accepting *id* is never generated. In this case, either the computation never terminates or it halts at some *id*, $DB(t_m)$, for which it is not possible to generate a successor *id*, $DB(t_{m+1})$. In either case, the rulebase fails to derive $ACC(t_0)$. Thus by induction,

$$R(M) + DB(t_0) \vdash ACC(t_0) \quad \textit{iff} \quad M \textit{ halts and accepts},$$

where t_0 denotes the first instant of time.

Since $DB(t_0) = DB(\overline{s})$, formula (3) is satisfied by placing the following rule into the rulebase $R(M)$, where the expression $FIRST_{time}(t)$ binds t to the constant symbol denoting the first instant of time:

$$ACCEPT \leftarrow FIRST_{time}(t),\ ACC(t).$$

In this way, by providing rules which define the predicate $ACC(t)$, we can show that inference with embedded implications is non-recursive.

5.3 Encoding the Transition Function

This section develops rules defining the predicate $ACC(t)$ so that it satisfies formula (4). These rules encode the transition function of the machine M. There are two types of rules: those which detect a halting state, and those which encode the transition of the machine from one *id* to the next.

The first type of rule is straightforward. For each accepting state q_a, the following rule is added to the rulebase $R(M)$:

$$ACC(t) \leftarrow HEAD^{q_a}(j,t).$$

During inference, the variable t is bound to the time at which the accepting state is reached, and the variable j is bound to the position of the control head at that time.

The second type of rule is more complex. For each entry in the machine's transition function, we write a single rule. For instance, suppose the machine has the following transition:

> If the control head is in state q and is scanning the symbol c, then write symbol c' onto the tape, move the control head one cell to the right, and put it into state q'.

The following rule encodes this transition:

$$\begin{aligned} ACC(t) \quad \leftarrow \quad & HEAD^q(j,t),\ CELL^c(j,t),\ NEXT(j,j'), \\ & \forall_{t'}[ACC(t') \leftarrow NEXT(t,t'),\ HEAD^{q'}(j',t'),\ CELL^{c'}(j,t')]. \end{aligned}$$

During inference, the body of this rule does several things. The first line determines whether the control head is in state q and scanning symbol c. If so, it extracts the position of the head j and computes its next position j'. The second line is universally quantified and does two things. First, it creates a new constant symbol t'. Just as the symbol t denotes the current instant of time, the new symbol t' will denote the next instant of time. Second, the body of the embedded implication adds new atomic formulas to the rulebase. In particular, it adds t' to the end of the counter, thereby extending its range. It also records the new state and position of the control head and the new contents of tape cell j. In this way, these rules simulate the movement of the Turing machine from one *id* to the next.

These rules determine the *changes* caused to a *id* by a machine transition. The greater part of an *id*, however, remains unchanged by such transitions. Indeed, except for the tape cell under the control head, the contents of all tape cells are unchanged. This is an instance of the *frame axiom*, and rules are needed to encode it. Such rules are necessary because we are representing time explicitly and must therefore copy the unchanged portion of an *id* from one instant of time to the next. The rules themselves are straightforward. Details may be found in [5].

These rules complete the definition of the rulebase $R(M)$, thereby establishing formula (4), and thus formula (3).

6 Expressive Completeness

This section views our inference system as a query language for relational databases and investigates its expressibility. In section 5, it was shown that non-recursive queries could be expressed. However, because the logic is monotonic (lemma 2), there are still many queries which it cannot express, such as relational-algebra queries involving complementation. By augmenting the logic with negation-as-failure, non-monotonic queries can be expressed as well. In fact, this augmented logic is expressively complete; that is, it can express any typed, generic query. This is the main result of this section.

The proof itself is an adaptation of proofs found in [4] and [6]. In [4] it is shown that when universal quantifiers are *not* allowed in the premise of a rule, then embedded implications express exactly the typed, generic queries in PSPACE. In [6], syntactic restrictions are identified which enable embedded implications to express exactly the queries in NP and the queries in each level of the polynomial time hierarchy. The interested reader is referred to these papers for details which, for reasons of space, have not been included in this paper.

In introducing negation, a certain amount of care is required, since negation can easily take inference outside of the recursively enumerable sets, making it fully undecidable. For this reason, we restrict negation to those predicates which are defined purely in terms of Horn rules, that is, to predicates P whose definition would not change if all non-Horn rules were removed from the rulebase. This is sufficient to make inference complete while keeping it within re.

The following example demonstrates some of the expressive power of embedded implications. In particular, it uses embedded implications with negation to implement the query $EVEN$, a query which cannot be expressed in Datalog with negation [8].

Example 8. Suppose that DB is a set of ground atomic formulas of the form $A(\overline{x})$. Suppose also that R is the following collection of rules:

$$
\begin{aligned}
&EVEN \leftarrow SELECT(\overline{x}),\ [ODD \leftarrow B(\overline{x})].\\
&ODD \leftarrow SELECT(\overline{x}),\ [EVEN \leftarrow B(\overline{x})].\\
&EVEN \leftarrow \sim SOMELEFT.\\
&SOMELEFT \leftarrow SELECT(\overline{x}).\\
&SELECT(\overline{x}) \leftarrow A(\overline{x}),\ \sim B(\overline{x}).
\end{aligned}
$$

Then $R + DB \vdash EVEN$ iff DB has an even number of elements.

In this example, the rulebase determines the parity of a relation A. From the perspective of top-down inference, the rulebase introduces a new relation B, initially empty, and hypothetically copies A to B one tuple at a time. As tuples are copied to B, the first two rules "flip back and forth" between the two subgoals $EVEN$ and ODD. During this phase, the fifth rule selects tuples from A which are not yet in B. When A has been completely copied to B, the third and fourth rules infer that $EVEN$ is true. Thus, $EVEN$

recursively determines the parity of the difference relation $A \sim B$, with recursion terminating when the difference is empty and the parity is even.

Note that it does not matter in which order the elements of A are copied to B. Every order will give the same answer: either every order will result in a proof of $EVEN$ or every order will result in a proof of ODD. This idea of order independence is important in the proof of our expressibility result.

6.1 The Main Result

This section gives a precise statement of our expressibility result. Central to this result are the notions of typed databases and of generic database queries. These notions have been defined precisely by Chandra and Harel [7, 8], and the definitions are repeated here. Syntactic restrictions are then developed which guarantee that a set of embedded implications will implement a generic database query. Finally, a precise statement of the main result is given. Informally, this result provides a syntactic characterization of the generic database queries.

Definition 10 *(Relational Database) Let U be a countable set, called the universal data domain. A relational database DBof type $\overline{\alpha} = (\alpha_1, ..., \alpha_m)$ is a tuple $(D, R_1, ..., R_m)$ where D is a finite subset of U and R_i is an α_i-ary relation over D, i.e., $R_i \subseteq D^{\alpha_i}$. D is called the (data) domain of DB, written dom(DB).*

In logical systems such as ours, a relational database is represented as a set of ground atomic formulas. U is a universal set of constant symbols, and for each relation R_i there is a predicate symbol P_i whose ground atomic formulas represent R_i. In addition, this section assumes the existence of a monadic predicate D which gives the user access to the data domain, i.e., $DB \vdash D(x)$ iff $x \in dom(DB)$.

Definition 11 *(Generic Query) A generic query of type $\overline{\alpha} \rightarrow \alpha_0$ is a partial recursive function ψ which takes a database DBof type $\overline{\alpha}$ and returns a relation $\psi(DB)$ over $dom(DB)$ of arity α_0. In addition, ψ must satisfy the following consistency criterion: if DB' can be derived from DBby a renaming (i.e., a permutation) of the symbols in U, then $\psi(DB')$ can be derived from $\psi(DB)$ by the same renaming.*

If one predicate symbol is reserved for the output relation, then a rulebase defines a typed database query. Not all such queries are generic however. Intuitively, genericity means that a query treats all constant symbols as equal. Thus, a rulebase represents a non-generic query if the rules give special treatment to some constant symbols. This possibility can be avoided by using rules that are constant free, i.e., which contain variables but no constant symbols.

Genericity also means that a query's output relation contains only those constant symbols appearing in the database. Thus, a query is non-generic if it is unsafe, i.e., if the output relation is infinite. This can occur when variables appear in the head of a rule but not the body. For this reason, such rules are often barred from Datalog. This restriction is not enough, however, to guarantee the safety of embedded implications. Suppose, for instance, that a rulebase contains the following two rules: $B(x) \leftarrow C(x)$ and $A(x) \leftarrow [B(x) \leftarrow C(x)]$. Then $A(b)$ is true for all constant symbols b, regardless of the data domain. This possibility can be avoided by a simple syntactic restriction: in any embedded implication of the form $A \leftarrow \phi_1 ... \phi$, we require that all variables in the head must also appear in an atomic premise in the body. Thus the rule $A(x) \leftarrow D(x), [B(x) \leftarrow C(x)]$ is allowed, but the rule $A(x) \leftarrow [B(x) \leftarrow C(x)]$ is not. This guarantees safety and genericity.

The following definition summarizes the above considerations.

Definition 12 *$\mathcal{QL}$ is a set of rulebases, where each rulebase is a finite set of embedded implications with negation-as-failure. Furthermore a rulebase R is in $\mathcal{QL}$ iff it satisfies all of the following conditions:*

- *Only predicates defined solely in terms of Horn rules are negated.*
- *There are no constant symbols in the rulebase.*
- *In every embedded implication, any variable in the head must also appear in a non-negated atomic premise in the body.*

The following theorems are the main result of this section.

Theorem 3 *For any rulebase R in $\mathcal{QL}$, let ψ_R be the query defined as follows:*

$$\overline{x} \in \psi_R(DB) \quad \textit{iff} \quad R + DB \vdash OUT(\overline{x}).$$

Then ψ_R is a typed, generic query.

Theorem 4 *If ψ is a typed, generic query, then there is a rulebase R_ψ in $\mathcal{QL}$ such that for any database DB,*

$$R_\psi + DB \vdash OUT(\overline{x}) \quad \textit{iff} \quad \overline{x} \in \psi(DB).$$

Corollary 4 *The set of relational queries expressed by $\mathcal{QL}$ is equal to the set of typed, generic queries.*

Theorem 3 is straightforward. In particular, the syntactic restrictions defining $\mathcal{QL}$ ensure that every rulebase in $\mathcal{QL}$ expresses a query which is typed and generic. Theorem 4, however, is more difficult and relies on the encodings of Turing machines developed in section 5.

6.2 Sketch of Proof

This section outlines the proof of theorem 4. The proof is a simple adaptation of that in [5], to which the interested reader is referred for details.

The first step is to reduce the problem of expressing typed queries to that of expressing yes/no queries. Whereas a typed query returns a set of tuples, a yes/no query simply returns *true* or *false*. This reduction is necessary in order to use the Turing-machine encodings of section 5.

Lemma 10 *Let φ be a generic yes/no query, that is, a generic query of type $\overline{\alpha} \rightarrow 0$. Then there is a rulebase R_φ in $\mathcal{QL}$ such that for all databases DB of type α,*

$$R_\varphi + DB \vdash YES \quad \textit{iff} \quad \varphi(DB) = true,$$

where YES is a predicate symbol of arity zero.

Theorem 4 follows from this lemma in a straightforward way. The next step is to prove this lemma by constructing the rulebase R_φ.

Since φ is a generic yes/no query, it is partial recursive, and hence there is a Turing machine M which accepts the language $\{DB \mid \varphi(DB) = true\}$. We construct the rulebase R_φ so that during inference, it simulates the computations of machine M. Specifically, when given a database DB, R_φ performs the following steps:

1. It encodes the database as bits on the input portion of the machine tape.

2. It simulates machine M step by step.

3. It infers YES iff M halts and accepts its input.

Section 5 showed how to construct a rulebase in $\mathcal{QL}$ which encodes the computations of an arbitrary Turing machine. This takes care of steps 2 and 3. To use this construction, however, step 1 must be completed. That is, R_φ must contain rules which generate a representation of the database DB in terms of the predicates $CELL^c(j,t)$, where c denotes a tape symbol, j denotes tape position, and t denotes time. Embedded implications and negation-as-failure are essential to these rules. It is not necessary, however, to use universal quantifiers in the rule bodies, and consequently, the rules are similar to those found in [5]. The interested reader is thus referred to this report for details.

7 Discussion

This paper has identified a subset of intuitionistic logic which expresses exactly the generic database queries. Syntactically, it is an extension of Datalog in which rules themselves may appear in the bodies of other rules. Thus, rules of the form $A(x) \leftarrow \forall_y[B(x,y) \leftarrow C(x,y)]$ are allowed. Such rules are called *embedded implications*. When interpreted intuitionistically, these rules have the ability to create new constant symbols during inference.

In their ability to create and manipulate new constant symbols, embedded implications are similar to other database query languages. Abiteboul and Vianu, in particular, have developed a family of procedural and declarative languages with such capabilities [1, 2]. The language presented in this paper is comparable to their Deterministic Declarative Language *detDL* in that constant symbols created during inference will not appear in the answer to a query. Both languages also have the power of arbitrary Turing machines, both are expressively complete, and both employ simple syntactic restrictions to guarantee that queries are safe, i.e., have finite answers.

However, the work presented in this paper is different from the above in several respects. First, the power to create constant symbols does not come from a new proof-theoretic mechanism; rather, it is a consequence of the intuitionistic semantics. Second, constant symbols created during inference may appear in any argument position of any predicate without jeopardizing safety. Perhaps the biggest difference, though, is conceptual: The constant symbols created by embedded implications are hypothetical; that is, they are created in order to prove a subgoal, and are destroyed when this goal is completed. Semantically, this amounts to a hypothetical expansion of the data domain.

The investigation of embedded implications is a natural continuation of other work on hypothetical reasoning. In particular, in [4, 6], the language *Hypothetical Datalog* is developed, in which entries can be hypothetically added or removed from a database during inference. This paper has augmented this work by studying the hypothetical expansion of the data domain as well as the hypothetical addition of arbitrary rules.

References

[1] S. Abiteboul and V. Vianu. A Transaction Language Complete for Database Update Specification. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 260–268, 1987. Published in expanded form as Rapports de Recherche no. 715, INRIA, 78153 Le Chesnay Cedex, France.

[2] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. Technical Report 900, Institut National de Recherche en Informatique et en Automatique (INRIA), Le Chesnay Cedex, France, 1988.

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

[4] A.J. Bonner. Hypothetical Datalog: Complexity and Expressibility. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 144–160. Springer-Verlag, 1988. Published as volume 326 of *Lecture Notes in Computer Science.*

[5] A.J. Bonner. Hypothetical Datalog: Complexity and Expressibility. Technical Report DCS-TR-231, Department of Computer Science, Rutgers university, New Brunswick, NJ 08903, 1988. Also published in volume 76 (1990) of *Theoretical Computer Science (TCS)*, pages 3–51.

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

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

[8] A.K. Chandra and D. Harel. Structure and Complexity of Relational Queries. In *Proceedings of the Symposium on the Foundations of Computer Science (FOCS)*, pages 333–347, 1980.

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

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

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

[12] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic.* Methuen and Co. Ltd., London, 1968.

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

[14] Sanjay Manchanda. *A Dynamic Logic Programming Language for Relational Updates*. PhD thesis, Department of Computer Science, State University of New York at Stony Brook, Stony Brook, New York, December 1987. Also published as Technical Report TR 88-2, Department of Computer Science, The University of Arizona, Tuscon, Arizona 85721, January, 1988.

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

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

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

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

[19] G. Nadathur and D. Miller. An Overview of λProlog. In R.A. Kowalski and K.A. Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 810–827, Cambridge, MA, 1988. MIT Press.

[20] D.S. Warren. Database Updates in Pure Prolog. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 244–253, 1984.