J. LOGIC PROGRAMMING 1997, 32(2):119-170 119

A LOGICAL SEMANTICS FOR HYPOTHETICAL RULEBASES WITH DELETION

ANTHONY J. BONNER

▷ This paper addresses a limitation of most deductive database systems: they cannot reason hypothetically. Although they reason effectively about the world as it is, they are poor at tasks such as planning and design, where one must explore the consequences of hypothetical actions and possibilities. To address this limitation, we have developed a logic-programming language in which users can create hypotheses and draw inferences from them. Most previous work in this area has focussed on the hypothetical *insertion* of facts into a database, since insertion is accounted for by a well-established logic: intuitionistic logic. In contrast, our language includes hypothetical *deletion* as well as insertion. In earlier work, we established the data complexity and expressibility of this language. In this paper, we develop its logical semantics, and take a closer look at its expressibility. The paper makes three main contributions. First, we show that hypothetical queries lead naturally to a new notion of expressibility. In this new light, we show that classical logic is poor at hypothetical reasoning, since it cannot express some simple hypothetical queries. Second, we develop a logical semantics for hypothetical insertions and deletions, including a proof theory, model theory, and fixpoint theory. We also give numerous examples showing the utility of the logic and the subtle effect that deletion has on its expressive power. Finally, we augment the logic with negation-as-failure, so that non-monotonic queries can be expressed. We then develop the proof theory and model theory for the logic with negation. The proof theory is inspired by the stratified semantics of Apt, Blair and Walker, and the model theory is inspired by the perfect model semantics of Przymusinski. ◁

Address correspondence to University of Toronto, Department of Computer Science, Toronto, Ontario, Canada M5S 3G4. Email: bonner@db.toronto.edu

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010

1. INTRODUCTION

Researchers from several areas have recognized the need for computer systems that reason hypothetically. Decision support systems (DSS) are a good example, especially in domains like financial planning where many "what if" scenarios must be considered [41, 53]. A typical example is an analyst who must predict a company's deficit for the upcoming year assuming that employee salaries are increased by a given percentage. Or he might want a table of deficit predictions for a number of hypothetical salary increases [66, 67]. Similar problems occur in computer aided design (CAD). Here, one must evaluate the effect on the overall design of local design alternatives and of various external factors [30, 58]. For example, an engineer may need to know how much the price of an automobile would increase if supplier X raised his prices by Y percent [30]. The number of hypothetical scenarios multiplies quickly when several factors are varied simultaneously, such as prices, interest rates, tax rates, etc. One may also need to consider variations in more complex factors, such as government regulations, company policy, tax laws, etc.

The database community has addressed some of these needs by developing systems that integrate query processing with hypothetical updates. Such systems allow a user to pose queries not only to the real database, but also to hypothetical databases. Hypothetical databases are derived from a real database by a series of hypothetical assumptions, or updates. Early work in this area was done by Stonebraker, who showed that hypothetical databases can be efficiently implemented by slight extensions to conventional database mechanisms [61, 60]. He pointed out that hypothetical databases are useful for debugging purposes, for generating test data, and for carrying out a variety of simulations. He also argued that "there are advantages to making hypothetical databases central to the operation of a database management system" [60].

The logic-programming community has taken these ideas one step further, integrating hypothetical updates not just with query processing, but with logical inference as well. Since the premise of a logical rule is just a query, several researchers have developed *hypothetical rules*, in which the premise can query not only a real database, but hypothetical databases as well. Vieille, et al, for instance, have developed a deductive database along these lines [66, 67], and Warren and Manchanda have used hypothetical rules to reason about database updates [68, 44]. In [50], Miller shows that hypothetical insertions can structure the runtime environment of logic programs, resulting in programs that are more elegant, more efficient, and easier to maintain. In [51], he develops a theory of lexical scoping based on the hypothetical creation of constant symbols during inference.

These logical systems are well suited to solving problems in Artificial Intelligence, especially problems that involve reasoning about alternative courses of action. For example, an AI program may need to infer that *if* the pawn took the knight, *then* the rook would be threatened. The program may also have to consider sequences of possible moves, exploring hypothetical possibilities to great depth. Hypothetical inference has also extended the capabilities of expert systems. Gabbay and Reyle, for instance, have reported a need to augment Prolog with hypothetical rules in order to encode the British Nationality Act, because the act contains rules such as, *You are eligible for citizenship if your father would be eligible if he were still alive* [33]. And McCarty, also motivated by legal applications, has developed a wide class of hypothetical rules for computer-based consultation systems, especially systems for reasoning about corporate tax law and estate tax law [47, 49, 57].

Theoretical work on hypothetical inference has also been carried out, largely by the logic-programming community. Most of this work focuses on the hypothetical *insertion* of atoms into a database. One reason for this focus is that hypothetical insertion fits neatly into a well-known logical system: *intuitionistic logic* [29]. Gabbay was the first to show that intuitionistic logic models hypothetical insertion [32]. Working independently, McCarty and Miller extended this result to operations that create new constant symbols during inference, and they developed fixpoint semantics based on intuitionistic logic [47, 50]. Earlier, Statman showed that inference in intuitionistic logic is PSPACE-complete in the propositional case [59]. In [10, 11], we show that the complexity of answering queries with hypothetical insertions is PSPACE-complete in the function-free predicate case (*i.e.*, the case of greatest interest to databases systems).

In [10], we introduced a logic called *Hypothetical Datalog* that performs hypothetical *deletion* as well as insertion. In that work, we established theoretical results on the complexity and expressibility of the logic. We showed, for instance, that its data complexity is complete for EXPTIME. By augmenting the logic with negation-as-failure, and by considering stratified rulebases, the logic expresses all database queries in EXPTIME [10] in the sense defined by Chandra and Harel [20]. In [10, 9, 7, 11, 12], we extended these results to lower complexity classes—PSPACE, PHIER, Σ_k^P and NP—for various fragments of the logic. It is interesting that all these complexity classes can be characterized by a single language with a natural series of syntactic restrictions. In those papers, we treated hypothetical inference operationally, especially in the presence of negation-as-failure. In this paper, we complete the picture by developing a logical semantics for Hypothetical Datalog. In the tradition of logic programming, we develop several semantics and show that they are equivalent. These semantics include a proof theory (*i.e.*, a logical inference system), a model theory, and a fixpoint theory.

We also show that hypothetical queries lead naturally from the traditional notion of *database* query to a more general notion of *rulebase* query. Intuitively, rulebase queries are queries posed to a deductive database system, that is, to a relational database augmented with a set of logical rules (the rulebase). Just as a database query maps *relational* databases to relations, a rulebase query maps *deductive* databases to relations. The notion of rulebase queries finds it motivation in expert systems, and is based on a principle that we call *rulebase independence*. Originally introduced in [8, 11], rulebase independence is a condition that we place on queries to a deductive database. Intuitively, a user should be able to formulate queries without having a detailed knowledge of the rulebase. For example, if the rulebase encodes the British Nationality Act, then a user should be able to formulate queries (such as, *Am I eligible for citizenship?*) even though he does not understand the myriad of rules defining nationality. Similar requirements exist in relational database systems, since a user must be able to formulate database queries without knowing the exact contents of the database. Using this idea, we show that classical logic is inadequate for hypothetical reasoning, since it cannot express some simple hypothetical queries in a rulebase-independent way. Thus, if hypothetical queries are to be expressed declaratively, then new logical mechanisms are needed.

This paper develops those mechanisms in the form of a logic programming language called Hypothetical Datalog. This language combines logical inference with hypothetical database updates. The basic building blocks of the language are hypothetical queries, which allow a user to pose queries not only to the real database, but to hypothetical databases as well. A hypothetical query updates the database, evaluates a query on the new database, and then returns the database to its original state. For example, consider the following hypothetical query: *If the white knight at position X were moved to position Y, would the black king be in check?* To answer this query, the system would first delete the fact that the knight is at position X, and insert the fact that it is at position Y; then, it would ask whether the king is in check; and finally, it would return the knight to position X. Although this process might seem procedural, we show how to formulate it as a declarative language with a logical semantics. In this way, the database is *automatically* restored to its original state, relieving the user of this task. For complex hypothetical queries, this can significantly reduce the user's programming burden as well as ensure that the database is reliably restored. In addition, because the queries are declarative, they can be used in the premises of rules. In this way, we extend the language of hypothetical queries to a logic for building rulebases.

The development in this paper is for a language without function symbols. This is the kind of language typically used in database applications. Although it is function-free, we allow the language to have an *infinite* set of constant symbols. It should therefore be possible to extend our development to include function symbols, by treating each Herbrand term as a distinct constant symbol. Moreover, by focussing on the function-free case, we see that hypothetical queries can increase the power of a logic. For instance, function-free Horn logic (also known as Datalog) can only solve problems in P. We give simple examples showing that Hypothetical Datalog can find Hamiltonian paths in a graph, and can determine whether combinational circuits are valid, problems that cannot be solved by Datalog, unless P=NP. We give other examples illustrating the subtle effect that deletion has on expressive power. We show, for instance, that in a top-down proof procedure, recursion can go to exponential depth before goals begin to repeat (exponential in the number of constant symbols). In contrast, with insertion alone, recursion can go only to polynomial depth. This provides an intuitive explanation of why inference with insertion alone is PSPACE-complete, while with both insertion and deletion, it is EXPTIME-complete [10, 11].

Like classical and intuitionistic logics, Hypothetical Datalog is *monotonic*. Thus, increasing the set of base facts can only increase (not decrease) the set of inferred facts. However, because it is monotonic, it cannot express *non*-monotonic queries. Thus, one cannot ask for the difference of two database relations. To express such queries, we augment the logic with a well-known non-monotonic operator: *negation-as-failure*. This is a natural extension, one that is just as useful for hypothetical rules as for Horn rules, as we show through numerous examples. Furthermore, as shown in [10, 11], Hypothetical Datalog with negation-as-failure is expressively complete for EXPTIME. That is, it expresses *all* database queries in EXPTIME, both monotonic and non-monotonic.

There has been considerable work in recent years on the semantics of negation-as-failure. Most of this work has focussed on Horn rulebases and the myriad problems created by recursion through negation [55, 35, 34, 64]. More recently, some researchers have focussed on the semantics of *intuitionistic* rulebases with negation [38, 52, 36, 26]. These works all focus on *arbitrary* (*i.e.*, non-stratified) rulebases. Unfortunately, none of these semantics is suited to our needs, for several reasons. Foremost, intuitionistic logic models hypothetical *insertion*, not deletion. Moreover, most of these works focus on *finite* failure, whereas our logic uses *infinite* failure. Some of these works do not offer a proof theory, while others impose syntactic restrictions that exclude many stratified rulebases. We discuss these works in greater detail in Section 7. The problems encountered all reflect the difficulty of dealing with recursion through negation.

Fortunately, none of our complexity and expressibility results depend on recursion through negation. For instance, in [10, 9, 7, 18, 12, 11], we show that *stratified* hypothetical rules are expressively complete for many well-known complexity classes. In [9, 7], we show that each stratum can raise the complexity of inference by one level in the polynomial time hierarchy. In addition to these complexity-theoretic properties, it is worth pointing out that stratified hypothetical rulebases act as a kind of benchmark in the semantics of negation (just as stratified Horn rulebases do). One reason is that most natural examples of negation are stratified. Indeed, this paper provides many such examples. Another reason is that the semantics of non-stratified hypothetical rulebases is likely to remain unsettled for some time (just as in Horn logic). Moreover, without a clear semantics for the stratified case, it is hard to judge whether a non-stratified semantics is correct. For all these reasons, this paper focuses on stratified hypothetical rulebases.

To augment our logic with negation-as-failure, we first generalize the notion of stratification from Horn rulebases to hypothetical rulebases. We then develop the proof theory and model theory for the augmented logic. The proof theory is inspired by the stratified semantics of Apt, Blair and Walker [4], and the model theory is inspired by the perfect model semantics of Przymusinski [55]. As is the classical case, we show that every stratified hypothetical rulebase has a single perfect model. Nothing in the development depends on any finiteness assumptions, so all quantities may be infinite: the database, the rulebase, the set of constant symbols, and even the number of strata.

2. EXPRESSING HYPOTHETICAL QUERIES

This section addresses the question of what it means to express a hypothetical query. We begin with several informal examples of hypothetical queries and rules. We then show that hypothetical queries lead naturally to new notions of query and expressibility. In particular, we generalize the notion of *database* query to that of *rulebase* query. The generalization is motivated by a principle that we call *rulebase independence*. Intuitively, a user should be able to formulate queries to a rulebase without having a detailed knowledge of its contents. Similar requirements exist in database systems, since a user must be able to formulate queries to a database without knowing its exact contents.

Using this idea, we show that classical logic is inadequate for hypothetical reasoning. In particular, we show a simple hypothetical query that classical logic cannot express in a rulebase independent way. This result explains why much of the work on hypothetical reasoning uses intuitionistic logic instead of classical logic. Intuitionistic logic expresses many hypothetical queries in a rulebase independent way. However, it expresses only hypothetical *insertions*. We show that if hypothetical deletion is allowed, then there are some simple hypothetical queries that intuitionistic logic cannot express in a rulebase independent way. In fact, we show that these queries cannot be expressed by any reasonable first-order logic. Thus, if hypothetical queries are to be expressed declaratively, then new logical mechanisms are needed.

For this reason, we introduce a modal-like notation for expressing hypothetical queries. In particular, $Q[add : P]$ means that *Q would be true if P were added to the database*, and $Q[del : P]$ means that *Q would be true if P were deleted from the database*. The notation $R, DB \vdash \psi$ means that formula ψ can be inferred from rulebase R and database DB. In this paper, R is a set of logical rules, and DB is a set of ground atomic formulas. Examples illustrating this notation are given in [10]. For convenience, we present a selection of these examples below.[1] Additional examples can be found in [11]. Our first set of examples involve hypothetical insertion only. This is enough to introduce the idea of rulebase independence. We shall consider hypothetical deletion shortly.

2.1. Introductory Examples

The examples in this section are based on a deductive database for a university. In this system, the database contains information about what courses the students have taken, and the rulebase infers which students are eligible to graduate. Facts are represented as ground atomic formulas. For instance, the formula $course(c)$ means that c is a course, the formula $take(s, c)$ means that student s has taken course c, and the formula $grad(s)$ means that student s is eligible to graduate. The database thus contains facts such as $take(thorne, csc250)$, and the rulebase contains rules such as

$$grad(S) \leftarrow take(S, his101) \wedge take(S, eng201).$$

This rule means that a student, S, can graduate by taking History 101 and English 201. The expression $R, DB \vdash grad(s)$ means that we can infer that student s is eligible to graduate. We use the Prolog convention that logical variables begin in upper case, while constant symbols begin in lower case.

Example 2.1. [Hypothetical Insertion] Consider the following query: *Could Thorne graduate if he took csc452?* In other words, if $take(thorne, csc452)$ were added to the database, could we infer $grad(thorne)$? This query can be formalized at the meta-level as follows:[2]

$$R, DB + take(thorne, csc452) \;\vdash\; grad(thorne) \tag{2.1}$$

The formula $grad(thorne)[add : take(thorne, csc452)]$ expresses this query in Hypothetical Datalog. That is $R, DB \vdash grad(thorne)[add : take(thorne, csc452)]$ if and only if condition (2.1) is satisfied.

[1] The notation used in the examples of this section is slightly different from the notation developed in Section 3. In particular, various quantifiers and connectives, which are implicit in Hypothetical Datalog, are made explicit in this section, to aid understandability.

[2] In this paper, the symbol "+" denotes set union. Also, if A is not a set, then $DB + A$ is used as an abbreviation for $DB + \{A\}$.

Example 2.2. [Non-Determinism] Consider the following query: *Retrieve those students who could graduate if they took (at most) one more course.* At the meta-level, we want those students s such that for some course c,

$$R, DB + take(s,c) \ \vdash \ grad(s) \tag{2.2}$$

The formula $\psi(S) \ = \ \exists C \ course(C) \wedge grad(S)[add : take(S,C)]$ expresses this query. That is, condition (2.2) is satisfied if and only if $R, DB \vdash \psi(s)$. Observe that the hypothetical insertion is non-deterministic, since it depends on the non-deterministic choice of a value for C, as reflected by the existential quantifier.

Example 2.3. [Hypothetical Rules] Suppose the university has the following 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 is expressed by the following two rules:

$$within1(S,D) \ \leftarrow \ \exists C \ course(C) \wedge grad(S,D)[add : take(S,C)].$$
$$grad(S, math\&phy) \ \leftarrow \ within1(S, math) \wedge within1(S, phy).$$

where the predicate $grad(s,d)$ 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 discipline d. Note that the premise of the first rule is a hypothetical query similar to the query in Example 2.2.

2.2. Rulebase Queries

A database query is often treated as a mapping that takes a relational database as input and returns a relation as output [20, 19]. The expressibility of a query language is therefore the set of mappings that the language represents. Many expressibility results based on this idea have appeared in the literature [3, 21, 22, 40, 2, 1, 39], and we have established numerous results of this kind using Hypothetical Datalog as a query language [10, 9, 7, 18, 11, 12]. However, hypothetical queries lead naturally to a generalized notion of expressibility, one that is appropriate for deductive databases and rule-based systems. In this notion, a query is still a mapping from databases to relations, but now the input to the mapping is a *deductive* database, not just a *relational* database.

The motivation for this generalized notion of expressibility is a principle we call *rulebase independence.* Specifically, *a user should be able to formulate queries to a deductive database without knowing the contents of the rulebase*, that is, without understanding the knowledge encoded in the rules. For instance, if a deductive database encodes the U.S.-Canada Income Tax Treaty, then a user who is not an expert on the treaty should still be able to pose questions to the system, such as, *Is my pension income from Canada taxable in the U.S.?* Similar constraints exist in relational database systems, since a user must be able to formulate database queries without knowing the contents of the database.

Rulebase independence is important for several reasons. First, it permits casual use of a deductive database system: one does not require expert knowledge of the rulebase in order to formulate queries. Second, it increases reliability: without rulebase independence, if a user's knowledge of the rulebase is inaccurate, then his queries may be formulated incorrectly. Third, rulebase independence increases flexibility by making changes to the rulebase less expensive: without rulebase independence, application programs that query the rulebase, and libraries of commonly used queries, would have to be updated each time the rulebase is changed. Finally, rulebase independence is important not only for the formulation of queries but also for the construction of rulebases. Since the premise of a rule is itself a rulebase query, keeping rule premises independent of the rest of the rulebase has all the advantages just listed for queries: it makes rules easier to formulate and more reliable, and it prevents small changes from propagating throughout the whole system.

Before formalizing these ideas, we illustrate the principle of rulebase independence by a negative example, that is, by a query expression that is rulebase *dependent.*

Example 2.4. [Rulebase Dependence] Consider the rulebase R_1 consisting of just the following Horn rule:

$$grad(S) \leftarrow take(S, his101) \wedge take(S, eng201).$$

This rule says that a student, S, can graduate by taking History 101 and English 201. Now, suppose we wish to ask the following query: *Retrieve those students who are within (at most) one course of graduation.* Knowing the contents of R_1, we could express this query with the formula $\psi(S) = take(S, eng101) \vee take(S, his101)$. That is, students who have taken English 201 or History 101 are within one course of graduation. Since the formula $\psi(S)$ works for all databases, it expresses the query in a *database-independent* way. Intuitively, it assumes nothing about the contents of the database of students and courses. Thus, when the database is updated, $\psi(S)$ will continue to give correct answers. However, $\psi(S)$ does not express the query in a *rulebase independent* way, since it does not work for all rulebases. For example, it does not work for the rulebase R_2 consisting of just the following Horn rule:

$$grad(S) \leftarrow take(S, his101) \wedge take(S, eng201) \wedge take(S, math250).$$

To make the above ideas precise, we define what a query is and what it means to express a query in logic. For relational databases, these ideas are well understood, and we simply review the main concepts. We then extended the concepts to deductive databases. To keep the presentation simple, we restrict the development to classical Horn logic, but the ideas can be extended to other logical systems (such as intuitionistic logic and Hypothetical Datalog) in a natural way.

Database Queries. To express queries to a relational database, we distinguish two disjoint sorts of predicate: base predicates and query predicates. For the purpose of this section, a relational database is any set of ground atomic formulas constructed entirely from base predicate symbols. Query predicates are reserved for constructing logical formulas that query the database.

As a simple example, suppose we wish to find the transitive closure of a binary predicate p stored in the database. This query is expressed by the following rulebase, R_0, consisting of two rules:

$$tc(X,Y) \leftarrow p(X,Y) \qquad\qquad tc(X,Y) \leftarrow p(X,Z),\ tc(Z,Y).$$

where tc is a query predicate. That is, for any relational database DB, and any pair of constants a, b,

$$R_0 + DB \vdash^c tc(a,b) \quad \textit{iff} \quad (a,b) \textit{ is in the transitive closure of } p \tag{2.3}$$

where $\vdash^c$ denotes entailment in first-order classical logic. It is important that tc is a query predicate, since otherwise some databases would contain atoms of the form $tc(a,b)$, so statement (2.3) would not be true.

To generalize from this example, we need a precise definition of database query. Following Chandra and Harel [20], we define a *database query* to be a mapping from relational databases to relations.[3]Thus, if $\mathcal{Q}$ is a database query, and DB is a relational database, then $\mathcal{Q}(DB)$ is a relation.

Definition 2.1. [Expressing Database Queries] Let R_0 be a set of rules, and let $\psi(X_1, ..., X_n)$ be a logical formula with free variables $X_1, ..., X_n$ (in some order). Then, R_0 and ψ *express* the database query $\mathcal{Q}$ where

$$(c_1, ..., c_n) \in \mathcal{Q}(DB) \qquad \textit{iff} \qquad R_0 + DB \vdash^c \psi(c_1, ..., c_n)$$

for every relational database DB, and every n-tuple of constant symbols $(c_1, ..., c_n)$.

Observe that R_0 and ψ may be constructed from any sort of predicate symbol. Also observe that in some logical systems, it is not strictly necessary to distinguish between R_0 and ψ. For instance, in classical logic, R_0 can be absorbed into ψ, by the deduction theorem; that is,

$$R_0 + DB \vdash^c \psi \quad \textit{iff} \quad DB \vdash^c \psi \leftarrow R_0$$

However, in deductive databases and logic programming, it is traditional to *not* do this. Moreover, once negation-as-failure is introduced into a logical system, the deduction theorem is often invalid, so it becomes necessary to distinguish R_0 from ψ.

Rulebase Queries. It is not hard to generalize the above ideas from relational to deductive databases. We first distinguish *three* disjoint sorts of predicates: base predicates, derived predicates, and query predicates. For the purpose of this section, we define a *deductive database* to be a pair $[R, DB]$ where DB is a set of ground atomic formulas constructed from base predicates, and R is a set of Horn rules constructed from base and derived predicates. Query predicates are reserved for constructing logical formulas that query the deductive database.

We define a *rulebase query* to be a mapping from deductive databases to relations. Thus, if $\mathcal{Q}$ is a rulebase query, and $[R, DB]$ is a deductive database, then $\mathcal{Q}(R, DB)$ is a relation.[4]For example, R could be a collection of rules defining the

[3] In the definition of [20], the mapping must satisfy certain conditions, such as computability and genericity. However, these conditions are not relevant to the present discussion.

[4] As with database queries, we might want to impose conditions on the mapping $\mathcal{Q}$. For instance, we could require that $\mathcal{Q}(R_1, DB) = \mathcal{Q}(R_2, DB)$ if R_1 and R_2 are logically equivalent. Of course, the notion of equivalence depends on what logical system we are using. However, these

U.S.-Canada Income Tax Treaty, DB could be a collection of facts about individual incomes, and a query could be, *Is Thorne's U.S. pension income taxable in Canada?* Of course, the answer to the query depends both on R and on DB, both of which may change over time. We would like to express the query in a rulebase-independent manner, so that it is insensitive to these changes.

Definition 2.2. [Expressing Rulebase Queries] Let R_0 be a set of rules, and let $\psi(X_1, ..., X_n)$ be a logical formula with free variables $X_1, ..., X_n$ (in some order). Then, R_0 and ψ *express* the rulebase query $\mathcal{Q}$ where

$$(c_1, ..., c_n) \in \mathcal{Q}(R, DB) \quad \textit{iff} \quad R_0 + R + DB \vdash^c \psi(c_1, ..., c_n)$$

for every deductive database $[R, DB]$, and every n-tuple of constants $(c_1, ..., c_n)$.

Observe that R_0 and ψ may be constructed from any sort of predicate symbol. Again, in some logical systems, it may be possible to absorb R_0 into ψ, by a deduction theorem.

2.3. Examples: Classical and Intuitionistic Logic

This section gives examples of rulebase queries involving hypothetical insertion. The examples illustrate the definitions above as well as the use of classical and intuitionistic logic as query languages. The examples are similar to those in Section 2.1, and are based on a deductive database for a university. The database DB thus contains facts about courses, and the rulebase R contains Horn rules about when students can graduate.

In the examples, $\mathcal{Q}_{grad}$ is a rulebase query that returns a unary relation. In particular,

$$(s) \in \mathcal{Q}_{grad}(R, DB) \quad \textit{iff} \quad R + DB \vdash^c grad(s)$$

for any constant s, and any deductive database $[R, DB]$. Intuitively, $\mathcal{Q}_{grad}(R, DB)$ is the set of students who are eligible to graduate according to the rules in R and the facts in DB. In these examples, rulebase queries will be expressed using the empty rulebase $R_0 = \{\}$, and a formula $\psi(S)$ with one free variable, S. For instance, the query $\mathcal{Q}_{grad}$ is expressed by the formula $\psi(S) = grad(S)$.

Example 2.5. [Hypothetical Insertion] We would like to express the following rulebase query: *Retrieve those students who could graduate if they took* $csc450$. Formally, we want to express the query $\mathcal{Q}^1_{hypo}$, where

$$(s) \in \mathcal{Q}^1_{hypo}(R, DB) \quad \textit{iff} \quad (s) \in \mathcal{Q}_{grad}[R,\ DB + take(s, csc450)]$$

and $\mathcal{Q}_{grad}$ is defined above. In both classical and intuitionistic logic, $\mathcal{Q}^1_{hypo}$ is expressed by the formula $\psi(S) = grad(S) \leftarrow take(S, csc450)$. To see this, observe that for any constant symbol s,

considerations are not relevant to the present discussion.

$$\begin{array}{lll} & R + DB \vdash \psi(s) & \\ \textit{iff} & R + DB \vdash grad(s) \leftarrow take(s, csc450) & \\ \textit{iff} & R + DB + take(s, csc450) \vdash grad(s) & \text{by the deduction theorem,} \\ \textit{iff} & (s) \in \mathcal{Q}_{grad}[R, DB + take(s, csc450)] & \text{by the definition of } \mathcal{Q}_{grad}, \\ \textit{iff} & (s) \in \mathcal{Q}^1_{hypo}(R, DB) & \text{by the definition of } \mathcal{Q}^1_{hypo}. \end{array}$$

Example 2.5 shows that classical logic can express some simple hypothetical queries in a rulebase independent way. However, classical logic is poor at expressing more-complex hypothetical queries, as we shall see. To express hypothetical insertions, intuitionistic logic is much more suitable [12, 18, 7, 11]. From a logic-programming standpoint, intuitionistic logic can be viewed as an extension of classical logic. It uses the same syntax as classical logic, and is equivalent to classical logic for Horn rulebases and atomic queries. That is, if $\vdash^i$ denotes entailment in first-order intuitionistic logic, then

$$R \vdash^i A \qquad \textit{iff} \qquad R \vdash^c A \tag{2.4}$$

for any Horn rulebase R, and any ground atomic formula A. However, intuitionistic and classical logic have different semantics and different theorems [29]. For example, in intuitionistic logic, implication is not defined in terms of disjunction and negation, but has an independent semantic definition. Thus, the formula $\alpha \leftarrow \beta$ is *not* equivalent to $\alpha \vee \sim \beta$ intuitionistically. One consequence is that intuitionistic implication has the following property:

$$R \vdash^i \exists X\ \rho(X) \qquad \textit{iff} \qquad R \vdash^i \rho(x) \quad \textit{for some constant } x \tag{2.5}$$

for any Horn rulebase R, and any Horn rule $\rho(X)$ with free variable X. Classical logic does not enjoy this property.[5] This difference contributes to making intuitionistic logic a better logic for expressing hypothetical insertions, as the next two examples suggest.

Example 2.6. [Intuitionistic Logic] We would like to express the following rulebase query: *Retrieve those students who could graduate if they took one more course.* Formally, we want to express the query $\mathcal{Q}^2_{hypo}$ where

$$\begin{array}{ll} & (s) \in \mathcal{Q}^2_{hypo}(R, DB) \\ \textit{iff} & (s) \in \mathcal{Q}_{grad}[R,\ DB + take(s, c)] \quad \textit{for some constant } c \end{array}$$

and $\mathcal{Q}_{grad}$ is defined above. In intuitionistic logic, $\mathcal{Q}^2_{hypo}$ can be expressed by the formula $\psi(S) = \exists C\ grad(S) \leftarrow take(S, C)$. To see this, observe that for any constant s,

$$\begin{array}{lll} & R + DB \vdash^i \psi(s) & \\ \textit{iff} & R + DB \vdash^i \exists C\ [grad(s) \leftarrow take(s, C)] & \\ \textit{iff} & R + DB \vdash^i grad(s) \leftarrow take(s, c) & \text{for some c, by property (2.5),} \\ \textit{iff} & R + DB + take(s, c) \vdash^i grad(s) & \text{by the deduction theorem,} \\ \textit{iff} & R + DB + take(s, c) \vdash^c grad(s) & \text{by property (2.4),} \\ \textit{iff} & (s) \in \mathcal{Q}_{grad}[R,\ DB + take(s, c)] & \text{by the definition of } \mathcal{Q}_{grad}, \\ \textit{iff} & (s) \in \mathcal{Q}^2_{hypo}(R, DB) & \text{by the definition of } \mathcal{Q}^2_{hypo}. \end{array}$$

[5] Actually, both classical and intuitionistic logics enjoy this property when $\rho(X)$ is a positive existential formula; but in intuitionistic logic, implications are positive formulas, while in classical logic, they are not (since classical implication is defined in terms of negation).

Example 2.7. [Classical Logic] The formula $\psi(S)$ used in Example 2.6 expresses a completely different query in classical logic. This is because of the following classical equivalence:

$$\exists C\ [grad(s) \leftarrow take(s,C)] \quad \equiv \quad grad(s) \leftarrow \forall C\ take(s,C) \tag{2.6}$$

That is, because $grad(s)$ is independent of C, the existential quantifier can be brought inside the implication sign and converted to a universal, in the usual way. Consequently, in classical logic, $\psi(S)$ expresses the query, *Retrieve those students who could graduate if they took* <u>*every*</u> *course?* Formally, for any constant s,

$$\begin{array}{lll} & R + DB \vdash^c \exists C\ [grad(s) \leftarrow take(s,C)] & \\ \textit{iff} & R + DB \vdash^c grad(s) \leftarrow \forall C\ take(s,C) & \text{by property (2.6),} \\ \textit{iff} & R + DB + \forall C\ take(s,C) \vdash^c grad(s) & \text{by the deduction theorem.} \end{array}$$

In the last line, we have effectively augmented the rulebase with a rule stating that student s has taken every course. This is true for every student, every database, and every rulebase. Thus, for rulebases R_1 and R_2 of Example 2.4, the formula $\exists C\ grad(s) \leftarrow take(s,C)$ is true for *every* student, even for students who have taken no courses at all! Of course, retrieving those students who "could graduate if they took every course" is a legitimate hypothetical query. In both classical and intuitionistic logics, this query is expressed by the formula $\psi'(S) = grad(S) \leftarrow \forall C\ take(S,C)$.

Examples 2.6 and 2.7 illustrate that intuitionistic logic makes distinctions that classical logic does not. For instance, intuitionistic logic distinguishes between the formulas $\exists C\ grad(s) \leftarrow take(s,C)$ and $grad(s) \leftarrow \forall C\ take(s,C)$, while classical logic does not. That is, in classical logic, these formulas are equivalent, so they express the same query; while in intuitionistic logic, they are not equivalent, so they express different queries. In general, classical logic has more equivalences than intuitionistic logic, since every theorem of intuitionistic logic is also a theorem of classical logic, but not vice versa [29]. Intuitively, this means that intuitionistic logic makes finer distinctions between formulas. This seems to be why intuitionistic logic is better than classical logic at expressing hypothetical queries.

2.4. Limitations of Classical Logic

As seen in Example 2.5, classical logic can express some simple hypothetical queries in a rulebase independent way. However, classical logic is not good at expressing more-complex hypothetical queries, even if they are just slightly more complex. This section gives an example of such a query, and proves that it cannot be expressed in classical logic.[6]

To make the example concrete, consider a deductive database for a university, as described in Section 2.1. We are interested in a promising young student, Tomasz, who has taken many courses in mathematics and in physics. We suspect that Tomasz might qualify for a physics degree by taking the senior physics course,

[6] A summary of this material appears in [8].

*phy*450, or he might qualify for a math degree by taking the senior math course, *math*452. We want to know if Tomasz is in one of these two situations. That is, we want to ask the query, *Could Tomasz graduate by taking phy*450 *or by taking math*452*?* We show that this rulebase query cannot be expressed in classical logic.

To express this query formally, let G denote the proposition that Tomasz can graduate, let A denote the proposition that he has taken *phy*450, and let B denote the proposition that he has taken *math*452. We therefore want a rulebase R_0 and a logical formula ψ such that for any database DB, and any Horn rulebase R,

$$\begin{array}{ll} & R_0 + R + DB \ \vdash^c \ \psi \\ \textit{iff} & R + DB + A \ \vdash^c \ G \quad \textit{or} \quad R + DB + B \ \vdash^c \ G \end{array} \tag{2.7}$$

Without loss of generality, we can assume that $R_0 = \{\}$, since R_0 can be absorbed into ψ, by the deduction theorem. We therefore need only look for ψ. We show that in classical logic, no such ψ exists. First, however, we consider two promising candidates, and show why they do not work.

The first candidate for ψ is the formula $G \leftarrow (A \vee B)$, which we might read as, "*G is true if A is true or B is true.*" This formula does not work because of the following classical equivalence:

$$G \leftarrow (A \vee B) \quad \equiv \quad (G \leftarrow A) \wedge (G \leftarrow B)$$

Hence,

$$\begin{array}{ll} & R + DB \ \vdash^c \ \psi \\ \textit{iff} & R + DB + A \ \vdash^c \ G \quad \textit{and} \quad R + DB + B \ \vdash^c \ G. \end{array} \tag{2.8}$$

But this is not what we had in mind. For instance, if we choose R and DB so that $R + DB + A \vdash^c G$ and $R + DB + B \not\vdash^c G$, then $R + DB \not\vdash^c \psi$ by condition (2.8), which violates condition (2.7).

Another promising candidate for ψ is the formula $(G \leftarrow A) \vee (G \leftarrow B)$, which we might read as, "*G is true if A is true, or G is true if B is true.*" In intuitionistic logic, this formula *does* express query (2.7). However, in classical logic, this formula does not work. This is because of the following classical equivalences, which are analogous to equivalence (2.6):

$$\begin{aligned} (G \leftarrow A) \vee (G \leftarrow B) &\equiv (G \vee \sim A) \vee (G \vee \sim B) \\ &\equiv G \vee \sim A \vee \sim B \\ &\equiv G \leftarrow A \wedge B \end{aligned} \tag{2.9}$$

Thus, $R + DB \ \vdash^c \ \psi$ if and only if $R + DB + A + B \ \vdash^c \ G$. Again, this is not what we had in mind.

We have thus eliminated two promising candidates for ψ. Neither candidate expresses query (2.7). Is there some other formula that does? Lemma 2.1 below assures us that there is not. In this lemma, deductive databases may be restricted to consist of Horn rules, or they may contain arbitrary first-order formulas. The lemma is true in both cases. The lemma is also true if deductive databases are restricted to be non-recursive.

Lemma 2.1. Let A, B and G be distinct ground atomic formulas made from base or derived predicate symbols. Then, in classical first-order logic, there is no formula ψ such that for any deductive database $[R, DB]$,

$$R + DB \ \vdash^c \ \psi \quad \textit{iff} \quad R + DB + A \ \vdash^c \ G \quad \textit{or} \quad R + DB + B \ \vdash^c \ G$$

Proof. [By contradiction.] Suppose that such a ψ exists. We show that when the database is empty, ψ cannot give correct answers for all three of the following rulebases:

$$R_1 = \{G \leftarrow A\} \qquad R_2 = \{G \leftarrow B\} \qquad R_3 = \{G \leftarrow A \wedge B\}$$

In particular, we show that if ψ returns correct answers for R_1 and R_2, then it must return incorrect answers for R_3. The proof exploits the equivalences of statement (2.9) above, *i.e.*, $R_1 \vee R_2 \equiv R_3$.

$$\begin{array}{ll}
\text{Clearly} & R_1 + A \vdash^c G \quad \text{and} \quad R_2 + B \vdash^c G \\
\text{thus} & R_1 \vdash^c \psi \quad \text{and} \quad R_2 \vdash^c \psi \quad \text{by the definition of } \psi, \\
\text{so} & (R_1 \vee R_2) \vdash^c \psi \\
 & R_3 \vdash^c \psi \quad \text{since } R_1 \vee R_2 \equiv R_3, \\
 & R_3 + A \vdash^c G \quad \text{or} \quad R_3 + B \vdash^c G \quad \text{by the definition of } \psi, \\
 & \{G \leftarrow A \wedge B\} + A \vdash^c G \quad \text{or} \quad \{G \leftarrow A \wedge B\} + B \vdash^c G \\
 & \text{which is a contradiction.}
\end{array}$$

□

Thus, query (2.7) *cannot* be expressed in classical logic. However, it *can* be expressed in intuitionistic logic by the formula $\psi = (G \leftarrow A) \vee (G \leftarrow B)$ as mentioned above. In Section 3.3, we shall see that it can also be expressed in Hypothetical Datalog in a straightforward way.

2.5. Hypothetical Deletion

So far, we have only looked at hypothetical insertion. This was enough to show that classical logic is inadequate for hypothetical reasoning. As mentioned above, intuitionistic logic is the correct way to formalize hypothetical insertion [32, 47, 50, 8, 18, 12]. However, even intuitionistic logic does not model hypothetical *deletion*. In fact, we shall see in this subsection that no reasonable first-order logic can do this. We then consider some alternatives. First, however, we give some informal examples of queries involving hypothetical deletion, and show how they are expressed in Hypothetical Datalog.

Example 2.8. [Hypothetical Deletion] Consider the following query: *If Tomasz had not taken History* 250, *could he have graduated?* In Hypothetical Datalog, this query is expressed by the formula $grad(tomasz)[del : take(tomasz, his250)]$, which has the following property:[7]

$$\begin{array}{rl}
 & R, DB \vdash grad(tomasz)[del : take(tomasz, his250)] \\
\textit{iff} & R, DB - take(tomasz, his250) \vdash grad(tomasz)
\end{array}$$

for any rulebase R, and any database DB.

Example 2.9. [Non-Determinism] Consider the following query: *Retrieve those students who took one more course than necessary to graduate.* That is, we want

[7] In this paper, the symbol "$-$" denotes set difference. Also, if A is not a set, then $DB - A$ is used as an abbreviation for $DB - \{A\}$.

those students who could have graduated by taking one less course. Formally, we want those students s such that for some course c,

$$\begin{aligned} & R, DB \vdash take(s,c) \\ and \quad & R, DB - take(s,c) \vdash grad(s) \end{aligned} \tag{2.10}$$

The formula $\psi(S) = \exists C \; take(S,C) \wedge grad(S)[del : take(S,C)]$ expresses this query. That is, condition (2.10) is satisfied if and only if $R, DB \vdash \psi(s)$. Observe that the hypothetical deletion is non-deterministic, since it depends on the non-deterministic choice of a value for C, as reflected by the existential quantifier.

When hypothetical deletion is allowed, even very simple rulebase queries cannot be expressed in most logics, including classical and intuitionistic logic. To see this, consider the query, *Could $A(a)$ be inferred if $B(a)$ were deleted from the database?* Suppose this query can be expressed by some formula ψ. Then ψ has the following property:

$$R + DB \vdash \psi \quad \textit{iff} \quad R + [DB - B(a)] \vdash A(a) \tag{2.11}$$

for any deductive database $[R, DB]$. This property leads immediately to a contradiction. To see this, consider the following two rulebases:

$$R_1 = \{ \; A(x) \leftarrow B(x) \; \} \qquad\qquad R_2 = \left\{ \begin{array}{l} A(x) \leftarrow B(x) \\ A(a) \end{array} \right\}$$

If $DB = \{B(a)\}$, then $R_1 + DB$ is logically equivalent to $R_2 + DB$. This is true both in classical logic and in intuitionistic logic. In fact, it is true in any logic obeying the laws of modus ponens and variable instantiation, *i.e.*, in any reasonable logic. Thus $R_1 + DB$ and $R_2 + DB$ are logically indistinguishable. Hence, it should be the case that $R_1 + DB \vdash \psi$ if and only if $R_2 + DB \vdash \psi$. But this is a contradiction, since according to condition (2.11), $R_1 + DB \not\vdash \psi$ and $R_2 + DB \vdash \psi$. This contradiction shows that even the simplest queries involving hypothetical deletion cannot be expressed in a normal, first-order framework.

Simulating Deletion. To a limited extent, hypothetical deletion can be simulated by a simple methodology involving hypothetical insertion and negation-as-failure. (Negation-as-failure is then the device that takes us beyond first-order.) In this approach, for every predicate, B, we create two new predicates, $insB$ and $delB$. Intuitively, $insB$ means that B has been inserted into the database, and $delB$ means that B has been deleted from the database. In this methodology, instead of inserting B, we insert $insB$, and instead of deleting B, we insert $delB$. The state of B is then inferred by the rule $B \leftarrow insB, \sim delB$. Intuitively, this rule says that B is true if it has been inserted and has not yet been deleted. This idea can be used to good effect in many applications, such as Example 5.5 in Section 5. However, an obvious problem with this methodology is that once B has been deleted, it cannot be re-inserted, since once $delB$ is in the database, it cannot be removed. More sophisticated methodologies are possible, but it is extremely unlikely that any such approach can simulate deletion completely, simply because of a complexity mismatch. In [10, 11], we show that the data complexity of hypothetical deletion is complete for EXPTIME, and that for insertion with negation-as-failure, it is in PSPACE. Thus, insertion plus negation cannot simulate deletion, unless PSPACE=EXPTIME. The reason for this difference in data complexity is illustrated in Section 3.5, where we show that with hypothetical deletion, recursion can go to exponential depth without repeating any goals; whereas with insertion, it can only go to polynomial depth. Besides these differences in complexity and recursion depth, using negation-as-failure to model hypothetical deletion would mean introducing all the difficulties of negation into the semantics. Recursion through deletion would then require recursion through negation, which is especially problematic. Finally, the use of negation-as-failure would give deletion a non-monotonic semantics.

Hypothetical Datalog. This paper takes a different approach by developing a simple monotonic logic for both hypothetical insertion and deletion. Instead of merging the two components of a deductive database, R and DB, into a single set, $R + DB$, Hypothetical Datalog maintains them as separate logical entities. The logic can therefore distinguish between stored and derived data. Update operators act only on the stored data. That is, $[add : C]$ adds the atom C to the stored data, and $[del : C]$ deletes C from the stored data. Since derived data is never updated directly, we avoid the difficult view-update problem [27, 28, 25, 6, 31, 43].

Like classical and intuitionistic logics, Hypothetical Datalog does *not* distinguish between different sorts of predicates, such as base and derived predicates. Of course, predicates can always be divided into sorts, as an option, but it is not imposed by the theory. Consequently, the extent of a predicate may consist of both stored and derived data. For example, if $DB = \{A(a), B(b)\}$ and R contains the rule $A(x) \leftarrow B(x)$, then $A(a)$ is stored and $A(b)$ is derived. In fact, a single atom may be *both* stored and derived. For instance, if $DB = \{A(b), B(b)\}$ and R contains the rule $A(x) \leftarrow B(x)$, then the atom $A(b)$ is stored and derived. For this reason, it is possible to delete a stored fact, and then rederive it. For instance, in the last example, if we delete $A(b)$ from the stored data, then it will be rederived from the remaining data. That is, the statement $R, DB \vdash A(b)[del : A(b)]$ is true in this case. Of course, if we delete *both* $A(b)$ and $B(b)$ from the stored data, then $A(b)$ is no longer derivable, so $R, DB \not\vdash A(b)[del : A(b), B(b)]$.

Although deleted data can be rederived in principle, it may never happen in practice. This is because deductive database systems typically impose restrictions on how predicates can be used. For instance, it is common to distinguish between base and derived predicates, where only base predicates can be updated. Moreover, the database contains only base predicates, and rule heads mention only derived predicates. With such restrictions, the extent of each predicate is either stored or derived, and deleted data is not rederived. We emphasize, however, that such restrictions are a matter of practice, and none of the theory developed in this paper depends on them.

Since we do not deal with the view-update problem, the updates considered in this paper are relatively simple. Nevertheless, they have an enormous effect on the computational power of the logic, and on its ability to express queries. For instance, without any updates at all, Hypothetical Datalog reduces to classical Datalog (function-free Horn logic), so it can only express database queries computable in polynomial time. However, with hypothetical insertion, it expresses all the database queries in PSPACE, and with both insertion and deletion, it expresses all the database queries in EXPTIME [10, 11].

3. SYNTAX AND PROOF THEORY

Hypothetical Datalog was introduced in [10]. This section reviews the syntax and proof theory of the logic, and presents new examples illustrating its expressive power. The rest of the paper then develops the model theory and fixpoint theory of the logic, and extends it with negation-as-failure.

The inference system is an extension of Horn logic, both syntactically and proof-theoretically. It is a minimal extension in that it augments the proof theory of Horn logic with just two rules of inference: one for hypothetical insertion, and one for hypothetical deletion. In this inference system, all logical expressions are function-free. We focus on the function-free case because of its importance to database systems; however, it should not be difficult to extend the system to include function symbols. Several simple examples illustrate hypothetical inference, and two extended examples illustrate its computational power. The first extended example shows that insertion alone can solve problems that classical Datalog cannot, unless P=NP. The second extended example illustrates why insertion and deletion together are more powerful that insertion alone.

3.1. Hypothetical Inference

The language of Hypothetical Datalog includes three (finite or infinite) enumerable sets: variables, $X, Y, Z, ...$, constant symbols, $a, b, c, ...$, and predicate symbols, $P, Q, R, ...$. Hypothetical queries and rules are constructed from these symbols. A database is any set of ground atomic formulas constructed from these symbols. As in the previous section, we use the Prolog convention that variables begin in upper case, while constant symbols begin in lower case.

Definition 3.1. [Hypothetical Queries] A hypothetical query is a formula having one of the following forms:

- A, where A is an atomic formula.
- $A[add : B_1, ..., B_k]$, where $A, B_1, ..., B_k$ are atomic formulas.
- $A[del : B_1, ..., B_k]$, where $A, B_1, ..., B_k$ are atomic formulas.

Definition 3.2. [Hypothetical Rules] A hypothetical rule is a formula of the form $A \leftarrow \phi_1, \phi_2, ..., \phi_k$ where $k \geq 0$, A is an atomic formula, and each ϕ_i is a hypothetical query.

A set of hypothetical rules is called a *hypothetical rulebase*, or simply a *rulebase* for short. Note that as a special case, Horn rules are hypothetical rules.

Definition 3.3. [Hypothetical Inference] Let R be a hypothetical rulebase. We define an inference system over R as follows, where DB is a database:

Axioms: If $A \in DB$ then $R, DB \vdash A$ is an axiom.
Inference Rules:

1. If $A \leftarrow \phi_1, ..., \phi_m$ is a ground instantiation of a rule in R, then

$$\frac{R, DB \vdash \phi_i \text{ for each } i}{R, DB \vdash A}$$

2. For any ground query of the form $A[add : B_1, ..., B_k]$,

$$\frac{R, DB + \{B_1, ..., B_k\} \vdash A}{R, DB \vdash A[add : B_1, ..., B_k]}$$

3. For any ground query of the form $A[del : B_1, ..., B_k]$,

$$\frac{R, DB - \{B_1, ..., B_k\} \vdash A}{R, DB \vdash A[del : B_1, ..., B_k]}$$

In this inference system, each inference rule has the following interpretation: if the expression above the horizontal line can be inferred, then the expression below the line can also be inferred. The following elementary example shows how inference can be performed in a "top-down" manner by inverting these rules. Additional examples are given in [10, 11]. Many of the examples in this paper are best understood from a top-down point of view.

Example 3.1. [Hypothetical Inference] Suppose that R consists of the following three rules:

$$B \leftarrow C \qquad\qquad C \leftarrow D \qquad\qquad A \leftarrow B[add : D]$$

Then $R, DB \vdash A$ for any database DB. This can be proved by a straightforward, top-down argument:

	$R, DB \vdash A$	
if	$R, DB \vdash B[add : D]$	by inference rule 1, using $A \leftarrow B[add : D]$,
if	$R, DB + D \vdash B$	by inference rule 2,
if	$R, DB + D \vdash C$	by inference rule 1, using $B \leftarrow C$,
if	$R, DB + D \vdash D$	by inference rule 1, using $C \leftarrow D$.

But the last line is trivially true, since it is an axiom.

For convenience, we shall sometimes use rules such as $A \leftarrow C[add : E][del : F]$. This rule is an abbreviation for the following two rules:

$$A \leftarrow B[add : E] \qquad\qquad B \leftarrow C[del : F]$$

where B is a predicate symbol not used anywhere else in the rulebase. Intuitively, these rules say the following: *If C can be inferred after adding E to the database and deleting F, then A can be inferred.*

The following basic result shows that hypothetical inference is monotonic. The proof is a straightforward induction over the length of derivations.

Lemma 3.1 Monotonicity. Suppose that $R_1 \subseteq R_2$ and $DB_1 \subseteq DB_2$. Then, for any ground hypothetical query ϕ, if $R_1, DB_1 \vdash \phi$ then $R_2, DB_2 \vdash \phi$.

3.2. Discussion

It is implicit in Definition 3.3 that the variables in a rule are universally quantified. As in Horn logic, if a variable appears in the body of a rule but not the head, then the universal quantifier can be moved inside the rule body and converted to an existential. For example, in the rule $A(X) \leftarrow B(X,Y)[del : C(X,Y)]$, the variable Y does not appear in the head. If this rule appears in a rulebase, R, then the inference rules in Definition 3.3 imply that for any constant symbol x,

$$R, DB \vdash A(x) \quad \textit{if} \quad R, DB \vdash B(x,y)[del : C(x,y)] \quad \textit{for some constant } y.$$

Consequently, with an abuse of notation, this rule can be read in two equivalent ways:

$$\forall X\ \forall Y\ A(X) \leftarrow B(X,Y)[del : C(X,Y)]$$
$$\forall X\ A(X) \leftarrow \exists Y\ B(X,Y)[del : C(X,Y)]$$

The latter interpretation enables Hypothetical Datalog to express many of the examples given in Section 2. For instance, the following rule defines those students who are "within one course of graduation":

$$within1(S) \leftarrow course(C),\ grad(S)[add : take(S,C)]$$

Observe that in these examples, the hypothetical updates are non-deterministic, since they depend on the non-deterministic choice of a value for a variable (as reflected by the existential quantifier). This raises the practical question of how such choices are to be made during inference. For example, suppose the rulebase contains the rule $A \leftarrow B(X)[del : C(X)]$. How do we find a value for X that makes the premise true?, *i.e.*, which instantiation of $C(X)$ should we delete from the database? Must we test all possible instantiations? This problem has been partially solved by McCarty, who has developed an SLD-style proof procedure for hypothetical insertions, a proof procedure based on top-down inference and unification, as in Prolog [48].

For hypothetical deletions, a simple syntactic restriction suffices. The idea is to ensure that all variables are bound to values in the database during inference. This can be done in several ways. Perhaps the simplest way is to require that every variable in a rule appear in an atomic premise of the rule. We shall say that such rules are *guarded*. Thus, the following rule is guarded:

$$A(X) \leftarrow D(X,Y),\ B(X,Y)[del : C(X,Y)]$$

while the following two rules are not:

$$A(X) \leftarrow B(X,Y)[del : C(X,Y)] \qquad\qquad A(X,Y) \leftarrow B(Y).$$

As illustrated in Section 3.3, if a rulebase consists entirely of guarded rules, then during top-down inference, the variables in a rule can be bound to constants before hypothetical premises are evaluated. The same restriction can be used in logic programs with negation-as-failure to prevent floundering (see Section 5.3). We use it here to avoid similar problems with hypothetical updates.[8]In the examples in this paper, all rulebases in Hypothetical Datalog are guarded or can easily be transformed to be so. The same is true of the rulebases used to prove the complexity and expressibility results in [10, 9, 7, 18, 11, 12]. We emphasize, however, that guardedness is a matter of practice, and none of the theory developed in this paper depends on it. In particular, our soundness and completeness results hold for arbitrary rulebases, not just guardedness ones.

[8]By complicating the definition, the notion of guardedness can be weakened (made less restrictive) in several ways. We leave this as an exercise for the reader.

In Section 2.2, we showed how to express database and rulebase queries in classical logic. These definitions are easily extended to Hypothetical Datalog. Suppose that R_0 is a hypothetical rulebase, and $\psi(X_1, ..., X_n)$ is a hypothetical query with free variables $X_1, ..., X_n$ (in some order). Then, R_0 and ψ express the *database* query $\mathcal{Q}_d$ where

$$(c_1, ..., c_n) \in \mathcal{Q}_d(DB) \quad \textit{iff} \quad R_0,\, DB \vdash \psi(c_1, ..., c_n)$$

for every database DB, and every n-tuple of constant symbols $(c_1, ..., c_n)$. This idea is developed in detail in [10, 7, 12, 9, 18, 11]. Likewise, R_0 and ψ express the *rulebase* query $\mathcal{Q}_r$ where

$$(c_1, ..., c_n) \in \mathcal{Q}_r(R, DB) \quad \textit{iff} \quad R_0 + R,\, DB \vdash \psi(c_1, ..., c_n)$$

for every deductive database $[R, DB]$, and every n-tuple of constants $(c_1, ..., c_n)$. Observe that two classes of rulebase query naturally arise, depending on whether or not R is allowed to contain hypothetical rules as well as Horn rules. Section 2 gave numerous examples of rulebase queries expressed in Hypothetical Datalog. In these examples, R may contain both Horn and hypothetical rules.

3.3. Examples

This section gives simple examples of hypothetical inference, adapted from [10] and [8]. More-complex examples are given in Sections 3.4 and 3.5. The examples all show how to combine simple hypothetical updates into complex hypothetical queries.

Example 3.2. [Cascaded Hypotheticals] Suppose R contains rules defining a predicate, D, plus the following $n+1$ rules:

$$\begin{array}{l} A_1 \leftarrow A_2[del : B_1] \\ A_2 \leftarrow A_3[del : B_2] \\ \quad \cdots \\ A_n \leftarrow A_{n+1}[del : B_n] \\ A_{n+1} \leftarrow D \end{array}$$

Then $\quad R, DB \vdash A_i \quad$ if $\quad R,\, DB - \{B_i, ..., B_n\} \;\vdash\; D.$

The next two examples not only illustrate hypothetical inference, they also illustrate, in simplified form, the kind of rules used in Sections 3.4 and 3.5. In these two examples, the database DB is assumed to include the following atomic formulas, which define a finite sequence of constant symbols $\langle a_1, a_2, ..., a_n, a_{n+1}\rangle$:

$$\begin{array}{l} NEXT(a_1, a_2), \quad NEXT(a_2, a_3), ..., \quad NEXT(a_n, a_{n+1}), \\ FIRST(a_1), \quad LAST(a_{n+1}). \end{array}$$

The first example is a recursive version of Example 3.2.

Example 3.3. [Linear Recursion] Suppose R consists of rules defining the predicate D plus the following three rules:

$$E \leftarrow FIRST(X),\ A(X).$$
$$A(X) \leftarrow NEXT(X,Y),\ A(Y)[del : B(X)].$$
$$A(X) \leftarrow LAST(X),\ D.$$

Then $\quad R, DB \vdash E \quad$ if $\quad R,\ DB - \{B(a_1), ..., B(a_n)\}\ \vdash\ D.$

Example 3.4. [Non-Linear Recursion] Suppose R consists of rules defining the predicate D plus the following four rules:

$$E \leftarrow FIRST(X),\ A(X).$$
$$A(X) \leftarrow NEXT(X,Y),\ A(Y)[add : B(X)],\ A(Y)[add : C(X)].$$
$$A(X) \leftarrow LAST(X),\ D.$$

Then $\ R, DB \vdash E\ $ if the following 2^n expressions are all true:

$$R,\ DB + \{B(a_1), B(a_2), B(a_3), ..., B(a_n)\}\ \vdash\ D$$
$$R,\ DB + \{C(a_1), B(a_2), B(a_3), ..., B(a_n)\}\ \vdash\ D$$
$$R,\ DB + \{B(a_1), C(a_2), B(a_3), ..., B(a_n)\}\ \vdash\ D$$
$$R,\ DB + \{C(a_1), C(a_2), B(a_3), ..., B(a_n)\}\ \vdash\ D$$
$$\cdots$$
$$R,\ DB + \{C(a_1), C(a_2), C(a_3), ..., C(a_n)\}\ \vdash\ D$$

The next example shows how Hypothetical Datalog expresses the rulebase query studied in Section 2.4, which cannot be expressed in classical logic.

Example 3.5. [Rulebase Independence] Let R_0 be a rulebase consisting of the two rules $C \leftarrow G[add : A]$ and $C \leftarrow G[add : B]$. Then for any hypothetical rulebase R, and any database DB,

$$\begin{array}{ll} & R_0 + R,\ DB\ \vdash\ C \\ \textit{if} & R_0 + R,\ DB\ \vdash\ G[add : A] \quad \textit{or} \quad R_0 + R,\ DB\ \vdash\ G[add : B] \\ \textit{if} & R_0 + R,\ DB + A\ \vdash\ G \quad \textit{or} \quad R_0 + R,\ DB + B\ \vdash\ G \\ \textit{if} & R,\ DB + A\ \vdash\ G \quad \textit{or} \quad R,\ DB + B\ \vdash\ G \end{array}$$

where the last line follows by monotonicity (Lemma 3.1). Observe that if C is not mentioned anywhere in R or DB, then every *if* can be replaced by *iff*. The first *if* can be replaced since there is no other way to infer C. The second *if* can be replaced since there is no other way to infer hypothetical insertions. The last *if* can be replaced since R_0 contains no rules that infer G.

The semantics developed in this paper apply to infinite rulebases, infinite databases, and infinite sets of constant symbols. In the next two subsections, however, we consider examples in which all three of these sets are finite. By doing so, we can study the computational properties of the logic. Example 3.4, for instance, illustrates how a fixed set of hypothetical rules can express 2^n meta-level queries, where n is the size of the data domain. The next subsection exploits this capability to construct a rulebase whose data complexity is coNP complete.

3.4. The Power of Hypothetical Inference

In [10, 11], we show that hypothetical inference is more powerful than well-known database query languages, such as relational algebra and Datalog. These languages can only express queries that are computable in polynomial time (polynomial in the size of the data domain). In contrast, the inference system of Definition 3.3 can express queries whose data complexity is complete for EXPTIME, which is known to be strictly more powerful that P. The proof of this complexity result involves an intricate encoding of alternating Turing machine computations using hypothetical rules. In this section, we provide a more intuitive illustration of the power of hypothetical inference. We construct a hypothetical rulebase R that solves the validity problem for propositional formulas. This problem is impossible to solve in Datalog, unless P=NP.

To solve the validity problem, we encode a propositional formula as a database, DB. We then construct the rulebase R so that $R, DB \vdash VALID$ if and only if the propositional formula is valid, where $VALID$ is a predicate of arity zero. Since the rulebase R is fixed (*i.e.*, is independent of the propositional formula), the data complexity of hypothetical inference is coNP-hard.[9]

It is convenient to think of a propositional formula as a combinational circuit. From this point of view, the database DB represents a circuit, and $R, DB \vdash VALID$ if and only if the output of the circuit is 1 for every possible set of inputs. Without loss of generality, we can assume the circuit is constructed entirely of nor gates. We use constant symbols to represent wires and atomic formulas to represent circuit interconnections. The formula $NOR(a, b, c)$ means that wire a is the output of a nor gate whose input wires are b and c. Similarly, the formulas $ZERO(a)$ and $ONE(a)$ mean that the value of wire a is 0 and 1, respectively. Finally, the formula $OUPUT(b)$ means that wire b is the output of the entire circuit.

Given a database DB representing a circuit and its input values, the four rules below evaluate the circuit. If R_0 denotes these rules, then $R_0, DB \vdash EVAL$ if and only if the output of the circuit is 1, where $EVAL$ is a zero-ary predicate symbol. Notice that the rulebase is entirely Horn.

$$EVAL \leftarrow OUPUT(X),\ \ ONE(X).$$
$$ONE(Out) \leftarrow NOR(Out, In_1, In_2),\ \ ZERO(In_1),\ \ ZERO(In_2).$$
$$ZERO(Out) \leftarrow NOR(Out, In_1, In_2),\ \ ONE(In_1).$$
$$ZERO(Out) \leftarrow NOR(Out, In_1, In_2),\ \ ONE(In_2).$$

To determine whether the circuit is valid, we construct rules that hypothetically insert every possible set of input values into the database, one set at a time, and evaluate the circuit for each set. To set the input values, the rules repeatedly select a wire and hypothetically set its value to 0 or 1. So that the rulebase can select input wires one after another, we assume that the input wires are ordered. That is, if the input wires are $a_1, a_2, ..., a_n$, then we assume that the database contains the following atomic formulas:

$$NEXT(a_1, a_2),\ \ NEXT(a_2, a_3), ...,\ \ NEXT(a_n, a_{n+1}),$$
$$FIRST(a_1),\ \ LAST(a_{n+1}).$$

[9] Informally, the *data complexity* of an inference system is the complexity of inference when the rulebase and query are fixed and the database varies (acts as input) [21, 65].

where a_{n+1} is an extra constant symbol, introduced to simplify the construction of the rulebase. (In Section 5.4, we shall see that negation-as-failure permits us to do away with this linear order.)

Let DB be a database that encodes the circuit and orders the inputs in this way. Given this database, we form a rulebase R from the four Horn rules above plus the following three rules:

$$VALID \leftarrow FIRST(X),\ SET(X).$$
$$SET(X) \leftarrow NEXT(X,Y),\ SET(Y)[add: ONE(X)],\ SET(Y)[add: ZERO(X)].$$
$$SET(X) \leftarrow LAST(X),\ EVAL.$$

These rules are best understood from a top-down perspective. The first rule begins the inference process by selecting the first input line. The second rule then recurses down the input lines, selecting one after another. After selecting an input line, the second rule hypothetically sets the input wire first to 0, and then to 1. Formally, the input wire is set to 1 in the left branch of the proof tree, and to 0 in the right branch. In this way, while selecting the n input wires, the second rule generates a proof tree with 2^n branches, each branch corresponding to a different set of input values. In each of these branches, the database contains different hypothetical values for the input wires, exactly as in Example 3.4.

After all the input wires have been assigned a value, the third rule invokes the predicate $EVAL$ to evaluate the circuit, *i.e.*, to determine whether the output of the circuit is 1. $EVAL$ is invoked in each of the 2^n branches of the proof tree, that is, for each possible set of input values. The predicate $VALID$ is inferred if and only if $EVAL$ is inferred in every branch. Thus $R, DB \vdash VALID$ if and only if DB represents a circuit whose output is 1 for every set of input values. The rulebase R therefore solves a problem whose data complexity is coNP-hard.

3.5. Recursion to Exponential Depth

From the perspective of top-down inference, recursion in Datalog can only go to polynomial depth before goals begin to repeat.[10] This is because a goal in Datalog is a ground atomic formula, and for a given rulebase, only polynomially many different goals are possible. However, in the hypothetical system of Definition 3.3, the database changes during inference, so the same atomic goal may repeat many times while the database changes. In hypothetical inference, therefore, the entire expression $R, DB \vdash \psi$ should be viewed as a goal. For a given rulebase, there are exponentially-many goals of this form, since there are exponentially-many databases.[11] Recursion may therefore go to exponential depth before goals begin repeating. That is, there is an exponential upper bound on the number of distinct recursive calls.

This section shows that the exponential upper bound is tight. That is, we construct a rulebase, R, and a database, DB, such that the expression $R, DB \vdash INC$ can be derived only after exponentially-many levels of recursion. In particular, to infer the atom INC, the rulebase effectively counts in binary from 0 to 2^{n-1}, where each increment of the counter requires at least one recursive call.

[10] Polynomial in the size of the data domain, *i.e.*, in the number of constant symbols.

[11] Each database is a subset of the set of all ground atomic formulas, *i.e.*, a subset of a set of polynomial size.

At each step in the derivation, the database represents an n-bit binary number. Initially, the database represents the number 0, and as inference proceeds, the database is hypothetically updated to represent larger and larger numbers. The initial database, DB_0, consists of the following entries:

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

The first two lines represent a linear order $\langle a_1, a_2, ..., a_n \rangle$. We use this order to encode n-bit binary numbers. Each constant a_i represents a binary digit: the formula $ZERO(a_i)$ means that the i^{th} digit is 0, and the formula $ONE(a_i)$ means that the i^{th} digit is 1. In this way, any number between 0 and $2^n - 1$ can be represented as a database. For example, the third line above encodes the n-bit binary representation of zero. (Although numbers up to $2^n - 1$ can be represented in the database, the rulebase will count only up to 2^{n-1}, since this simplifies the construction.)

Let DB_i be a database that represents the number i as an n-bit binary number, for $0 \leq i \leq 2^{n-1}$. We construct a rulebase, R, such that $R, DB_i \vdash INC$. This inference is carried out by counting in binary from i to 2^{n-1}. In particular, we construct R so that

1. The expression $R, DB_i \vdash INC$ is derived by recursively deriving the expression $R, DB_{i+1} \vdash INC$ for $0 \leq i < 2^{n-1}$.
2. The expression $R, DB_{2^{n-1}} \vdash INC$ is derived in one step.

Thus, deriving the expression $R, DB_0 \vdash INC$ requires at least 2^{n-1} recursive calls.

Item 2 is easy to satisfy. During inference, the database represents a binary number between 0 and 2^{n-1}. To determine whether we have reached 2^{n-1}, we need only check the highest-order bit to see if its value is 1. The following rule does exactly this:

$$INC \leftarrow LAST(X), ONE(X).$$

To satisfy item 1, we construct rules which effectively add one to DB_i to produce DB_{i+1}. The idea is to add 1 to the first (lowest-order) digit of the number, and then propagate a carry bit to higher order digits as needed. The following rules implement this process through hypothetical database updates:

$$INC \leftarrow FIRST(X), \; CARRY(X)$$
$$CARRY(X) \leftarrow ONE(X), \; NEXT(X, Y), \; CARRY(Y)[del : ONE(X)][add : ZERO(X)]$$
$$CARRY(X) \leftarrow ZERO(X), \; INC[del : ZERO(X)][add : ONE(X)]$$

The predicate $CARRY(X)$ represents carry bits. During top-down inference, the goal $CARRY(a_i)$ intuitively means, *carry a* 1 *into the* i^{th} *digit*. The first rule starts the process by creating the subgoal $CARRY(a_1)$, which carries a 1 into the first digit. The second rule then recursively generates the subgoals $CARRY(a_2)$, $CARRY(a_3)$, ..., propagating the carry bit to higher-order digits as needed.

Given the subgoal $CARRY(a_i)$, either the second or the third rule is invoked, depending on the value of the i^{th} digit. When the i^{th} digit is 1, the second rule is used. This rule changes the digit to 0 by updating the database hypothetically.

The rule then generates the subgoal $CARRY(a_{i+1})$, thus propagating the carry bit to the next higher digit. The second rule thus simulates the process of adding 1 to a digit whose value is already 1.

When the i^{th} digit is 0, the third rule is used. This rule changes the digit to 1 by updating the database hypothetically. At this point, the carry bit has propagated as far as it can, and the process of incrementing the number in the database is finished. The third rule then starts the process all over again by invoking the predicate INC. In this way, once the number in the database has been incremented, it is incremented again, recursively.

Together, the four rules above define a rulebase R for which the expression $R, DB_0 \vdash INC$ is derivable, but only after exponentially-many levels of recursion.

Note that rulebase R uses both hypothetical addition and deletion. This is crucial to its ability to recurse to exponential depth. When a rulebase uses only hypothetical insertion, recursion can only go to polynomial depth before goals begin to repeat. This is because there are only polynomially many distinct atomic formulas over a given data domain. Thus, only polynomially many atoms can be inserted into a database before reaching saturation, *i.e.*, before every possible atom has been inserted. This is why the data complexity of hypothetical insertion is only PSPACE-complete, while for insertion and deletion together, it is EXPTIME-complete [10, 11]. This difference in complexity is reflected in the difference in maximum recursion depths.

4. MODEL THEORY

Section 3 presented the proof theory for hypothetical rulebases. This section develops the model theory, and shows that the proof theory is sound and complete. A fixpoint theory in the logic-programming tradition is also developed. The fixpoint theory includes a T-operator, which is shown to be monotonic and continuous and to have a unique minimal fixpoint.

Before developing the model theory of hypothetical rulebases in general, it is useful to first consider the special case in which all rules are Horn and all queries are atomic. In this case, the inference system of Section 3.1 is classical, and we can appeal to the classical model theory of Horn rules [5, 63]. According to this theory, a Horn rulebase R and a database DB have a unique minimal model, or least fixpoint, which we denote $lfp(R + DB)$. Furthermore $R, DB \vdash B$ if and only if $B \in lfp(R + DB)$. Posing an atomic query to the rulebase is therefore equivalent to asking if the atom is in the least fixpoint. For this reason, R and DB can be regarded as a description of the much larger database $lfp(R + DB)$, a database which we could explicitly construct if we had the time, the space, and the inclination.

The situation is more complex when we ask hypothetical queries, even when the rulebase is Horn. For instance, to answer the query $R, DB \vdash B[add : C]$, we cannot simply inquire of the fixpoint $lfp(R + DB)$. The difficulty is that the query $B[add : C]$ changes the underlying database DB *before* the rules in R are applied. To answer this query, we need the fixpoint $lfp(R + DB + C)$, not the fixpoint $lfp(R + DB)$. For other hypothetical queries, we will need other fixpoints. For instance, for the query $B[del : C]$, we need $lfp[R + (DB - C)]$. In general, given a hypothetical query ϕ and a database DB, we first generate a new database DB', and then we pose an atomic query to the fixpoint $lfp(R + DB')$.

To answer hypothetical queries, it is convenient to think of a Horn rulebase, R, not as a description of a single database $lfp(R + DB)$, but as a description of a *mapping* that takes a database, DB', as input and returns a larger database, $lfp(R + DB')$, as output. To answer atomic queries, we need access to just one value of the mapping, $lfp(R + DB)$, but to answer hypothetical queries, we need access to its other values as well.

This point of view generalizes from Horn rules to hypothetical rules in a straightforward way. A hypothetical rulebase R can be thought of as specifying a mapping M. This mapping takes a database DB as input and returns a larger database $M(DB)$ as output. The rules in R impose restrictions on the mapping. For instance, the Horn rule $A \leftarrow B$ says that for every database DB, if $B \in M(DB)$ then $A \in M(DB)$. This restriction is "context free" in that the value of M at one point is independent of the values of M at other points. In contrast, the restrictions imposed by hypothetical rules are "context sensitive." For instance, the rule $A \leftarrow B[del : C]$ says that for all databases DB, if $B \in M(DB - C)$ then $A \in M(DB)$. In this way, the value of a mapping at one point is constrained by the values of the mapping at other points. Hypothetical rules can therefore impose requirements on a mapping that are impossible to express with Horn rules.

In general, there will be many mappings that satisfy the constraints imposed by a rulebase R. We say that each such mapping is a *model* of R. This leads in a natural way to a model-theoretic definition of entailment. The rest of this section formalizes this idea and shows that the inference system of Definition 3.3 is sound and complete with respect to this semantics.

Before developing this semantics, it is worth noting the relationship between the mappings of this section and the Kripke-like structures of intuitionistic logic [29, 11, 18]. An essential component of any intuitionistic structure is a mapping, π, from a set of states to a set of databases. The mappings developed in this section are both a restriction and a generalization of these intuitionistic mappings. On the one hand, the mappings of this section are more restrictive in that they only take a database as input, whereas intuitionistic mappings can take arbitrary inputs. On the other hand, they are more general in that they are not required to be monotonic, as intuitionistic mappings are. Because of this non-monotonicity, the semantics developed in this section can be extended in a straightforward way to a semantics for hypothetical reasoning with negation-as-failure (Section 5). In contrast, adding negation-as-failure to intuitionistic logic is problematic, as pointed out in [32] and [17]. Finally, it is worth pointing out, that like the models of this section, the canonical intuitionistic model developed in [11, 7] is essentially a mapping from databases to databases. In a similar fashion, the specialized intuitionistic models developed in [50] and [18] are mappings from rulebases to databases. This section builds upon and extends this tradition.

The development here builds especially upon ideas introduced in [10]. For instance, the idea of interpreting a rulebase as a mapping, and the definition of a hypothetical T-operator were both used in [10]. However, as explained in [10], the treatment there is entirely proof-theoretic, is narrowly aimed at establishing complexity results, and only those concepts needed to achieve that end were developed. In particular, a full-blown model theory and fixpoint theory were intentionally avoided. For instance, the development in [10] includes no notion of a model of a rulebase, nor of logical entailment, nor of soundness and completeness. The development of the T-operator is equally Spartan. For instance, it is not treated as a lattice operator, nor is it shown to be monotonic and continuous, nor is it shown to have a unique minimal fixpoint. In fact, the very idea of a fixpoint is not even mentioned. In short, the development in [10] does not provide the substance of a logical semantics. This section does, thus completing the logical development of Hypothetical Datalog in the negation-free case.

4.1. Models and Entailment

To make the preceding discussion precise, let $\mathcal{DB}$ be the set of ground atomic formulas constructible from the predicate symbols and constant symbols in our language. Any database constructible from these symbols is a subset of $\mathcal{DB}$. A mapping $M : 2^{\mathcal{DB}} \rightarrow 2^{\mathcal{DB}}$ thus takes a database DB as input, and returns another database $M(DB)$ as output. These mappings are the basic structures of our model theory.

Not all such mappings can be a model of a rulebase, R, however. In fact, there are some mappings that are not a model of any rulebase. For instance, according to the inference system of Definition 3.3, $R, DB \vdash A$ for any atom A in the database DB. We should therefore expect that $DB \subseteq M(DB)$ for every model M of R. The semantic development is simplified somewhat if we impose this restriction on mappings at the outset. With this in mind, we make the following definitions.

Definition 4.1. A mapping $M : 2^{\mathcal{DB}} \rightarrow 2^{\mathcal{DB}}$ is inflationary if $DB \subseteq M(DB)$ for all databases $DB \subseteq \mathcal{DB}$.

Definition 4.2. [Structures] A hypothetical structure is an inflationary mapping $M : 2^{\mathcal{DB}} \rightarrow 2^{\mathcal{DB}}$.

Definition 4.3. [Satisfaction] Let M be a hypothetical structure, let DB be a database, let $A, B_1, ..., B_k$ be ground atomic formulas, and let $\phi_1, ..., \phi_k$ be ground hypothetical queries. Then

$$
\begin{array}{lll}
M, DB \models A & \textit{iff} & A \in M(DB) \\
M, DB \models A[add : B_1, ..., B_k] & \textit{iff} & M, DB + \{B_1, ..., B_k\} \models A \\
M, DB \models A[del : B_1, ..., B_k] & \textit{iff} & M, DB - \{B_1, ..., B_k\} \models A \\
M, DB \models \phi_1, ..., \phi_k & \textit{iff} & M, DB \models \phi_i \quad \textit{for each } 1 \leq i \leq k.
\end{array}
$$

If $M, DB \models \phi$ then we say that M *satisfies* ϕ at DB.

Definition 4.4. [Instantiation] Let R be a hypothetical rulebase. Then $inst(R)$ is the set of all ground instances of the rules in R, that is, the result of replacing, in every possible way, the variables in R by constant symbols in the language.

Definition 4.5. [Models] Let R be a hypothetical rulebase, and let M be a hypothetical structure. Then M is a model of R if and only if the following is true for all databases DB, and all rules $A \leftarrow \phi_1, ..., \phi_k$ in $inst(R)$:

$$M, DB \models A \quad \textit{if} \quad M, DB \models \phi_1, ..., \phi_k$$

Definition 4.6. [Entailment] Let R be a hypothetical rulebase, let DB be a database, and let ϕ be a ground hypothetical query. Then $R, DB \models \phi$ if and only if $M, DB \models \phi$ for all models M of R.

4.2. Soundness and Completeness

This section establishes the soundness and completeness of the inference system of Definition 3.3 with respect to the model theory developed above. That is, we prove the following two theorems.

Theorem 4.1 Soundness. If $R, DB \vdash \phi$ then $R, DB \models \phi$, for any ground hypothetical query ϕ, and any database DB.

Theorem 4.2 Completeness. If $R, DB \models \phi$ then $R, DB \vdash \phi$, for any ground hypothetical query ϕ, and any database DB.

The proof of soundness is not difficult, and is an immediate consequence of the next lemma and of Definition 4.6.

Lemma 4.3. Let M be a model of a hypothetical rulebase R. If $R, DB \vdash \phi$ then $M, DB \models \phi$, for any database DB, and any ground hypothetical query ϕ.

PROOF. M satisfies a set of results that parallel the axioms and inference rules of Definition 3.3. First, since M is inflationary, we have that $M, DB \models A$ for any ground atomic formula, A, in DB. This corresponds to the axioms of Definition 3.3. Likewise, the following results correspond to the three inference rules of Definition 3.3:

1. If $A \leftarrow \phi_1, ..., \phi_m$ is a ground instantiation of a rule in R, then if $M, DB \models \phi_i$ for each i, then $M, DB \models A$
2. For any ground query of the form $A[add : B_1, ..., B_k]$, if $M, DB + \{B_1, ..., B_k\} \models A$ then $M, DB \models A[add : B_1, ..., B_k]$
3. For any ground query of the form $A[del : B_1, ..., B_k]$, if $M, DB - \{B_1, ..., B_k\} \models A$ then $M, DB \models A[del : B_1, ..., B_k]$

Item 1 follows because M is a model of R. Items 2 and 3 follow immediately from the definition of satisfaction. □

To prove completeness, we define a hypothetical structure, M_R, which we call the *canonical model* of the rulebase R. This model, defined proof-theoretically, provides the necessary link between inference and semantics.

Definition 4.7. Let R be a hypothetical rulebase. Then M_R is the hypothetical structure such that $M_R(DB) = \{A \in \mathcal{DB} \mid R, DB \vdash A\}$ for every database DB.

Clearly M_R is a mapping from $2^{\mathcal{DB}}$ to $2^{\mathcal{DB}}$. The mapping is inflationary, since $R, DB \vdash A$ for all $A \in DB$. The canonical model is therefore a hypothetical structure. The following basic result says that satisfaction in this structure is equivalent to inference from the rulebase R.

Lemma 4.4. $M_R, DB \models \phi$ iff $R, DB \vdash \phi$, for any ground hypothetical query ϕ.

PROOF. There are three cases depending on whether ϕ is atomic, a hypothetical insertion, or a hypothetical deletion. In each case, the proof is an immediate consequence of the definition of M_R. For instance, when ϕ is a hypothetical deletion,

	$M_R, DB \models A[del : B_1, ..., B_k]$	
iff	$M_R, DB - \{B_1, ..., B_k\} \models A$	by Definition 4.3,
iff	$A \in M_R(DB - \{B_1, ..., B_k\})$	by Definition 4.3,
iff	$R, DB - \{B_1, ..., B_k\} \vdash A$	by Definition 4.7,
iff	$R, DB \vdash A[del : B_1, ..., B_k]$	by Definition 4.3.

□

Lemma 4.5. The canonical model M_R is a model of R.

PROOF. Let $A \leftarrow \phi_1, ..., \phi_k$ be a ground instance of a rule in R. Then, for any database DB,

if	$M_R, DB \models \phi_1, ..., \phi_k$	
then	$M_R, DB \models \phi_i$	for each i, by Definition 4.3,
	$R, DB \vdash \phi_i$	for each i, by Lemma 4.4,
	$R, DB \vdash A$	by Definition 3.3, since $A \leftarrow \phi_1, ..., \phi_k$ is an instance of a rule in R,
	$A \in M_R(DB)$	by Definition 4.7,
	$M_R, DB \models A$	by Definition 4.3.

M_R is thus a model of $A \leftarrow \phi_1, ..., \phi_k$. Since this is true for every instance of every rule in R, M_R is a model of R by Definition 4.5. □

Completeness follows immediately from Lemmas 4.4 and 4.5. To see this, suppose that $R, DB \models \phi$. Then $M, DB \models \phi$ for every model M of R by Definition 4.6; so $M_R, DB \models \phi$, by Lemma 4.5; so $R, DB \vdash \phi$ by Lemma 4.4.

4.3. Fixpoint Semantics

Section 4 developed a model-theoretic semantics for hypothetical rulebases. This section complements the model theory with a fixpoint theory, in the tradition of logic programming. We show that every rulebase has a unique minimal model, which can be constructed in a bottom-up fashion by repeated application of a T operator. The development follows the approach originally introduced by Apt, Van Emden and Kowalski [63, 5], which is now standard in logic programming. First, we define a lattice of structures. Then, we define an operator on this lattice, and show that it is monotonic and continuous. Finally, we invoke the Tarski fixpoint theorem.

Just as the fixpoint theory of classical Horn logic is based on a lattice of classical structures, the fixpoint theory of hypothetical inference is based on a lattice of hypothetical structures. The first definition presents the basic lattice operations.

Definition 4.8. Let M_1 and M_2 be hypothetical structures, and let $\mathcal{M}$ be a set of hypothetical structures. Then

- $M_1 \leq M_2$ if and only if $M_1(DB) \subseteq M_2(DB)$ for every database DB.
- $\sqcup\mathcal{M}$ is the hypothetical structure such that $\sqcup\mathcal{M}(DB) = \cup_{M \in \mathcal{M}} M(DB)$ for every database DB.

- $\sqcap\mathcal{M}$ is the hypothetical structure such that $\sqcap\mathcal{M}(DB) = \cap_{M\in\mathcal{M}}\ M(DB)$ for every database DB.

The operators $\sqcup$ and $\sqcap$ are called the *join* and *meet*, respectively. Note that $\leq$ is a partial order. The following basic result is straightforward.

Lemma 4.6. The set of all hypothetical structures is a complete lattice under the partial order $\leq$ and the operations $\sqcup$ and $\sqcap$.

Because a hypothetical structure is an inflationary mapping, the minimal element of this lattice is the identity mapping $\mathcal{I}$, where $\mathcal{I}(DB) = DB$ for all databases DB. The maximal element is the structure $M_\top$ where $M_\top(DB) = \mathcal{DB}$, the set of all ground atomic formulas. Note that $M_\top$ is a model of every hypothetical rulebase, since the head of every instance of every rule is true in this structure.

The following result about the lattice of hypothetical structures is straightforward.

Lemma 4.7. If $M_1 \leq M_2$ and $M_1, DB \models \phi$ then $M_2, DB \models \phi$, for any ground hypothetical query ϕ, and any database DB.

For each hypothetical rulebase R, we define an operator T_R. This operator takes a hypothetical structure M as input and returns another hypothetical structure $T_R(M)$ as output.

Definition 4.9. [T Operator] Let R be a set of hypothetical rules. Then T_R is a mapping from hypothetical structures to hypothetical structures. In particular, for each structure M, and each database DB, the value of $[T_R(M)](DB)$ is

$$DB\ \cup\ \{A \mid\ M, DB \models \phi_1, ..., \phi_k\ \textit{ for some rule } A \leftarrow \phi_1, ..., \phi_k \textit{ in } inst(R)\}$$

In this definition, the union with database DB is included to ensure that $T_R(M)$ is inflationary and thus a well-defined hypothetical structure (See Definition 4.2). The following basic results about the operator T_R are straightforward.

Lemma 4.8. Let R be a hypothetical rulebase, and let M be a hypothetical structure. Then M is a model of R if and only if it is a fixpoint of T_R, that is, if and only if $T_R(M) \leq M$.

Monotonicity. We now show that the operator T_R is *monotonic*; that is, if the input to the operator increases, then so does the output. Because it is monotonic, the operator has a least fixpoint *lfp*(T_R), which is also the unique minimal model of the rulebase R. We show that the minimal model is identical to the canonical model developed in Section 4.2.

Lemma 4.9 Monotonicity. If $M_1 \leq M_2$ then $T_R(M_1) \leq T_R(M_2)$.

PROOF. Follows immediately from Lemma 4.7. □

Since T_R is a monotonic operator on a complete lattice, the meet of any of its fixpoints is also a fixpoint. This result, which generalizes the model intersection property of Horn clause logic [63, 5], follows immediately by the Tarski fixpoint theorem [62]. As a special case, the meet of all the fixpoints of T_R is itself a fixpoint of T_R, the least fixpoint. We thus have the following result.

Theorem 4.10 Least Fixpoint. The operator T_R has a least fixpoint $lfp(T_R)$. That is, $lfp(T_R) \leq M$ for all fixpoints M of T_R.

Since the fixpoints of T_R are exactly the models of R, it follows that every hypothetical rulebase has a unique minimal model, which is exactly $lfp(T_R)$. As the following corollary shows, the minimal model is special in that satisfaction in this model is equivalent to logical entailment.

Corollary 4.11. For any ground hypothetical query ϕ,

$$R, DB \models \phi \quad \textit{iff} \quad lfp(T_R), DB \models \phi$$

Proof. If $R, DB \models \phi$ then $M, DB \models \phi$ for every model M of R by Definition 4.6. In particular, $lfp(T_R), DB \models \phi$ since $lfp(T_R)$ is a model of R. In the other direction, if $lfp(T_R), DB \models \phi$ then $M, DB \models \phi$ for any model M of R by Lemma 4.7, since $lfp(T_R) \leq M$. Thus $R, DB \models \phi$. $\square$

The next result is included for completeness. It shows that the minimal model $lfp(T_R)$ is identical to the canonical model M_R developed in Section 4.2.

Lemma 4.12. The minimal model of R is identical to the canonical model of R.

Proof. For any ground atomic formula A, and any database DB,

$$\begin{array}{lll} & A \in M_R(DB) & \\ \textit{iff} & R, DB \vdash A & \text{by Definition 4.7,} \\ \textit{iff} & R, DB \models A & \text{by soundness and completeness,} \\ \textit{iff} & lfp(T_R), DB \models A & \text{by Corollary 4.11,} \\ \textit{iff} & A \in [lfp(T_R)](DB) & \text{by Definition 4.3.} \end{array}$$

Thus $[lfp(T_R)](DB) = M_R(DB)$ for all databases DB. Hence $lfp(T_R) = M_R$. $\square$

Continuity. We now show that the operator T_R is *continuous*. Intuitively, this means that if a sequence of input values converges to M, then the sequence of output values converges to $T_R(M)$. Because the operator is continuous, its least fixpoint $lfp(T_R)$ can be constructed by an iterative, bottom-up procedure. This procedure starts at the bottom of the lattice, and climbs higher and higher up the lattice, converging on the least fixpoint. Continuity is proved by the first three lemmas below.

Lemma 4.13. Let $\mathcal{M}$ be a set of hypothetical structures. Then, for any database DB, and any ground hypothetical query ϕ,

$$\sqcup\mathcal{M}, DB \models \phi \quad \textit{iff} \quad M, DB \models \phi \ \textit{for some } M \textit{ in } \mathcal{M}.$$

Proof. There are three cases, depending on whether ϕ is atomic, a hypothetical insertion, or a hypothetical deletion. In each case, the proof is straightforward. For instance, when ϕ is a hypothetical deletion,

	$\sqcup\mathcal{M}, DB \models A[del : B_1, ..., B_k]$	
iff	$\sqcup\mathcal{M}, DB - \{B_1, ..., B_k\} \models A$	by Definition 4.3,
iff	$A \in [\sqcup\mathcal{M}](DB - \{B_1, ..., B_k\})$	by Definition 4.3,
iff	$A \in \cup_{M \in \mathcal{M}}\ M(DB - \{B_1, ..., B_k\})$	by Definition 4.8,
iff	$A \in M(DB - \{B_1, ..., B_k\})$	for some M in $\mathcal{M}$,
iff	$M, DB - \{B_1, ..., B_k\} \models A$	by Definition 4.3,
iff	$M, DB \models A[del : B_1, ..., B_k]$	by Definition 4.3.

□

Lemma 4.14. Let $M_1 \leq M_2 \leq M_3 ...$ *be a (possibly infinite) non-decreasing sequence of hypothetical structures. Then, for any database* DB*, and any ground hypothetical queries* $\phi_1, ..., \phi_k$*,*

if $\sqcup_{i \geq 1} M_i, DB \models \phi_1, ..., \phi_k$ *then* $M_{i_0}, DB \models \phi_1, ..., \phi_k$ *for some* i_0.

Proof. Suppose that $\sqcup_{i \geq 1} M_i, DB \models \phi_1, ..., \phi_k$. Then, for all $1 \leq j \leq k$,

$\sqcup_{i \geq 1} M_i, DB \models \phi_j$	by Definition 4.3,
$M_{i_j}, DB \models \phi_j$	for some i_j, by Lemma 4.13,
$M_i, DB \models \phi_j$	for all $i \geq i_j$, by Lemma 4.7, since $M_i \geq M i_j$,
$M_{i_0}, DB \models \phi_j$	for $i_0 = max(i_1, ..., i_k)$.

Hence $M_{i_0}, DB \models \phi_1, ..., \phi_k$. □

Lemma 4.15 Continuity. If $M_1 \leq M_2 \leq M_3 \leq \ldots$ *is a (possibly infinite) non-decreasing sequence of hypothetical structures, then* $T_R(\sqcup_{i \geq 1} M_i) = \sqcup_{i \geq 1} T_R(M_i)$.

Proof. We use $\sqcup_i$ as an abbreviation for $\sqcup_{i \geq 1}$. Clearly $M_j \leq \sqcup_i M_i$ for any $j \geq 1$. Thus $T_R(M_j) \leq T_R(\sqcup_i M_i)$ by the monotonicity of T_R, so $\sqcup_j T_R(M_j) \leq T_R(\sqcup_i M_i)$. We must therefore prove the converse, that $T_R(\sqcup_i M_i) \leq \sqcup_j T_R(M_j)$. To do this, choose a database DB, and an atom A in $[T_R(\sqcup_i M_i)](DB)$. We must show that $A \in [\sqcup_i T_i(M_i)](DB)$. There are two cases. First, suppose that $A \in DB$. Then

$A \in [T_i(M_i)](DB)$	for every i, by Definition 4.9,
$A \in \cup_i [T_i(M_i)(DB)]$	
$A \in [\sqcup_i T_i(M_i)](DB)$	by Definition 4.8.

Second, suppose that $A \notin DB$. Then, since A is in $[T_R(\sqcup_i M_i)](DB)$,

$[T_R(\sqcup_i M_i)], DB \models A$	by Definition 4.3,
$\sqcup_i M_i, DB \models \phi_1, ..., \phi_k$	for some rule $A \leftarrow \phi_1, ..., \phi_k$ in $inst(R)$, by Definition 4.9,
$M_i, DB \models \phi_1, ..., \phi_k$	for some i, by Lemma 4.14,
$A \in [T_R(M_i)](DB)$	by the Definition 4.9,
$A \in \cup_i [T_R(M_i)(DB)]$	
$A \in [\sqcup_i T_R(M_i)](DB)$	by Definition 4.8.

We have thus shown that $[T_R(\sqcup_i M_i)](DB) \subseteq [\sqcup_i T_R(M_i)](DB)$ for every database DB. Hence $T_R(\sqcup_i M_i) \leq \sqcup_i T_R(M_i)$ by Definition 4.8. □

The following standard definitions define $T_R^*(M)$, which is the result of repeatedly applying the operator T_R to M.

Definition 4.10. If R is a hypothetical rulebase, and M a hypothetical structure, then

- $T_R^0(M) \ = \ M$
- $T_R^{i+1}(M) \ = \ T_R[T_R^i(M)]$
- $T_R^*(M) \ = \ \sqcup_{i \geq 0} T_R^i(M)$

Starting from the identity mapping $\mathcal{I}$, the operator T_R produces a sequence of larger-and-larger hypothetical structures $T_R^0(\mathcal{I})$, $T_R^1(\mathcal{I})$, $T_R^2(\mathcal{I})$ ... The main result of this section is that this sequence converges to the least fixpoint of T_R, as stated in the next theorem. This result follows from the monotonicity and continuity of the operator T_R, as originally shown by Tarski [62].

Theorem 4.16. $lfp(T_R) = T_R^*(\mathcal{I})$.

5. NEGATION AS FAILURE

In [10, 11], it is shown that the inference system of Definition 3.3 has a data complexity that is EXPTIME-complete. This means that the inference system has great computational power, including the power to solve any NP-complete problem. Despite this power, though, there are some simple, low-complexity queries that the inference system cannot express. This is because it is *monotonic*: as the database expands, the answer to a query also expands. This behavior is typical of many logical systems, such as classical Horn logic, full classical logic, intuitionistic logic, and modal logics. Such systems cannot express non-monotonic queries, such as relational algebra queries involving complementation. Retrieving those students who are *not* eligible to graduate is an example of a non-monotonic query. To capture such queries in the framework of hypothetical inference, this section augments our logic with a well-known non-monotonic operator: *negation-as-failure*. This is a natural extension to the logic since any practical logic-programming language has to incorporate negation-as-failure in some form. Furthermore, this operator allows the logic to express *all* the database queries in EXPTIME, both monotonic and non-monotonic [10, 11]. In this sense, the logic extended with negation is expressively complete.

In the extended logic, hypothetical rules may have negated premises. Thus, rules of the form $A \leftarrow \sim B[del : C]$ are allowed. The expression $\sim B[del : C]$ is interpreted as the failure to prove $B[del : C]$. Thus, A is inferred if $B[del : C]$ cannot be inferred. This semantics is similar to the semantics of negation as commonly defined in Horn logic programming. Indeed, much of this section extends the theory of negation-as-failure from Horn rules to hypothetical rules, generalizing several important techniques.

A major difficulty with negation-as-failure is that it is not always well defined. This is true both for Horn rulebases and for hypothetical rulebases. For instance, given the two rules $A \leftarrow \sim B$ and $B \leftarrow \sim A$, it is unclear whether A is to be inferred, or B, or both, or neither. Numerous approaches to this problem have been proposed. One approach has been to identify classes of Horn rulebases for which such problems do not arise. Perhaps the best known of these classes is the *stratified rulebases* [4]. These rulebases are layered, and within each layer, a negated premise refers only to rules found in the layers below. In this way, recursion never occurs through a negated premise and the semantics of negation is always well-defined. Another approach has been to allow limited recursion through negation. This approach generalizes the notion of stratification to *local stratification* [55]. Intuitively, a rulebase is locally stratified if its ground instantiation is stratified (with possibly infinitely-many strata). Locally stratified rulebases are perhaps the largest class of logic programs for which the semantics of negation is not controversial. Another approach to negation-as-failure attempts to define semantics for arbitrary Horn programs with negation. There have been numerous attempts here too. Perhaps the best known are the well-founded semantics [34, 64] and the stable model semantics [35]. Although these semantics offer different interpretations of recursion through negation, they are equivalent when rulebases are stratified or locally stratified.

This section focuses on stratified rulebases, extending the theory of stratified negation from Horn rules to hypothetical rules. By focussing on stratified rules, we factor out the many problems resulting from recursion through negation. Moreover, stratification is precisely the restriction needed to prove the complexity and expressibility results in [10, 9, 7, 18, 11, 12]. It is also worth noting that as in Horn logic, stratified rulebases act as a kind of benchmark in the semantics of negation: without a clear semantics for the stratified case, it would be hard to judge whether a semantics for arbitrary rulebases is correct.

In [10], we defined an operational semantics for stratified hypothetical rulebases. This semantics was sufficient for proving complexity results. However, a model theory and a logical proof theory were not presented. This section provides these missing pieces. We first generalize the notion of stratification from Horn rulebases to hypothetical rulebases. We then develop the proof theory and model theory for stratified rulebases. The proof theory is based on the stratified semantics of Apt, Blair and Walker [4], and the model theory is based on the perfect model semantics of Przymusinski [55]. By combining and extending these two traditions, we achieve a semantics that does not depend on any finiteness assumptions. Thus, all quantities may be infinite: the database, the rulebase, the set of constant symbols, and even the number of strata.

The following example is adapted from [17].

Example 5.1. [Negation-as-failure] The rules below are part of a stratified hypothetical rulebase that defines a student's eligibility for financial aid. Intuitively, a student s is eligible for a stipend if he is a near-graduate but not a graduate. On the other hand, if he is neither a graduate nor a near-graduate, then he is eligible for a fellowship.

$$\begin{aligned}
&stipend(S) \leftarrow admitted(S),\ near_grad(S),\ \sim grad(S).\\
&fellowship(S) \leftarrow admitted(S),\ \sim near_grad(S),\ \sim grad(S).\\
&near_grad(S) \leftarrow course(C),\ grad(S)[add : take(S,C)]
\end{aligned}$$

In applying the rule for $near\_grad(s)$, we ask if student s is within one course of graduation. That is, is there some course c such that, if $take(s,c)$ were assumed to be true, then $grad(s)$ would also be true? The student is eligible for a fellowship only if this hypothetical test fails. Conversely, he is eligible for a stipend only if it succeeds.

Note that this hypothetical test is vacuously true if $grad(s)$ is true (as long as there exists a course somewhere in the database!). But we do not want to give a stipend to a student who has already graduated, so we include the test $\sim grad(s)$ in the rule for $stipend(s)$. This means that a stipend is available only to those students who need *exactly* one course to graduate.

5.1. Stratification

This section gives a precise definition of stratified hypothetical rulebases.

The first step is to extend some of the definitions given in Section 3.1. The hypothetical queries defined in Definition 3.1 will now be referred to as *positive* hypothetical queries. If ϕ is a positive hypothetical query, then $\sim \phi$ is a *negative* hypothetical query. A hypothetical query thus has six possible forms: B, $B[add: C_1, ..., C_k]$, $B[del: C_1, ..., C_k]$, or the negation of any of these, where B and each C_i are atomic formulas. In each case, we call the predicate symbol of B the *goal predicate* of the query. Thus, in the rule $A \leftarrow B_1, \sim B_2[add: C]$, both B_1 and B_2 are goal predicates. In a top-down inference procedure, goal predicates become subgoals and are resolved against the heads of rules. For convenience, the predicate symbol A, appearing in the rule head, shall be called the *head predicate*.

Definition 5.1. [Stratified Rulebase] Let R be a set of hypothetical rules with possibly negated premises. R is stratified if it is the disjoint union of (possibly empty) rulebases $R_1, R_2, R_3, ...$, where

1. each predicate symbol is defined in at most one of the R_i;
2. no rule in R_1 has a negated premise;
3. if a goal predicate P appears in a negated premise in R_i, then each rule with head predicate P is contained in $\cup_{j<i} R_j$; and
4. if a goal predicate P appears in a non-negated premise in R_i, then each rule with head predicate P is contained in $\cup_{j\leq i} R_j$.

In this definition, we call the sequence $R_1, R_2, ...$ a *stratification* of R. We also say that the sequence $R_1, R_2, ...$ *stratifies* R. We call R_i the i^{th} *stratum*, and we also call R_1 the *bottom stratum*. We say that a predicate symbol *belongs* to the stratum in which it is defined. If a predicate symbol is not defined in any stratum, then it belongs to stratum 1. Such predicate symbols are defined only in the database. (Note that any predicate symbol may appear in the database, even if it is defined in the rulebase.) Finally, we say that a hypothetical query belongs to the stratum to which its goal predicate belongs.

Definition 5.1 guarantees that if recursion occurs, then it occurs only within a single stratum and that it never occurs through negation. Items 1 and 2 in this definition are not strictly necessary, since they do not limit the power or expressibility of the system; but they simplify the development that follows. Note that by item 2, if every rule in R contains a negated premise, then the bottom stratum is empty. If R is finite, then infinitely many of the strata are empty.

Notice that only goal predicates and head predicates play a role in the definition of stratification. Thus, in the rule $A \leftarrow B[add: C]$, rules defining C may appear in any stratum, including strata *above* those in which A and B are defined. Intuitively, the atom B is "executed" in order to prove A, while the atom C is "assumed" to be true in order to prove A. This is why C is not considered for the stratification;

i.e., because C is not "executed," it cannot lead to recursion through negation.

Example 5.2. [Stratification] The following rulebase is stratified by R_0, R_1, R_2:

$$R_2 \begin{cases} A_2(X) \leftarrow B_2(X,Y),\ A_2(Y)[add : C_3(Y)]. \\ A_2(X) \leftarrow D_2(X),\ \sim A_1(X)[add : C_2(X)]. \end{cases}$$
$$R_1 \begin{cases} A_1(X) \leftarrow B_1(X,Y),\ A_1(Y)[add : C_1(Y)]. \\ A_1(X) \leftarrow D_1(X),\ \sim A_0(X). \end{cases}$$
$$R_0 \begin{cases} A_0(X) \leftarrow B_0(X,Y),\ A_0(Y)[add : C_0(Y)]. \\ A_0(X) \leftarrow D_0(X). \end{cases}$$

5.2. Stratified Inference

This section develops the proof theory of stratified hypothetical rulebases. The idea is to derive expressions of the form $R \vdash \phi$ stratum-by-stratum, using the derivations of one stratum as the starting point for the derivations of the stratum above.

Given a stratified rulebase R, we define a sequence of inference systems, one for each stratum. These systems generalize the inference system of Definition 3.3 in two ways. First, each inference system corresponds to a particular stratum of R. The j^{th} inference system uses only those rules in the j^{th} stratum. Second, each inference system is provided with an unspecified set of axioms, $\mathcal{A}$. Each inference system thus defines a mapping cl_j that takes a set of axioms $\mathcal{A}$ as input, and returns a set of inferred expressions $cl_j(\mathcal{A})$ as output. We call $cl_j(\mathcal{A})$ the *j^{th} closure of $\mathcal{A}$*. These mappings encapsulate inference as a single operation, thus providing a convenient separation between inference, which occurs within strata, and negation-as-failure, which occurs between strata.

The proof theoretic semantics of R is defined stratum-by-stratum in terms of these mappings. Given the output from cl_j, we apply the closed world assumption to it, and use the resulting expressions as the input to cl_{j+1}. In this way, the derivations of one stratum are the starting point for the derivations of the stratum above. The rest of this section makes these ideas precise. To simplify the presentation, we assume that a rulebase R and a stratification $R_1, R_2, R_3, \ldots$ are given. We shall also refer to ground hypothetical queries as *goals*.

Definition 5.2. An inference expression for R is an expression of the form $R, DB \vdash \phi$, where DB is a database and ϕ is a goal. The inference expression is positive if ϕ is positive, and negative if ϕ is negative.

Definition 5.3. [Inference] Let $\mathcal{A}$ be a set of inference expressions for R. For each stratum, R_j, we define an inference system as follows, where DB is an arbitrary database:

Axioms:

1. If $A \in DB$ then $R, DB \vdash A$ is an axiom.
2. Each expression in $\mathcal{A}$ is an axiom.

Inference Rules:

1. If $A \leftarrow \phi_1, \ldots, \phi_m$ is a ground instantiation of a rule in the j^{th} stratum

of R, then

$$\frac{R, DB \vdash \phi_i \quad \text{for each } i}{R, DB \vdash A}$$

2. For any ground query of the form $A[add : B_1, ..., B_k]$ belonging to the j^{th} stratum,

$$\frac{R, DB + \{B_1, ..., B_k\} \vdash A}{R, DB \vdash A[add : B_1, ..., B_k]}$$

3. For any ground query of the form $A[del : B_1, ..., B_k]$ belonging to the j^{th} stratum,

$$\frac{R, DB - \{B_1, ..., B_k\} \vdash A}{R, DB \vdash A[del : B_1, ..., B_k]}$$

$cl_j(\mathcal{A})$ denotes the set of inference expressions derivable in this system.

Like all Gentzen-style inference systems, the inference system in Definition 5.3 is monotonic in the set of axioms. It is also idempotent and inflationary. We thus have the following basic results.

Lemma 5.1. cl_j is a function which maps sets of inference expressions for R into sets of inference expressions for R. Furthermore, this function has the following properties:

- *Monotonicity: If $\mathcal{A} \subseteq \mathcal{B}$ then $cl_j(\mathcal{A}) \subseteq cl_j(\mathcal{B})$*
- *Idempotence: $cl_j(\mathcal{A}) = cl_j(cl_j(\mathcal{A}))$*
- *Inflation: $\mathcal{A} \subseteq cl_j(\mathcal{A})$*

Given a set of inference expressions, $\mathcal{A}$, we define a new set, $\overline{\mathcal{A}}$. Informally, $\overline{\mathcal{A}}$ is the result of applying the *closed world assumption* to $\mathcal{A}$; *i.e.*, if an expression is not true, then assume it is false. This is the basis of negation-as-failure.

Definition 5.4. [Closed World Assumption] Let $\mathcal{A}$ be a set of inference expressions for R. Then $\overline{\mathcal{A}}$ is another set of inference expressions for R, where for any database DB, and any positive goal ϕ,

- $R, DB \vdash \phi \in \overline{\mathcal{A}}$ *iff* $R, DB \vdash \phi \in \mathcal{A}$
- $R, DB \vdash \sim \phi \in \overline{\mathcal{A}}$ *iff* $R, DB \vdash \phi \notin \mathcal{A}$

We have now assembled the main components of our proof theory for stratified rulebases: (i) Definition 5.4 captures the idea of negation-as-failure, which occurs between strata, and (ii) Definition 5.3 captures the idea of inference, which occurs within strata. It remains to combine these ideas into a unified semantics of stratified rulebases. The following definition does exactly this. By alternating inference with negation-as-failure, it constructs a set of inference expressions stratum-by-stratum in a bottom-up fashion.

Definition 5.5. [Inference with Negation]

- $\mathcal{A}^0 = \overline{\{\}}$
- $\mathcal{A}^{j+1} = \overline{cl_{j+1}(\mathcal{A}^j)}$ for $j \geq 0$.
- $\mathcal{A}^* = \overline{\cup_{j \geq 0} \mathcal{A}^j}$

Intuitively, $\mathcal{A}^j$ is the set of inference expressions derived by the first j strata of the rulebase. Using $\mathcal{A}^j$ as a starting point, $\mathcal{A}^{j+1}$ is generated by applying the rules in the $j+1^{st}$ stratum until saturation, after which the closed world assumption is applied.

Definition 5.6. Let DB be a database, and let ϕ be a goal. Then $R, DB \vdash \phi$ if and only if the expression $R, DB \vdash \phi$ is in $\mathcal{A}^*$.

This completes the definition of the proof theory of stratified hypothetical rulebases.

Example 5.3. [Inference with Negation] Suppose that R is the rulebase consisting of the single rule $A \leftarrow \sim B[add : C]$, and that $B \notin DB$. Then $R, DB \vdash A$. This can be proved by a straightforward, top-down argument:

$$\begin{array}{ll} & R, DB \vdash A \\ \mathit{if} & R, DB \vdash \sim B[add : C] \\ \mathit{if} & R, DB \not\vdash B[add : C] \\ \mathit{if} & R, DB + C \not\vdash B \end{array}$$

But the last line is trivially true, since there are no rules in R that infer B, and $B \notin DB + C$.

5.3. A Special Case

In the case of stratified Horn rules, the semantics developed above reduce to the semantics of Apt, Blair and Walker [4] and of Przymusinski [55]. It is worth noting, however, that these semantics differ from the semantics of Prolog in a special case: for rules of the form $A \leftarrow \sim B(X)$. Because of the possibility of floundering, Prolog gives such rules a special interpretation. This section discusses this interpretation, and argues that by introducing intermediate predicates, our semantics can be forced to interpret rules as Prolog does, and vice versa.

As in the negation-free case (Section 3.1), the proof theory of stratified hypothetical rulebases implies that the variables of a rule are universally quantified. Furthermore, if a variable does not appear in the head of a rule, then the universal quantifier can be moved inside the rule body and converted to an existential. For example, in the rule $A \leftarrow B(Y), \sim C(Y)$, the variable Y does not appear in the head. This rule can thus be read in two equivalent ways:

$$\forall Y\ [A\ \leftarrow\ B(Y),\ \sim C(Y)]$$
$$A\ \leftarrow\ \exists Y\ [B(Y),\ \sim C(Y)]$$

Note that Prolog interprets this rule in the same two ways. In this respect, our semantics treats Horn rules with negation in the same way that classical logic programming does.

However, our semantics, like those of [4] and [55], differ from the semantics of Prolog in one respect: in Prolog, rules such as $A \leftarrow \sim C(Y)$ are given a special interpretation. In such rules, the value of the variable Y is unguarded, that is, it is not constrained by a positive literal, such as the literal $B(Y)$ in the previous example. Unguardedness can cause the Prolog interpreter to flounder. For this reason, Prolog does *not* interpret this rule in either of the following ways:

$$\begin{aligned} &\forall Y\ [A \leftarrow\ \sim C(Y)] \\ &A \leftarrow \exists Y \sim C(Y) \end{aligned} \tag{5.1}$$

Instead, Prolog interprets it this way:

$$A \leftarrow\ \sim \exists Y\ C(Y) \tag{5.2}$$

That is, the existential quantifier is brought inside the negation sign.

This attempt by Prolog to avoid floundering is only partially successful, however. Both in Prolog and in our semantics, we can effectively obtain either interpretation (5.1) or interpretation (5.2) by introducing intermediate predicates. For instance, by introducing a unary predicate $E(Y)$, the following two rules effectively give us interpretation (5.1):

$$A \leftarrow E(Y) \qquad\qquad E(Y) \leftarrow\ \sim C(Y)$$

This is true both in Prolog and in our semantics. Furthermore, in both systems, these rules cause floundering during top-down inference. Likewise, by introducing a zero-ary predicate D, the following two rules effectively give us interpretation (5.2):

$$A \leftarrow\ \sim D \qquad\qquad D \leftarrow C(Y)$$

This is true both in Prolog and in our semantics. Furthermore, in both systems, these rules do not cause floundering during top-down inference. Prolog's special interpretation of unguarded rules is therefore a convenience that does not provide any additional expressive power and does not eliminate floundering.

Conceptually, then, the proof-theoretic semantics described in Section 5.2 provides a uniform interpretation of stratified rulebases. In the special case of stratified Horn rulebases, this semantics is equivalent to others in the literature [4, 55]. In the very special case of unguarded Horn rules, this semantics can be translated into the semantics of Prolog in a straightforward way.

5.4. Examples

With negation-as-failure and hypothetical updates, it is possible to write simple rulebases that solve complex problems, as the examples of this section illustrate. In the first two examples, the rulebase finds Hamiltonian paths in a directed graph. Since this is an NP-complete problem, these examples provide a simple demonstration of why the data complexity of hypothetical reasoning is not in P (assuming P $\neq$ NP). We then revisit the circuit-validity problem of Section 3.4, and show how with negation-as-failure, the linear order on input lines can be eliminated. In each example, the predicate $SOMELEFT$ acts as an "intermediate predicate" as described in Section 5.3. Additional examples can be found in [10, 11].

Example 5.4. [Hamiltonian Path, I] Suppose that DB is a database representing a directed graph. That is, $NODE(a) \in DB$ if and only if a is a node in the graph, and $EDGE(a,b) \in DB$ if and only if there is an edge in the graph from a to b. Let R be the following collection of rules:

$$
\begin{aligned}
YES &\leftarrow NODE(X),\ PATH(X)[del : NODE(X)].\\
PATH(X) &\leftarrow EDGE(X,Y),\ NODE(Y),\ PATH(Y)[del : NODE(Y)].\\
PATH(X) &\leftarrow\ \sim SOMELEFT.\\
SOMELEFT &\leftarrow NODE(X).
\end{aligned}
$$

Then $R, DB \vdash YES$ if and only if the graph represented by DB has a directed Hamiltonian path.

The rulebase in Example 5.4 is best understood from the perspective of top-down inference. During inference, the rulebase tries to construct a Hamiltonian path one node at a time. The first rule selects a node, x, at which the path is to begin. The second rule is then applied repeatedly, selecting a node, y, connected by an edge to the last node in the path. Each time a node is selected, it is removed from the graph. In this way, no node is selected twice. The last two rules say that a Hamiltonian path has been found when there are no more nodes left in the graph, that is, when every node has been visited exactly once. Note that each node selection is non-deterministic, so in effect, the rulebase searches the graph for all possible Hamiltonian paths.

The Hamiltonian path problem can also be expressed with hypothetical insertions. The idea is to mark nodes instead of deleting them. To do this, after selecting node x, we hypothetically add the formula $MARK(x)$ to the database. In this way, we record which nodes have been selected. We then select unmarked nodes instead of undeleted nodes. This process simulates hypothetical deletion through a combination of insertion and negation-as-failure, as described in Section 2.5. The following example illustrates the idea.

Example 5.5. [Hamiltonian Path, II] Let R be the following collection of rules:

$$
\begin{aligned}
YES &\leftarrow SELECT(X),\ PATH(X)[add : MARK(X)]\\
PATH(X) &\leftarrow EDGE(X,Y),\ SELECT(Y),\ PATH(Y)[add : MARK(Y)]\\
PATH(X) &\leftarrow\ \sim SOMELEFT.\\
SOMELEFT &\leftarrow SELECT(X).\\
SELECT(X) &\leftarrow NODE(X),\ \sim MARK(X).
\end{aligned}
$$

Then $R, DB \vdash YES$ if and only if the graph represented by DB has a directed Hamiltonian path.

In [10, 11], we showed that stratified hypothetical rules could express all database queries in EXPTIME. Unlike comparable expressibility results in the literature, hypothetical rules do not need the artificial assumption of a linearly ordered data domain. The next example illustrates why this is possible. It is a variation on the example developed in Section 3.4. In that example, the database DB encoded a circuit whose input lines were linearly ordered. The linear order was used by the rulebase to select the input wires one at a time. We now present the same example, but without the assumption of a linear order. Instead, we assume that the input wires are simply marked as such. That is, if a is a constant symbol denoting an input wire, then the database contains the atomic formula $INPUT(a)$. Eliminating the linear order is made possible by the combination of hypothetical updates and negation-as-failure.

The main problem is to write rules that select the input wires one at a time. We do this using the techniques of Example 5.5. First, each time a wire is selected, we mark it so that we do not select it again. Second, we use negation-as-failure to determine when there are no more unmarked wires left. The rules below do exactly this, marking each wire with a 0 or a 1 as it is selected. We add these six rules to the four Horn rules of Section 3.4 that define the predicate $EVAL$.

$$\begin{aligned} VALID &\leftarrow SELECT(X),\ VALID[add: ONE(X)], \\ &\qquad VALID[add: ZERO(X)] \\ VALID &\leftarrow\ \sim SOMELEFT,\ EVAL. \\ SOMELEFT &\leftarrow SELECT(X). \\ SELECT(X) &\leftarrow INPUT(X),\ \sim MARKED(X). \\ MARKED(X) &\leftarrow ONE(X). \\ MARKED(X) &\leftarrow ZERO(X). \end{aligned}$$

These rules repeatedly select an input wire and assign it a value of 0 or 1. The process is similar that of Section 3.4 except that now the order in which wires are selected is non-deterministic. This must be the case, since now there is no linear order on the wires, and no way for the rulebase to prefer one ordering over another. After input values have been assigned in this way, the second rule invokes the predicate $EVAL$. $EVAL$ is true if and only if the output of the circuit evaluates to 1, as described in Section 3.4. If $EVAL$ is inferred for every possible set of input values, then the predicate $VALID$ is inferred, and only then. These rules therefore form a rulebase R such that $R, DB \vdash VALID$ if and only if the circuit encoded in DB has an output of 1 for every set of inputs.

Using the above examples, we can compare the semantics of negation developed here to other proposals in the literature. As discussed earlier, such proposals are limited to hypothetical insertions, so we can compare them only in this special case. In general, these proposals are not restricted to stratified rulebases, although other syntactic restrictions are often imposed, either in the model theory or in the proof theory. Unfortunately, these restrictions often preclude many stratified rulebases, including many of the examples in this section. For example, in [38], Harland develops a semantics for hypothetical insertions with negation-as-failure. Example 5.5 above highlights the differences between Harland's treatment and our own. Harland's approach does not depend on stratification, and assigns a meaning even to the rule: $A \leftarrow\ \sim A$. On the other hand, Harland's semantics cannot handle the rulebase of Example 5.5. This is because Harland distinguishes between "completely defined" predicates, such as $APPEND$, and "incompletely defined" predicates, such as $CARCINOGEN$. Negation-as-failure can be applied only to completely defined predicates, while predicates inserted into the database must be incompletely defined. However, in Example 5.5, negation-as-failure is applied to the predicate $MARK$ in the fifth rule, and yet $MARK$ is also inserted into the database in the first two rules. This dual role for the predicate $MARK$ is crucial to the example. A similar observation applies to the rulebase encodings of Turing machines used in proving the expressibility results in [10, 9, 7, 18, 11, 12]. It does not appear that these rulebases would be handled correctly by Harland's semantics. Note, however, that the rules in Example 5.1 cause no such problems. This suggests that it may be useful to develop distinct semantic theories for distinct species of negation. We take up this issue again in Section 7.

5.5. Basic Properties

This section shows that the inference relation $\vdash$ of Definition 5.6 satisfies the inference rules of Definition 3.3. This means that the basic properties of hypothetical inference are unchanged by the presence of negated premises. One can also say that stratified inference as a whole behaves like inference within a single stratum. This property simplifies the proof of soundness and completeness given in Section 6.2. To establish this property, we first state some straightforward results about stratified inference. To simplify the presentation, we assume throughout this section that a rulebase R and a stratification $R_1, R_2, R_3, ...$ are given.

Lemma 5.2. Let ϕ be a positive goal. If the expression $R, DB \vdash \phi$ is in $\mathcal{A}^j$, then $\phi \in DB$ or ϕ belongs to some stratum $i \leq j$.

Lemma 5.3. Let ϕ be a positive goal. Then

- *$R, DB \vdash \sim \phi$ is in $\mathcal{A}^j$ iff $R, DB \vdash \phi$ is not in $\mathcal{A}^j$, for any superscript j.*
- *$R, DB \vdash \phi$ is in $\mathcal{A}^*$ iff $R, DB \vdash \phi$ is in $\mathcal{A}^j$, if ϕ belongs to stratum j.*
- *$R, DB \vdash \sim \phi$ is in $\mathcal{A}^*$ iff $R, DB \vdash \phi$ is not in $\mathcal{A}^j$ for any integer j.*

Lemma 5.4. Let ϕ be a goal belonging to stratum i. If $R, DB \vdash \phi$ then the expression $R, DB \vdash \phi$ is in $cl_j(\mathcal{A}^{j-1})$ for all $j \geq i$.

PROOF. We consider two cases.

Case (i): Suppose ϕ is a negative goal. Then $\phi = \sim \psi$, where ψ is a positive goal. Thus, the expression $R, DB \vdash \sim \psi$ is in $\mathcal{A}^*$, so by Lemma 5.3, the expression $R, DB \vdash \psi$ is not in $\mathcal{A}^n$ for any n. In particular, it is not in $\mathcal{A}^{j-1}$ for all $j \geq i$. Thus, the expression $R, DB \vdash \sim \psi$ is in $\mathcal{A}^{j-1}$, and so it is in $cl_j(\mathcal{A}^{j-1})$ since cl_j is an inflationary mapping.

Case (ii): Suppose ϕ is a positive goal. The proof is an easy induction on j. For the base case, observe that the expression $R, DB \vdash \phi$ is in $\mathcal{A}^i = \overline{cl_i(\mathcal{A}^{i-1})}$ by Lemma 5.3, so it is in $cl_i(\mathcal{A}^{i-1})$ since ϕ is positive. For the inductive step, suppose the expression is in $cl_j(\mathcal{A}^{j-1})$. Then it is also in $\overline{cl_j(\mathcal{A}^{j-1})} = \mathcal{A}^j$ since ϕ is positive. Thus, it is in $cl_{j+1}(\mathcal{A}^j)$ since cl_{j+1} is an inflationary mapping.

□

Theorem 5.5. Inference expressions for R have the following properties, for any database DB:

1. *If $A \leftarrow \phi_1, ..., \phi_k \in inst(R)$ and $R, DB \vdash \phi_i$ for each i, then $R, DB \vdash A$.*
2. *If $R, DB + \{B_1, ..., B_k\} \vdash A$ then $R, DB \vdash A[add : B_1, ..., B_k]$.*
3. *If $R, DB - \{B_1, ..., B_k\} \vdash A$ then $R, DB \vdash A[del : B_1, ..., B_k]$.*
4. *$R, DB \vdash \sim \phi$ iff $R, DB \not\vdash \phi$, for any positive goal ϕ.*

PROOF. We consider each item above in turn.

1. Let j be the stratum containing the rule $A \leftarrow \phi_1, ..., \phi_k$. Since rulebase R is stratified, each ϕ_i belongs to some stratum $j_i \leq j$. Suppose that $R, DB \vdash \phi_i$

for each i. Then the expressions $R, DB \vdash \phi_i$ are in the set $cl_j(\mathcal{A}^{j-1})$ by Lemma 5.4. This set is closed under the inference rules of Definition 5.3. Thus, the expression $R, DB \vdash A$ is also in $cl_j(\mathcal{A}^{j-1})$, so it is in $\mathcal{A}^j$, and therefore in $\mathcal{A}^*$. Hence $R, DB \vdash A$.

2. Suppose A belongs to stratum j. If $R, DB + \{B_1, ..., B_k\} \vdash A$ then the expression $R, DB + \{B_1, ..., B_k\} \vdash A$ is in $\mathcal{A}^*$, and so by Lemma 5.3, it is in $\mathcal{A}^j = \overline{cl_j(\mathcal{A}^{j-1})}$. It is therefore in the set $cl_j(\mathcal{A}^{j-1})$, since A is positive. This set is closed under the inference rules of Definition 5.3. Thus, by inference rule 2, the expression $R, DB \vdash A[add : B_1, ..., B_k]$ is also in $cl_j(\mathcal{A}^{j-1})$, so it is in $\mathcal{A}^j$, and therefore in $\mathcal{A}^*$. Hence $R, DB \vdash A[add : B_1, ..., B_k]$.
3. Parallel to the proof of item 2.
4. Follows immediately from Definitions 5.4–5.6.

□

6. PERFECT MODEL SEMANTICS

Section 5.2 provided a proof-theoretic development of hypothetical inference with negation-as-failure. This section provides a model-theoretic development.

In classical Horn logic programming, the model theory is truly classical only in the negation-free case. When negation is introduced, new devices are needed to account for the non-monotonicity of the resulting system. A common approach, due to Przymusinski, is to introduce a preference relation on the classical models of a rulebase, and then to focus on the most preferred, or *perfect*, models [55]. We extend this idea to hypothetical rulebases and hypothetical structures. The first step is to define what it means for a hypothetical structure to be a model of a rule with negated premises. It is a simple matter to extend the definitions of Section 4.1 to include such rules. For instance, the definitions of hypothetical structures and models remain unchanged. All that is required is to extend the notion of satisfaction to include negative goals by adding the following item to Definition 4.3, for any positive goal ϕ:

$$M, DB \models \sim \phi \quad \textit{iff} \quad M, DB \not\models \phi \tag{6.1}$$

6.1. *Perfect Models*

This section extends the perfect model semantics of Horn rulebases to stratified hypothetical rulebases. We first define a preference relation on the models of a rulebase, and then define logical entailment in terms of models that are minimal with respect to this relation. The preference relation itself depends on how the rulebase is stratified. The whole process amounts to a prioritized version of the closed world assumption in which atoms belonging to lower strata have higher priority. The first step is to partition sets of atoms into subsets based on the strata to which they belong. To simplify the presentation, we assume throughout this section that a particular rulebase R and a particular stratification $R_1, R_2, R_3, ...$ are given.

Definition 6.1. [Projection] Let G be a set of ground atomic formulas. Then $G|_j$ is the set of atoms in G that belong to the j^{th} stratum.

Using this notation, we define a preference relation on hypothetical structures. For each hypothetical structure M and each database DB, the preference relation is based on the atoms in the set $M(DB)$ and on the strata they belong to. We first look at the atoms belonging to stratum 1, then the atoms belonging to stratum 2, etc. Intuitively, if two structures first differ at stratum j, then we prefer the structure with the fewest atoms in that stratum.

Definition 6.2. [Preference] Let M_1 and M_2 be two hypothetical structures. Then

- $M_1 \preceq_i M_2$ *iff* $M_1(DB)|_i \subseteq M_2(DB)|_i$ for all databases DB.
- $M_1 =_i M_2$ *iff* $M_1 \preceq_i M_2$ and $M_2 \preceq_i M_1$.
- $M_1 \preceq M_2$ *iff* for each stratum j,
  if $M_1 =_i M_2$ for all $1 \leq i < j$, then $M_1 \preceq_j M_2$.

Note that $\preceq$ is transitive and reflexive, and that $M_1 = M_2$ if and only if $M_1 \preceq M_2$ and $M_2 \preceq M_1$. If $M_1 \preceq M_2$ then we say that M_1 is *preferable* to M_2. The following result relating to preference is straightforward.

Lemma 6.1. Suppose that $M_1 =_i M_2$, and that ϕ is a goal belonging to the i^{th} stratum. Then $M_1, DB \models \phi$ if and only if $M_2, DB \models \phi$, for any database DB.

As in the perfect model semantics of stratified Horn rulebases, we focus on the minimal elements of the preference relation.

Definition 6.3. [Preferred Models] Let $\mathcal{M}_R$ be the set of models of R. A preferred model of R is a minimal element of $\mathcal{M}_R$, *i.e.*, minimal with respect to the preference relation $\preceq$.

The following definition completes our generalization of the classical, perfect-model semantics.

Definition 6.4. [Preferred Entailment] Let ϕ be a goal. Then $R, DB \models \phi$ if and only if $M, DB \models \phi$ for all preferred models M of R.

6.2. Soundness and Completeness

This section shows that the proof theory developed in Section 5.2 is sound and complete for the model theory in Section 6.1. Unlike most such proofs, where soundness and completeness are proved separately, the presence of negation requires us to prove them simultaneously. This is because soundness at one stratum relies on completeness at the stratum below, and vice versa. The proof itself builds upon the properties of inference established in Section 5.5, especially Theorem 5.5. Central to our proof is the notion of a *canonical model*. This model, which is defined proof theoretically, provides the necessary link between the proof theory and the model theory. As we shall see, the canonical model is the unique preferred model of a stratified rulebase. Preferred entailment thus amounts to satisfaction in this model. Establishing uniqueness is the main step in our proof. To simplify the presentation, we assume throughout this section that a particular rulebase R and a particular stratification $R_1, R_2, R_3, \ldots$ are given.

Definition 6.5. The canonical model of R is the hypothetical structure M_R where $M_R(DB) = \{A \mid R, DB \vdash A\}$.

Observe that M_R is inflationary, so it is indeed a hypothetical structure. The following lemma and its corollary establish two fundamental properties of the canonical model.

Lemma 6.2. $R, DB \vdash \phi$ *iff* $M_R, DB \models \phi$, *for any goal* ϕ *and any database* DB.

PROOF. There are four cases, depending on the form of ϕ.

1. If ϕ is atomic, then $M_R, DB \models \phi$ *iff* $\phi \in M_R(DB)$ *iff* $R, DB \vdash \phi$ by definition.

2. If $\phi = A[add : B_1, ..., B_k]$, then

$$\begin{array}{lll} & M_R, DB \models A[add : B_1, ..., B_k] & \\ \textit{iff} & M_R, DB + \{B_1, ..., B_k\} \models A & \text{by Definition 4.3,} \\ \textit{iff} & R, DB + \{B_1, ..., B_k\} \vdash A & \text{by case 1,} \\ \textit{iff} & R, DB \vdash A[add : B_1, ..., B_k] & \text{by Theorem 5.5.} \end{array}$$

3. If $\phi = A[del : B_1, ..., B_k]$, then

$$\begin{array}{lll} & M_R, DB \models A[del : B_1, ..., B_k] & \\ \textit{iff} & M_R, DB - \{B_1, ..., B_k\} \models A & \text{by Definition 4.3,} \\ \textit{iff} & R, DB - \{B_1, ..., B_k\} \vdash A & \text{by case 1,} \\ \textit{iff} & R, DB \vdash A[del : B_1, ..., B_k] & \text{by Theorem 5.5.} \end{array}$$

4. If $\phi = \sim \psi$, where ψ is a positive goal, then ψ matches case 1, 2 or 3, above. Hence,

$$\begin{array}{lll} & M_R, DB \models \sim \psi & \\ \textit{iff} & M_R, DB \not\models \sim \psi & \text{by statement (6.1),} \\ \textit{iff} & R, DB \not\vdash \psi & \text{by case 1, 2 or 3,} \\ \textit{iff} & R, DB \vdash \sim \psi & \text{by Theorem 5.5.} \end{array}$$

□

Corollary 6.3. The canonical model M_R *is a model of* R.

PROOF. Let $A \leftarrow \phi_1, ..., \phi_m$ be a ground instance of a rule in R. Then, for any database DB,

$$\begin{array}{lll} \textit{if} & M_R, DB \models \phi_1, ..., \phi_m & \\ \textit{then} & M_R, DB \models \phi_i & \text{for each } i, \\ & R, DB \vdash \phi_i & \text{for each } i, \text{ by Lemma 6.2,} \\ & R, DB \vdash A & \text{by Theorem 5.5,} \\ & M_R, DB \models A & \text{by Lemma 6.2.} \end{array}$$

This is true for every ground instance of every rule in R. Thus, M_R is a model of R by Definition 4.5. □

Lemma 6.5 below is the main step in showing that the canonical model M_R is the unique preferred model of the stratified rulebase R. Before proving this lemma, it is convenient to give the following simple result.

Lemma 6.4. Let M be a hypothetical structure such that $M_R =_i M$ for all $1 \leq i < j$. Then for any database DB and any goal ϕ below the j^{th} stratum, $M_R, DB \models \phi$ if and only if $M, DB \models \phi$.

PROOF. If ϕ is below the j^{th} stratum, then ϕ belongs to some stratum $i < j$. $M_R =_i M$ by hypothesis, so $M_R, DB \models \phi$ if and only if $M, DB \models \phi$ by Lemma 6.1. $\Box$

Lemma 6.5. Let M be a model of R such that $M_R =_i M$ for all $1 \leq i < j$. Then for any database DB, and any positive goal ϕ belonging to the j^{th} stratum,

if $M_R, DB \models \phi$ *then* $M, DB \models \phi$

PROOF. Since $M_R, DB \models \phi$, it follows by Lemma 6.2 that $R, DB \vdash \phi$. Because ϕ belongs to the j^{th} stratum, the inference expression $R, DB \vdash \phi$ must be in the set $\mathcal{A}^j$ by Lemma 5.3. In fact, since ϕ is positive, it must be in the set $cl_j(\mathcal{A}^{j-1})$ by Definition 5.4. It is therefore sufficient to prove the following statement:

$$\textit{if} \quad R, DB \vdash \phi \in cl_j(\mathcal{A}^{j-1}) \quad \textit{then} \quad M, DB \models \phi \tag{6.2}$$

If the expression $R, DB \vdash \phi$ is in $cl_j(\mathcal{A}^{j-1})$, then the expression is derivable by the inference system of Definition 5.3 using the set $\mathcal{A}^{j-1}$ as axioms. We prove statement (6.2) by induction on the length of this derivation.

Basis: Suppose the expression $R, DB \vdash \phi$ is derivable in 0 steps. Then the expression is an axiom. Thus, by Definition 5.3, either $\phi \in DB$ or the expression $R, DB \vdash \phi$ is in the set $\mathcal{A}^{j-1}$. Thus, if $\phi \notin DB$, then ϕ must belong to some stratum $i \leq j-1$ by Lemma 5.2, which is impossible, since we are given that ϕ belongs to stratum j. Thus $\phi \in DB$, so $\phi \in M(DB)$ by Definition 4.2, so $M, DB \models \phi$ by Definition 4.3.

Induction: Suppose that statement (6.2) is true for all inference expressions derivable in at most n steps, and suppose the expression $R, DB \vdash \phi$ is derivable in $n+1$ steps. We must show that $M, DB \models \phi$. There are three cases, depending on the form of ϕ.

Case (i): Suppose ϕ is atomic. Then the j^{th} stratum of R has a rule with a ground instantiation of the form $\phi \leftarrow \phi_1, ..., \phi_k$. Furthermore, for each i, the expression $R, DB \vdash \phi_i$ is derivable in at most n steps. Since the rulebase R is stratified, each ϕ_i is at or below the j^{th} stratum. If ϕ_i is *below* the j^{th} stratum, then $M, DB \models \phi_i$ by Lemma 6.4. On the other hand, if ϕ_i is *at* the j^{th} stratum, then ϕ_i must be a positive goal, since otherwise the rulebase would not be stratified. Thus $M, DB \models \phi_i$ by induction hypothesis. Thus $M, DB \models \phi_i$ for each premise of the rule $\phi \leftarrow \phi_1, ..., \phi_k$. Thus $M, DB \models \phi$ by Definition 4.5, since M is a model of R, and thus a model of the rule.

Case (ii): Suppose $\phi = A[add : B_1, ..., B_m]$. Then

the expression $R, DB \vdash A[add : B_1, ..., B_m]$ is derivable in $n+1$ steps,
so the expression $R, DB + \{B_1, ..., B_m\} \vdash A$ is derivable in n steps,
so $M, DB + \{B_1, ..., B_m\} \models A$ by induction hypothesis,
so $M, DB \models A[add : B_1, ..., B_m]$ by Definition 4.3,
so $M, DB \models \phi$.

Case (iii): Suppose $\phi = A[del : B_1, ..., B_m]$. Then

the expression $R, DB \vdash A[del : B_1, ..., B_m]$ is derivable in $n+1$ steps,
so the expression $R, DB - \{B_1, ..., B_m\} \vdash A$ is derivable in n steps,
so $M, DB - \{B_1, ..., B_m\} \models A$ by induction hypothesis,
so $M, DB \models A[del : B_1, ..., B_m]$ by Definition 4.3,
so $M, DB \models \phi$.

□

Corollary 6.6. If M is a model of R, then $M_R \preceq M$.

PROOF. Referring to Definition 6.2, suppose that $M_R =_i M$ for all $1 \leq i < j$. We must show that $M_R \preceq_j M$. To see this, let A be a ground atomic formula belonging to the j^{th} stratum. If $M_R, DB \models A$ then $M, DB \models A$ by Lemma 6.5. Thus, if $A \in M_R(DB)$ then $A \in M(DB)$, for any database, DB, and any atom, A, in the j^{th} stratum. Thus $M_R(DB)|_j \subseteq M(DB)|_j$ for all DB. Thus $M_R \preceq_j M$ by Definition 6.2. □

Corollary 6.7 Unique Preferred Model. A stratified rulebase, R, has a unique preferred model, and it is the canonical model, M_R.

PROOF. M_R is a model of R by Corollary 6.3. It is a minimal model by Corollary 6.6, and hence it is a preferred model by Definition 6.3. To see that M_R is is unique, let M be any other preferred model. Then $M_R \preceq M$ by Corollary 6.6. Hence, $M_R = M$, since M is preferred, and thus minimal. □

Theorem 6.8 Soundness and Completeness.
$R, DB \vdash \phi$ iff $R, DB \models \phi$, for any database DB and any goal ϕ.

PROOF. Since M_R is the unique preferred model of R, it follows from Definition 6.4 that $R, DB \models \phi$ if and only if $M_R, DB \models \phi$. The result then follows by Lemma 6.2. □

7. RELATED WORK

There has been considerable work in the area of logic and updates. The logical systems that are the closest to Hypothetical Datalog are Dynamic Prolog, developed by Manchanda and Warren [45, 44, 68], and EKS, developed by Vieille, et al [66, 67]. All three systems deal with both insertion and deletion, and in all three, updates take place on base facts, not derived facts. Nevertheless, the three systems are very different. In EKS, the semantics is purely operational. The emphasis here has been on implementation, not on logical foundations. In Dynamic Prolog, the semantics is based on Dynamic Logic [37]. Unlike Hypothetical Datalog, in which all predicates and rules have equal status, Dynamic Prolog distinguishes many types of predicates and rules. Both EKS and Dynamic Prolog emphasize committed updates as well as hypothetical updates, while work on Hypothetical Datalog includes a logical semantics for negation-as-failure and an extensive analysis of computational complexity and expressibility [10, 9, 7, 18, 11, 12].

Much of the remaining work on logic and updates can be divided into three classes: (i) elementary updates to arbitrary logical theories, (ii) complex updates to relational databases, and (*iii*) intuitionistic logic programming. We shall look at each class in turn.

The first class of work includes [69, 42, 28, 27, 56]. This work focuses on the semantics of elementary updates to complex theories. Our work is distinct from this class in two ways. First, we are concerned with combining elementary updates into complex hypothetical queries. Second, we are concerned with relational databases (sets of ground atomic formulas). In this context, elementary updates pose no conceptual problems, as can be seen from Definitions 3.3 and 4.3. In particular, the infamous *frame problem* of Artificial Intelligence does not arise [46, 56]. It should be possible to extend Hypothetical Datalog to include any set of elementary updates, not just insertion and deletion, and any set of databases, not just relational. In this sense, our work is orthogonal to work on the logical semantics of elementary updates. This idea has been adopted in the development of Transaction Logic [14, 13, 16, 15].

The second class of work includes [2, 1, 24, 23, 54]. Like Hypothetical Datalog, these systems are primarily concerned with relational databases, and with combining elementary updates into complex updates. However, these systems deal with *committed* updates, not with *hypothetical* queries. Because of their treatment of deletion, these systems are *non-monotonic*, even in the absence of negation. The result is that although these systems have an operational semantics, they typically lack a model theory and a logical inference system. Other differences are reflected in the semantics of negation and in computational complexity. For instance, languages presented in [2] have an inflationary semantics and a PSPACE complexity, whereas Hypothetical Datalog has a perfect-model semantics and an EXPTIME complexity.

The third class of work is perhaps the most similar to our own. However, it is concerned exclusively with hypothetical insertion. This is because hypothetical insertion fits neatly into a well-known logical system: *intuitionistic logic* [29]. Gabbay was the first to show that intuitionistic logic models hypothetical insertion [32]. Working independently, McCarty and Miller extended this result to operations that create new constant symbols during inference, and they developed fixpoint semantics based on intuitionistic logic [47, 50]. Earlier, Statman showed that inference in intuitionistic logic is PSPACE-complete in the propositional case [59]. In [10, 11], we showed that the complexity of answering queries with hypothetical insertions is PSPACE-complete in the function-free predicate case (*i.e.*, the databases case).

Researchers have also investigated the semantics of negation-as-failure for intuitionistic logic programming. Gabbay was the first to point out that negation-as-failure leads to logical paradoxes in an intuitionistic setting [32]. To overcome these paradoxes, some researchers distinguish two types of implication, denoted $\leftarrow$ and $\Leftarrow$ [17, 52, 36]. The former corresponds to our own use of implication, and the latter corresponds to our operator of hypothetical insertion. In fact, the rule $A \leftarrow (B \Leftarrow C)$ is functionally identical to $A \leftarrow B[add : C]$. One difference between the two approaches is that intuitionistic logic can insert not just atomic facts, but entire rules. In general, the rule $A \leftarrow (B \Leftarrow \psi)$ hypothetically inserts the formula ψ into the rulebase during inference, where ψ can be another rule. However, this capability does not add any power or expressiveness to the logic, since the same thing can be achieved by introducing extra predicate symbols. For instance, the rule $A \leftarrow [B \Leftarrow (C \leftarrow D)]$ hypothetically inserts $C \leftarrow D$ into the rulebase during inference. But this effect can also be achieved by the two rules $A \leftarrow B[add : E]$ and

$C \leftarrow D, E$. The first rule inserts the atom E into the database, which effectively "enables" the rule $C \leftarrow D$.

In [17], Bonner and McCarty develop a semantics of stratified intuitionistic rulebases, including a model theory and a sound-and-complete proof theory. In the special case in which all the rules are Horn, this semantics is equivalent to that of stratified Horn rulebases [4]. The present paper extends and simplifies the results in [17] to include hypothetical deletion as well as insertion. Results on the computational complexity and expressibility of this semantics are presented in [10, 9, 7, 18, 11, 12].

There have also been proposals for a semantics of non-stratified intuitionistic rules [38, 52, 36, 26]. Following [50], these proposals each define a model as a mapping from a set of logic programs to a set of values. Unlike the semantics presented in this paper, these proposals often provide a semantics of *finite* failure. Thus, in the special case of Horn rulebases, these works provide an alternative to much of the existing work on the semantics of negation [4, 55, 35, 34, 64]. One exception is the work of Dung [26], which proposes a stable model semantics for hypothetical insertion. Although no proof theory is presented, Dung's proposal allows arbitrary rulebases, stratified or unstratified. In the special case of Horn rulebases, it is equivalent to the stable model semantics developed in [35]. In [52], Olivetti and Terracini present a three-valued modal semantics for the propositional case, and in [36], Giordano and Olivetti develop a three-valued fixpoint semantics for the predicate case. In both cases, the semantics is of finite failure in arbitrary rulebases. [36] also develops a practical SLD-style proof procedure. Under a syntactic restriction called *allowedness*, this procedure is sound and avoids floundering. Unfortunately, this restriction is rather conservative, and it disallows many programs that are not really problematic, including many non-recursive programs. A detailed comparison of their work and our semantics of stratified intuitionistic rules [17] can be found in [36]. In [38], Harland develops a semantics in which predicates are labeled as "completely defined" or "incompletely defined" [38]. Unfortunately, restrictions on the labeling preclude many stratified programs, such as the rulebase in Example 5.5, and the rulebases used to achieve the expressibility results of [10, 9, 7, 18, 11, 12], as discussed in Section 5.4. The rules in Example 5.1 cause no such problems however. This suggests that it may be useful to develop distinct semantic theories for distinct species of negation.

Acknowledgements. The work of Thorne McCarty on the intuitionistic semantics of embedded implications was the original stimulus for this work. Discussions with Tomasz Imielinski were invaluable in giving the work a database perspective. Jan Chomicki, Ron van der Meyden and Kumar Vadaparty also provided valuable feedback during the development of the work presented herein. Some of the work presented here builds on ideas originally developed in collaboration with Thorne McCarty. This is mostly true of the work on negation-as-failure, which is an extension and simplification of our earlier work on the semantics of negation-as-failure in intuitionistic logic programming [17].

REFERENCES

1. S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences (JCSS)*, 41:181–229, 1990.

2. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences (JCSS)*, 43:62–124, 1991.
3. A.V. Aho and J.D. Ullman. Universality of Data Retrieval Languages. In *Proceedings of the ACM Symposium on the Principles of Programming Languages (POPL)*, pages 110–120, San Antonio, TX, Jan. 1979.
4. K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.
5. K.R. Apt and M.H. Van Emden. Contributions to the Theory of Logic Programming. *Journal of the ACM*, 29(3):841–862, 1982.
6. F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems (TODS)*, 6:557–575, 1981.
7. A.J. Bonner. Intuitionistic deductive databases and the polynomial time hierarchy. Submitted for publication.
8. 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. American Association for Artificial Intelligence (AAAI).
9. 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 29–31 1989.
10. A.J. Bonner. Hypothetical Datalog: Complexity and Expressibility. *Theoretical Computer Science (TCS)*, 76:3–51, 1990. Special issue on the 2^{nd} International Conference on Database Theory (ICDT'88).
11. A.J. Bonner. *Hypothetical Reasoning in Deductive Databases.* PhD thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, USA, October 1991. Published as Rutgers Technical Report DCS-TR-283.
12. A.J. Bonner. Hypothetical Reasoning with Intuitionistic Logic. In R. Demolombe and T. Imielinski, editors, *Non-Standard Queries and Answers*, Studies on Logic and Computation, chapter 8, pages 187–219. Oxford University Press, October 1994.
13. A.J. Bonner and M. Kifer. Transaction Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 257–279, Budapest, Hungary, June 21–24 1993. MIT Press.
14. A.J. Bonner and M. Kifer. An Overview of Transaction Logic. *Theoretical Computer Science (TCS)*, 133, October 1994.
15. A.J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP)*, Bonn, Germany, September 1996. To appear.
16. A.J. Bonner, M. Kifer, and M. Consens. Database programming in transaction logic. In *Proceedings of the International Workshop on Database Programming Languages (DBPL)*, Workshops in Computing, pages 309–337. Springer Verlag, 30 Aug–1 Sept 1993.
17. A.J. Bonner and L.T. McCarty. Adding Negation-as-Failure to Intuitionistic Logic Programming. In *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 681–703, Austin, Texas, Oct 29–Nov 1 1990. MIT Press.
18. A.J. Bonner, L.T. McCarty, and K. Vadaparty. Expressing Database Queries with Intuitionistic Logic. In *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 831–850, Cleveland, Ohio, October 16–20 1989. MIT Press.
19. A.K. Chandra. Theory of Database Queries. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 1–9, Austin, Texas, March 1988.
20. A.K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal*

of Computer and System Sciences (JCSS), 21(2):156–178, 1980.
21. A.K. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences (JCSS)*, 25(1):99–128, 1982.
22. A.K. Chandra and D. Harel. Horn Clause Queries and Generalizations. *Journal of Logic Programming (JLP)*, 2(1):1–15, 1985.
23. W. Chen. Declarative specification and evaluation of database updates. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases (DOOD)*, volume 566 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, December 1991.
24. W. Chen. Declarative updates of relational databases. *ACM Transactions on Database Systems (TODS)*, 20(1):42–70, 1995.
25. U. Dayal and P.A. Bernstein. Translation of update operations on relational views. *ACM Transactions on Database Systems (TODS)*, 8:381–416, 1982.
26. P.M. Dung. Declarative Semantics of Hypothetical Logic Programming with Negation as Failure. In *Proceedings of the Third International Workshop on Extensions of Logic Programming*, pages 45–58, Bologna, Italy, February 1992. Springer-Verlag. Published as volume 660 of *Lecture Notes in Artificial Intelligence*, 1993.
27. R. Fagin, G.M. Kuper, J.D. Ullman, and M.Y. Vardi. Updating logical databases. In P.C. Kanellakis, editor, *Advances in Computing Research*, volume 3, pages 1–18. Plenum Press, 1986.
28. R. Fagin, J.D. Ullman, and M.Y. Vardi. On the semantics of updates in databases. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 352–365, 1983.
29. M.C. Fitting. *Intuitionistic Logic, Model Theory and Forcing*. North-Holland, 1969.
30. The Committee for Advanced DBMS function. Third-Generation Database System Manifesto. *SIGMOD Record*, 19(3):31–44, September 1990. Also published as Memorandum No. UCB/ERL M90/28, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.
31. A.L. Furtado and K.C. Sevcik. Permitting updates through views of databases. *Information Systems*, 4:269–283, 1979.
32. 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.
33. 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.
34. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.
35. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080. MIT Press, 1988.
36. L. Giordano and N. Olivetti. Negation as Failure in Intuitionistic Logic Programming. In *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP)*, pages 431–445, Washington, D.C., 1992. MIT Press.
37. D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
38. J. Harland. A Kripke-like model for negation as failure. In *Proceedings of the North American Conference on Logic Programming (NACLP)*, pages 626–642, Cleveland, Ohio, October 1989. MIT Press.
39. Richard Hull and Jianwen Su. On the Expressive Power of Database Queries with Intermediate Types. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 39–51, Austin, Texas, March 1988.
40. N. Immerman. Relational Queries Computable in Polynomial Time. In *Proceedings*

of the ACM Symposium on Theory of Computing (STOC), pages 147–152, 1982.
41. R.H. Sprague Jr. and H.J. Watson, editors. *Decisions Support Systems: Putting Theory into Practice*. Prentice Hall, Englewood Cliffs, NJ, 1989.
42. H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge database and revising it. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pages 387–394, Boston, Mass., April 1991.
43. A.M. Keller. Updates to relational databases through views involving joins. In P. Scheuermann, editor, *Improving Database Usability and Responsiveness*, pages 363–384. Academic Press, 1982.
44. S. Manchanda and D.S. Warren. A Logic-based Language for Database Updates. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 10, pages 363–394. Morgan Kaufmann, 1988.
45. 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.
46. J.M. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969. Reprinted in *Readings in Artificial Intelligence*, 1981, Tioga Publ. Co.
47. L.T. McCarty. Clausal Intuitionistic Logic. I. Fixed-Point Semantics. *Journal of Logic Programming (JLP)*, 5(1):1–31, 1988.
48. L.T. McCarty. Clausal Intuitionistic Logic. II. Tableau Proof Procedures. *Journal of Logic Programming (JLP)*, 5(2):93–132, 1988.
49. 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.
50. D. Miller. A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming (JLP)*, 6:79–108, 1989.
51. D. Miller. Lexical scoping as universal quantification. In G. Levi and M. Martelli, editors, *Logic Programming: Proceedings of the Sixth International Conference*, pages 268–283, Cambridge, MA, 1989. MIT Press.
52. N. Olivetti and L. Terracini. N-Prolog and Equivalence of Logic Programs (Part 1). *Journal Of Logic, Language and Information*, 1(4):253–340, 1992.
53. R.L. Olson and R.H. Sprague Jr. Financial Planning in Action. In R.H. Sprague Jr. and H.J. Watson, editors, *Decisions Support Systems: Putting Theory into Practice*, pages 373–381. Prentice Hall, Englewood Cliffs, NJ, 1989.
54. G. Phipps, M.A. Derr, and K.A. Ross. Glue-Nail: A deductive database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 308–317, 1991.
55. T. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.
56. R. Reiter. On formalizing database updates: Preliminary report. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, March 1992.
57. D.A. Schlobohm and L.T. McCarty. EPS-II: Estate planning with prototypes. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 1–10. ACM Press, June 1989.
58. Editor: A.P. Sheth. Database Research at Bellcore. *SIGMOD Record*, 19(3):45–52, September 1990.

59. R. Statman. Intuitionistic Propositional Logic is Polynomial-Space Complete. *Theoretical Computer Science (TCS)*, 9(1):67–72, 1979.
60. M. Stonebraker. Hypothetical databases as views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 224–229, 1981.
61. M. Stonebraker and K. Keller. Embedding expert knowledge and hypothetical databases into a data base system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 58–66, Santa Monica, CA, 1980.
62. A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
63. M.H. Van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.
64. A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 221–230, March 1988.
65. M. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.
66. L. Vieille, P. Bayer, V. Kuchenhoff, and A. Lefebvre. EKS-V1, A Short Overview. Presented at the AAAI-90 Workshop on Knowledge Base Management Systems, July 1990, Boston, USA.
67. L. Vieille, P. Bayer, V. Kuchenhoff, A. Lefebvre, and R. Manthey. The EKS-V1 system. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, number 624 in Lecture Notes in Computer Science, pages 504–506. Springer-Verlag, 1992.
68. D.S. Warren. Database Updates in Pure Prolog. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 244–253, 1984.
69. M. Winslett. A model based approach to updating databases with incomplete information. *ACM Transactions on Database Systems (TODS)*, 13(2):167–196, 1988.