Extensional Reasoning

Timothy L. Hinrichs and Michael R. Genesereth
Stanford University
{thinrich,genesereth}@cs.stanford.edu

Abstract

Relational databases have had great industrial success in computer science, their power evidenced by theoretical analysis and widespread adoption. Often, automated theorem provers do not take advantage of database query engines and therefore do not routinely leverage that source of power. Extensional Reasoning is a novel approach to automated theorem proving where the system automatically translates a logical entailment query into a query about a database system so that the answers to the two queries are the same. This paper discusses the framework for Extensional Reasoning, describes algorithms that enable a theorem prover to leverage the power of the database in the case of axiomatically complete theories, and discusses theory resolution for handling incomplete theories.

1 Introduction

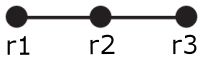
Relational database systems have been built to answer questions about enormous amounts of data, yet it is rare today for a theorem prover (Cyc's inference engine [MG03] a notable exception) to exploit those systems when confronted with large theories. Extensional Reasoning (ER) is an approach to automated theorem proving where the system transforms a logical entailment query into a database (a set of extensionally defined tables), a set of view definitions (a set of intensionally defined tables), and a database query.

For example, the map coloring problem is often stated as follows: given a map and a set of colors, paint each region of the map so that no two adjacent regions are the same color. One very natural way to encode this problem (according to the introductory logic students at Stanford who offer up this formulation year after year) centers around the following constraints.

$$color(x, y) \Rightarrow region(x) \wedge hue(y)$$
$$color(x, y) \wedge adj(x, z) \Rightarrow \neg color(z, y)$$

These constraints are the same regardless the map and the colors, i.e. regardless the definitions for *region*, *hue*, and *adj*; moreover, the definitions for these three predicates are often complete. For example, for the graph illustrated in Fig. 1, we would have the following definitions.

$$region(x) \Leftrightarrow (x = r_1 \vee x = r_2 \vee x = r_3)$$
$$hue(x) \Leftrightarrow (x = red \vee x = blue)$$
$$adj(x, y) \Leftrightarrow ((x = r_1 \wedge y = r_2) \vee (x = r_2 \wedge y = r_3))$$



Query: $next(x, y) \wedge next(y, z)$

Database:

next	
red	blue
blue	red

Figure 1: A 3-node graph and its corresponding database query

It so happens that the database formulation of this problem studied in, for example, [McC82] is quite different from the one above and is shown in Fig. 1. Given a table, *next*, that contains all the possible pairs of adjacent colors, the query for the three region graph would be

$$next(x, y) \wedge next(y, z)$$

where x represents the hue for r_1, y the hue for r_2, and z the hue for r_3.

The key difference between the formulations is that the database version entails all the valid colorings, whereas the classical version is only consistent with each valid coloring individually. In Extensional Reasoning, the transformation from one formulation into the other happens automatically.

A relational database corresponds to a special kind of logical theory—one that is axiomatically complete. Such theories are rare, and even when the theory is complete, recognizing that fact and constructing the appropriate database is nontrivial. Since the logical formulation of map coloring comprises an incomplete theory, to transform it into the database formulation we must first complete it. Because theory-completion in the context of Extensional Reasoning is performed solely for the sake of efficiency, a theory must be completed so that any entailment query about the original theory can be transformed into an entailment query about the completed theory where the answers to the queries are the same, another nontrivial problem.

Sometimes the work is worth the effort. A database can sometimes solve the database version of the query orders of magnitude faster than traditional techniques solve the classical version; moreover, the cost of a database query is bounded (polynomial in the size of the data and exponential in the size of the query), making the performance more predictable, sometimes an important feature from the users' perspective. These benefits appear to have several causes. First, the portion of the logical theory that is represented extensionally can be indexed very efficiently both for lookup and retrieval. Second, negation is treated with Negation as Failure, which can cause large reductions in the search space. Finally, if there is a solution to a problem, the candidates can be checked one at a time, which is in contrast to the theorem proving environment where disjunctive answers would require checking each subset of possible answers.

While we hope Extensional Reasoning will eventually be applied to a wide variety of logics, for the time being we have elected to focus on theories in a decidable logic, placing the issue of efficiency front and center. The particular logic we are studying is a fragment of first-order logic that is a perennial problem in the theorem proving community: it includes the domain closure axiom, which guarantees decidability while allowing arbitrary quantification. This logic, to which the example above belongs, allows

us to avoid issues of undecidability at this early stage in the development of Extensional Reasoning, while at the same time giving us the opportunity to make progress on an important class of problems.

Orthogonal to the choice of logic, Extensional Reasoning could be applied in a variety of settings that differ in the way efficiency is measured. Our work thus far measures efficiency the same way the theorem proving community does. Once the machine has been given a premise set and a query the clock starts; the clock stops once the machine returns an answer. We do not amortize reformulation costs over multiple queries, and we do not assume the premises or query have ever been seen before or will ever be seen again.

The bulk of this paper examines Extensional Reasoning in the case of complete theories and introduces algorithms for recognizing completeness and transforming a complete theory into a database system. For incomplete theories, some results on theory resolution are introduced. Further work on the incomplete case can be found in a companion paper [HG07]. Our technical contributions occur in the sections on Complete theories (3) and Incomplete theories (4). The necessary background is given in Sect. 2.

2 Background

In our investigations of Extensional Reasoning thus far, the logic we have considered is function-free first-order logic with equality, unique names axioms (UNA), and a domain closure axiom (DCA). The UNA state that every pair of distinct object constants in the vocabulary is unequal. The DCA states that every object in the universe must be one of the object constants in the vocabulary. Together, the UNA and DCA ensure that the only models that satisfy a given set of sentences are the Herbrand models of those sentences, and because the language is function-free, every such model has a finite universe. We call this logic Finite Herbrand Logic (FHL). It is noteworthy that entailment in FHL is decidable.

Besides the existence of UNA and a DCA, the definitions for FHL are the same as those for function-free first-order logic with equality. Terms and sentences are defined as usual. We say a sentence is *closed* whenever it has no free variables, and *open* whenever it has at least one free variable. The definition for a model is the usual one, but because all the models of FHL are isomorphic to finite Herbrand models, it is often convenient to treat a model as a set of ground atoms. When we do that, satisfaction is defined as follows.

Definition 1 (FHL Satisfaction). *The definition for the satisfaction of closed sentences, where the model M is represented as a set of ground atoms, is as follows.*

$\models_M s = t$ *if and only if* s *and* t *are syntactically identical.*

$\models_M p(t_1, \ldots, t_n)$ *if and only if* $p(t_1, \ldots, t_n) \in M$

$\models_M \neg\psi$ *if and only if* $\not\models_M \psi$

$\models_M \psi_1 \wedge \psi_2$ *if and only if* $\models_M \psi_1$ *and* $\models_M \psi_2$

$\models_M \forall x.\psi(x)$ *if and only if* $\models_M \psi(a)$ *for every object constant* a.

An open sentence $\psi(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$ is satisfied by M if and only if $\forall x_1 \ldots x_n.\psi(x_1, \ldots, x_n)$ is satisfied by M according to the above definition.

A set of sentences in FHL constitutes an incomplete theory whenever there is more than one Herbrand model that satisfies it. A set of sentences in FHL constitutes a complete theory whenever there is at most one Herbrand model that satisfies it. Logical entailment is defined as usual: $\Delta \models \phi$ in FHL if and only if every model that satisfies Δ also satisfies ϕ.

The language used in this paper for representing database systems is nonrecursive datalog with negation, denoted *datalog*$^\neg$, which is equivalent in expressive power to relational algebra and SQL. This particular language has been chosen because it is well-understood and because some of the algorithms for processing it are very similar to algorithms for processing classical logic, which allows for relatively clean comparisons.

A *datalog*$^\neg$ system consists of (1) a set of tables, named using the Extensional Database predicates (EDB predicates) and (2) a set of datalog rules whose heads use the Intensional Database predicates (IDB predicates). The EDB predicates and IDB predicates are disjoint. A datalog rule is an implication, where $:-$ is used in place of $\Leftarrow$.

$$h :- b_1 \wedge \cdots \wedge b_n$$

h, the head of the rule, is an atomic sentence. Each b_i is a literal, and the conjunction of b_is is called the body of the rule. Every rule must be safe: every variable that occurs in the head or in a negative literal must occur in a positive literal in the body. Collectively, the rule set must be non-recursive, which can be defined in terms of the rules' dependency graph. The dependency graph for a rule set consists of one node per predicate and an edge from u to v exactly when there is a rule with u in the head and v in the body, labelled with $\neg$ if v occurs in a negative literal. To be nonrecursive, the dependency graph for a rule set must be acyclic.

A model for a *datalog*$^\neg$ system is the same as that for FHL: a set of ground atoms. We use the standard stratified semantics. A consequence of these definitions is that every *datalog*$^\neg$ system is satisfied by exactly one Herbrand model, or more precisely, a database system is a representation of a single model. A database query is true exactly when the query is true in that model, written $\Gamma \cup \Lambda \models \phi$, where Γ consists of the extensional tables and Λ the view definitions. When confusion may arise, we will subscript $\models$ with FHL and D to indicate the semantics for FHL and *datalog*$^\neg$, respectively.

Because every *datalog*$^\neg$ system represents a single model, the logical theory corresponding to a database system is one that is axiomatically complete.

Definition 2 (Axiomatic Completeness). *A satisfiable set of sentences Δ is axiomatically complete with respect to language L if and only if for every closed sentence s in L, either $\Delta \models s$ or $\Delta \models \neg s$. Δ is complete with respect to vocabulary V if and only if it is complete with respect to the set of all first-order sentences that can be formed from V.*

3 Complete Theories

A satisfiable set of sentences in Finite Herbrand Logic can always be transformed into $datalog^{\neg}$ while preserving logical equivalence if those sentences constitute a complete theory. In this section, we discuss algorithms for recognizing that a set of sentences is complete and algorithms for transforming such a sentence set into a $datalog^{\neg}$ system.

3.1 Recognizing Complete Theories

In Finite Herbrand Logic, a satisfiable, complete theory is satisfied by exactly one Herbrand model—by exactly one set of ground literals. Entailment in FHL is decidable because there is a fixed, finite bound on the size of the universe. Consequently, checking whether a satisfiable FHL theory is complete is decidable. For each ground atom a in the language (of which there are finitely many), check whether Δ entails a or Δ entails $\neg a$. If for some a, neither is entailed, the theory is incomplete; otherwise, the theory is complete.

This decision algorithm relies on entailment queries to determine whether the theory is complete, but entailment is the very problem Extensional Reasoning is meant to confront. For this reason we developed an alternate algorithm that has more of a syntactic flavor—one that performs some inexpensive tests that are sufficient for ensuring completeness.

Complete theories have been studied in the nonmonotonic reasoning literature. Predicate completion, for example, was one of the early techniques used to define the semantics of Negation as Failure in Logic Programming [Llo84]. When applied to a set of nonrecursive rules, predicate completion produces a set of nonrecursive biconditional sentences, e.g.

$$\begin{aligned} p(x) &\Leftrightarrow (q(x) \wedge r(x)) \\ q(x) &\Leftrightarrow (x = a \vee x = b) \\ r(x) &\Leftrightarrow (x = e \vee x = f) \end{aligned} \tag{1}$$

With some small caveats, a nonrecursive set of biconditional definitions is guaranteed to comprise a complete theory.

Definition 3 (Biconditional Definition). *A biconditional definition is a sentence of the form $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$, where r is a relation constant, $\overline{x}$ is a sequence of non-repeating variables, and $\phi(\overline{x})$ is a sentence with free variables $\overline{x}$.*

Definition 4 (Nonrecursive biconditional definitions). *A set of biconditional definitions Δ is nonrecursive if and only if the dependency graph $\langle V, E\rangle$ is acyclic.*

V : the set of predicates in Δ

E : $\langle u, v\rangle$ is a directed edge if and only if there is a biconditional in Δ with u in the head and v in the body.

Theorem 1 (Biconditional Completeness). *Suppose Δ is a finite set of nonrecursive, biconditional definitions in FHL, where there is exactly one definition for each predicate in Δ and no definition for $=$. Δ is complete.*

In this paper we examine algorithms that attempt to reformulate a sentence set into a logically-equivalent set of nonrecursive biconditional definitions. Though incomplete, these algorithms run in low-order polynomial time, making them practical for certain classes of theories. These algorithms perform the recognition task by partitioning the sentence set and examining each partition individually. If each partition can be written as a single biconditional then by the way the partitions are chosen, the theory is guaranteed to be complete.

To partition the sentences, we ask the following question. Suppose the sentence set were originally written as nonrecursive, biconditional definitions. Further suppose that each definition were then rewritten independently of all the others without introducing any additional predicates while preserving logical equivalence. For each predicate p, how do we find all those sentences that were produced from the biconditional definition for p?

For example, the following set of clauses were produced by converting the nonrecursive, biconditional definitions for p, q, and r in Formula 1 into clausal form.

$$
\begin{array}{l}
p(x) \vee \neg q(x) \vee \neg r(x) \\
\neg p(x) \vee q(x) \\
\neg p(x) \vee r(x) \\
q(x) \vee x \neq a \\
q(x) \vee x \neq b \\
\neg q(x) \vee x = a \vee x = b \\
r(x) \vee x \neq e \\
r(x) \vee x \neq f \\
\neg r(x) \vee x = e \vee x = f
\end{array}
$$

How does one determine which clause came from which definition?

Because the original biconditionals were nonrecursive, there must be at least one of them whose body is expressed entirely in terms of equality; call each such biconditional a base table definition. All those clauses that contain one predicate besides equality must have originated in base table definitions. Within that set of clauses, all those that constrain the same predicate must have come from the definition for that predicate.

In the example above, applying this observation selects two sets of sentences.

$$
\begin{array}{l}
q(x) \vee x \neq a \\
q(x) \vee x \neq b \\
\neg q(x) \vee x = a \vee x = b
\end{array}
$$

and

$$
\begin{array}{l}
r(x) \vee x \neq e \\
r(x) \vee x \neq f \\
\neg r(x) \vee x = e \vee x = f
\end{array}
$$

The first is the set of sentences that came from the definition for q, and the second is the set of sentences that came from the definition for r.

The partitioning algorithm recurses on the remaining sentences, this time looking for sentences with a single predicate besides $=$ and the base table predicates. At iteration i,

the sentences that originated from definitions for predicates $p_1, \ldots, p_j$ have been found and the algorithm looks for sentences with a single predicate besides $\{p_1, \ldots, p_j, =\}$. This is a variation on the context-free grammar (CFG) marking algorithm: when the partition is found for predicate p, all occurrences of p in the remaining sentences are marked, and when a sentence has all but one of its predicates marked, it is added to the partition corresponding to the remaining predicate.

In the example, the remaining sentences are

$$p(x) \vee \neg q(x) \vee \neg r(x)$$
$$\neg p(x) \vee q(x)$$
$$\neg p(x) \vee r(x)$$

and since each contains one predicate, p, besides the predicates $\{q, r, =\}$, all of these sentences are grouped into the partition corresponding to the biconditional for p.

Alg. 1 embodies the approach outlined here. Each recursive call selects all those sentences with the same unmarked predicate p and calls REFORMULATE-TO-BICOND on those sentences. If they entail a biconditional for p, that biconditional is deemed to be the definition for p, and the algorithm recurses. If no such biconditional is entailed, the algorithm finds a different unmarked predicate and tries again. If there is no biconditional entailed for any of the unmarked predicates, the theory is incomplete.

Algorithm 1 TO-BICONDS(Δ, *basepreds*)

1: $sents := \{\langle p, d\rangle | d \in \Delta$ and $p \notin basepreds$ and $Preds[d]$ is p plus a subset of $basepreds\}$
2: $preds := \{p | \langle p, d\rangle \in sents\}$
3: $bicond :=$ NIL
4: **for all** $p \in preds$ **do**
5: $\quad partition := \{d | \langle p, d\rangle \in sents\}$
6: $\quad bicond :=$ REFORMULATE-TO-BICOND($partition$, p)
7: $\quad pred := p$
8: $\quad$ **when** $bicond$ **then** exit **for all**
9: **end for**
10: **when** $not(bicond)$ **then return** NIL
11: $remaining := \Delta - partition$
12: **when** $remaining = \emptyset$ **then return** $cons(bicond, \text{NIL})$
13: $rest :=$ TO-BICONDS($remaining$, $basepreds \cup \{pred\}$)
14: **when** $rest$ **then return** $cons(bicond, rest)$
15: **return** NIL

TO-BICONDS runs in $O(n^2(m + r))$, where n is the number of predicates in Δ, m is the size of Δ, and r is the cost of REFORMULATE-TO-BICOND. Under the conditions mentioned above, TO-BICONDS is sound and complete as long as REFORMULATE-TO-BICOND is sound and complete.

Theorem 2 (Soundness of TO-BICONDS). *Under the conditions listed below, if* TO-BICONDS($\Delta, \{=\}$) *returns the non-NIL* Γ*, then* Γ *is a set of nonrecursive biconditionals with one definition per predicate in* Δ *besides equality and* Δ *is logically equivalent to* Γ*.*

- Δ *is a satisfiable sentence set in FHL*
- REFORMULATE-TO-BICOND *is sound, i.e. if* REFORMULATE-TO-BICOND(Σ, p) *returns* $p(\overline{x}) \Leftrightarrow \phi(\overline{x})$ *then the result is a biconditional definition for* p *entailed by* Σ.

Theorem 3 (Completeness of TO-BICONDS). *Under the conditions listed below, if* Δ *is a complete theory in FHL then* TO-BICONDS$(\Delta, \{=\})$ *does not return NIL.*

- Δ *is a satisfiable, nonempty sentence set that can be partitioned into sets so that there is one set per relation constant in* Δ *besides equality:* $S_{r_1}, \ldots, S_{r_n}$.
- Δ *is logically equivalent to a set of nonrecursive biconditionals with one definition per relation constant besides equality:* $\{\beta_{r_1}, \ldots, \beta_{r_n}\}$
- β_{r_i} *is logically equivalent to* S_{r_i} *and* $Preds[\beta_{r_i}] = Preds[S_{r_i}]$*, for every* i
- REFORMULATE-TO-BICOND *is complete, i.e. if* Σ *entails some nonrecursive biconditional definition* $p(\overline{x}) \Leftrightarrow \phi(\overline{x})$ *then* REFORMULATE-TO-BICOND(Σ, p) *returns an equivalent biconditional definition for* p*; otherwise, it returns NIL.*

TO-BICONDS relies on REFORMULATE-TO-BICOND, an algorithm for checking whether a given set of sentences entails a biconditional definition. There are many such algorithms, which form a spectrum of inexpensive syntactic checks to expensive semantic checks. Our approach lies closer to the inexpensive end of the spectrum: it attempts to rewrite the sentences of a partition into sufficient conditions for p, $p(\overline{x}) \Leftarrow \phi(\overline{x})$, and necessary conditions for p, $p(\overline{x}) \Rightarrow \psi(\overline{x})$. It then tests whether $\phi(\overline{x})$ and $\psi(\overline{x})$ are logically equivalent, and again there is a spectrum of algorithms for checking logical equivalence.

To finish the example, the sentences of the last partition can be written as

$$p(x) \Leftarrow (q(x) \wedge r(x))$$
$$p(x) \Rightarrow (q(x) \wedge r(x))$$

Clearly there is a biconditional definition entailed by these two impications; likewise for the other partitions in the example. Notice that because of the way the partitions were constructed, the resulting set of biconditionals must be nonrecursive.

$$p(x) \Leftrightarrow (q(x) \wedge r(x))$$
$$q(x) \Leftrightarrow (x = a \vee x = b)$$
$$r(x) \Leftrightarrow (x = e \vee x = f)$$

3.2 Translating Complete Theories into Datalog

One of the benefits of the recognition algorithm described in the last section is that when successful, the algorithm produces a formulation of the theory, a set of nonrecursive biconditionals, that is amenable to translation into *datalog*$^{\neg}$; that transformation is the subject of this section.

The translation takes place in two steps. First, the set of nonrecursive biconditional definitions is partitioned into the portion that is turned into data (extensional tables)

and the portion that is turned into views (intensional tables). Second, the extensional portion is converted into explicit tables and the intensional portion is converted into *datalog*$^\neg$ view definitions.

The approach reported in this paper for partitioning the set of biconditionals is a simple one. As mentioned in the last section, every set of biconditional definitions must include at least one definition whose body is expressed entirely in terms of equality. Our partitioning algorithm assigns all such definitions to the extensional portion of the theory and all the remaining definitions to the intensional portion. More sophisticated partitioning algorithms are the subject of future work.

Regardless of what algorithm is used to split the set of biconditional definitions into the extensional and the intensional portions, the algorithms for transforming those biconditionals into *datalog*$^\neg$ are fairly straightforward.

Turning a biconditional definition into a *datalog*$^\neg$ view definition can be performed using a typical recursive walk of the sentence and a post-processing step at the end. The walk turns $\forall$ into $\neg\exists\neg$; it introduces new predicates when negation and $\exists$ are encountered; boolean connectives are treated as usual. We demonstrate by showing a step-by-step transformation of $r(x) \Leftrightarrow \forall y.(\neg p(y) \vee q(x,y))$.

$r(x) \Leftrightarrow \forall y.(\neg p(y) \vee q(x,y))$
$r(x) :- \forall y.(\neg p(y) \vee q(x,y))$ ($\Leftrightarrow$ turned into $:-$)
$r(x) :- \neg\exists y.\neg(\neg p(y) \vee q(x,y))$ ($\forall$ turned into $\neg\exists\neg$)
$r(x) :- \neg\exists y.(p(y) \wedge \neg q(x,y))$ ($\neg$ pushed through)
(1) $r(x) :- not(newreln(x))$ (new predicate invented)
where $newreln(x) \Leftrightarrow \exists y.(p(y) \wedge \neg q(x,y))$
$newreln(x) :- \exists y.p(y) \wedge \neg q(x,y)$ ($\Leftrightarrow$ turned into $:-$)
(2) $newreln(x) :- p(y) \wedge \neg q(x,y)$ (drop $\exists$)

The result is two rules (1) and (2), one for r and one for $newreln$. Notice that neither rule is safe—there are variables in negative literals that do not occur in positive literals. To be valid *datalog*$^\neg$, safety must be ensured. To achieve this, we introduce a new predicate $univ$ that is true of all the object constants in the vocabulary, i.e. all the objects in the universe. Then we add $univ(x)$ for every variable x that is not safe in some rule. The result is shown below.

$$r(x) :- not(newreln(x)) \wedge univ(x)$$
$$newreln(x) :- p(y) \wedge not(q(x,y)) \wedge univ(x)$$

Because the biconditional definitions are nonrecursive, the algorithm illustrated above, which we will refer to as VIEWS-TO-DATALOG, is guaranteed to produce a set of safe, stratified *datalog*$^\neg$ rules. The algorithm used for converting the body of the rule will convert any sentence into *datalog*$^\neg$; it will be called SENTENCE-TO-DATALOG.

VIEWS-TO-DATALOG can also be used to convert a biconditional into a table of data. Suppose we convert all the biconditionals, both those destined to become data and those destined to become views, into *datalog*$^\neg$ views using VIEWS-TO-DATALOG, treating $=$ as an uninterpreted predicate. We can then add in a table for $=$, which has one row per element in the universe. Then we can materialize a table by constructing the appropriate *datalog*$^\neg$ query and running it through a standard deductive database engine. In the case of very large theories, here again, we are relying on database algorithms to do what

they do best—manipulate large amounts of data. We will refer to the algorithm for converting a biconditional into an extensional table with the name MATERIALIZE.

The partitioning algorithm, the algorithm for converting a sentence into *datalog*$^\neg$, and the algorithm for transforming biconditionals into views and data are bundled together into BICONDS-TO-DATALOG, an algorithm for converting an entailment query in FHL about a set of nonrecursive biconditionals into *datalog*$^\neg$.

Algorithm 2 BICONDS-TO-DATALOG(Δ, ϕ)

Assumes: Δ is a set of nonrecursive biconditionals and ϕ is in the language of Δ

1: $(D, V) :=$ PARTITION(Δ)
2: $\Gamma :=$ MATERIALIZE($Preds[D]$, Δ)
3: $\Lambda :=$ VIEWS-TO-DATALOG(V)
4: $\psi :=$ SENTENCE-TO-DATALOG(ϕ)
5: **return** (Γ, Λ, ψ)

BICONDS-TO-DATALOG ensures that Δ entails ϕ under FHL semantics if and only if $\Gamma \cup \Lambda$ entails ψ under Datalog semantics. BICONDS-TO-DATALOG ensures something much stronger as well: Δ and $\Gamma \cup \Lambda$ have all the same models (under their respective semantics). This ensures all the logical consequences are exactly the same; the transformation preserves not only entailment of the query in question but of all queries in the language.

Theorem 4 (Equivalence Preservation of BICONDS-TO-DATALOG). *Let $\Delta \models_{FHL} \phi$ be a logical entailment query where Δ is a set of nonrecursive biconditionals with one definition per predicate besides equality, and let ϕ be in the language of Δ. Suppose* BICONDS-TO-DATALOG *produces (Γ, Λ, ψ). $\Delta \models_{FHL} \phi$ if and only if $\Gamma \cup \Lambda \models_D \psi$, under the following assumptions.*

- $\psi =$ SENTENCE-TO-DATALOG(ϕ)
- *If Δ is a set of nonrecursive biconditionals, then $(D, V) = Partition(\Delta)$ is a partitioning of Δ.*
- *If D is a set of nonrecursive biconditionals for predicates $\{r_1, \ldots, r_n\}$, then $\Gamma =$* MATERIALIZE*$(Preds[D], \Delta)$ consists of a table for each r_i, i.e. $\overline{a}$ is a tuple in the r_i table in Γ if and only if $\Delta \models r_i(\overline{a})$.*
- *If V is a set of nonrecursive biconditionals, then $\Lambda =$* VIEWS-TO-DATALOG*(V) is a nonrecursive, stratified set of datalog$^\neg$ rules such that for every set E of nonrecursive biconditional definitions for Preds[D],*

$$E \cup V \models_{FHL} \phi \text{ if and only if } \text{MATERIALIZE}(Preds[D], E) \cup \Lambda \models_D \psi$$

3.3 Extensional Reasoning

When given a logical entailment query $\Delta \models \phi$, the recognition algorithm TO-BICONDS determines whether Δ is complete and if so uses the transformation algorithm BICONDS-TO-DATALOG to turn the query into *datalog*$^\neg$. Then an algorithm for processing *datalog*$^\neg$ is

employed to answer the query. If Δ is not complete, traditional algorithms are employed to determine whether entailment holds. The following algorithm, ER-ENTAILEDP, relies on DB-ENTAILEDP to answer the *datalog*$^\neg$ query and FHL-ENTAILEDP to answer an arbitrary entailment query in FHL in the case of incomplete theories.

Algorithm 3 ER-ENTAILEDP(Δ, ϕ)

Returns: T if and only if $\Delta \models_{FHL} \phi$

```
1: Σ := TO-BICONDS(Δ)
2: if Σ then
3:    (Γ, Λ, ψ) := BICONDS-TO-DATALOG(Σ,φ)
4:    return DB-ENTAILEDP(Γ, Λ, ψ)
5: else
6:    return FHL-ENTAILEDP(Δ, φ)
7: end if
```

The theorems from previous sections ensure ER-ENTAILEDP is sound and complete.

Theorem 5 (Soundness and Completeness). *Suppose ϕ is a sentence in the language of Δ.* ER-ENTAILEDP(Δ, ϕ) *returns T if and only if $\Delta \models_{FHL} \phi$.*

One of the keys to the proof of this theorem bridges the gap between how the semantics of logic and the semantics of *datalog*$^\neg$ treat negation. Logic uses classical negation whereas *datalog*$^\neg$ uses negation as failure. While NAF is often thought of as a nonmonotonic and therefore unsound inference rule, in the case of complete theories, NAF is sound.

Theorem 6 (Soundness of NAF). *Negation as failure is a sound rule of inference when it is applied to a closed sentence in the language of an axiomatically complete theory while using a complete proof procedure.*

Proof. NAF is a meta inference rule based on a proof system $\vdash$. NAF infers $\neg\phi$ whenever $\Delta \nvdash \phi$. Suppose that ϕ is closed, Δ is axiomatically complete (entails ψ, $\neg\psi$, or both for every closed sentence ψ), and $\vdash$ is complete (finds a proof whenever entailment holds). Then, if $\Delta \nvdash \phi$, by the completeness of $\vdash$, $\Delta \not\models \phi$. By the completeness of Δ, we have $\Delta \models \neg\phi$. Thus, when $\Delta \nvdash \phi$, $\Delta \models \neg\phi$, which is the conclusion NAF produces; thus, it is sound. □

3.4 A Comparison of ER and Traditional Techniques

The central claim of this paper is that sometimes answering an entailment query using algorithms for processing *datalog*$^\neg$ is more efficient than using the typical algorithms for processing FHL. One of the main benefits of *datalog*$^\neg$ is its use of Negation as Failure, which happens to be a sound rule of inference in the case of complete theories. This section compares NAF to traditional treatments of negation from both the theoretical and the empirical perspective.

Theoretical Comparison

To demonstrate the power of NAF, we start by comparing SLDNF resolution and model elimination on an example, a fair comparison since the two proof procedures differ mainly in how they treat negation. SLDNF uses NAF, whereas model elimination uses classical negation. Consider the following biconditional definition for p.

$$p(x) \Leftrightarrow \left(\begin{array}{l} (p_1(x) \wedge p_2(x) \wedge p_3(x)) \vee \\ (p_4(x) \wedge p_5(x) \wedge p_6(x)) \vee \\ (p_7(x) \wedge p_8(x) \wedge p_9(x))) \end{array} \right)$$

Converting the biconditional above to view definitions using the algorithms of the last section basically amounts to dropping the $\Rightarrow$.

$$\begin{array}{lll} p(x) & :- & p_1(x) \wedge p_2(x) \wedge p_3(x) \\ p(x) & :- & p_4(x) \wedge p_5(x) \wedge p_6(x) \\ p(x) & :- & p_7(x) \wedge p_8(x) \wedge p_9(x) \end{array}$$

In contrast, converting the biconditional to clausal form produces the implications above along with implications for $\neg p(x)$. In this example, there are 27 possible ways to prove $\neg p(t)$ for a particular t, which is exponential in the size of the biconditional. The naive clausal form conversion will construct one clause for each one. (Structure-preserving clausal form conversion, which can be found in chapters 5 and 6 of [RV01], ensures the number of clauses is polynomial in the size of the original sentence set, but as we will see the size of the search space can still be exponential.)

$$\begin{array}{c} \neg p(x) \Leftarrow \neg p_1(x) \wedge \neg p_4(x) \wedge \neg p_7(x) \\ \neg p(x) \Leftarrow \neg p_1(x) \wedge \neg p_4(x) \wedge \neg p_8(x) \\ \neg p(x) \Leftarrow \neg p_1(x) \wedge \neg p_4(x) \wedge \neg p_9(x) \\ \vdots \\ \neg p(x) \Leftarrow \neg p_3(x) \wedge \neg p_6(x) \wedge \neg p_9(x) \end{array}$$

We compare SLDNF resolution on the *datalog*$^\neg$ with model elimination on the clauses. For the entailment query $\exists x.p(x)$, the two perform identically (assuming we turn off the reduction operation [AS92] for model elimination); however, for the query $\exists x.\neg p(x)$, the two techniques differ significantly. SLDNF resolution uses the rules with $p(x)$ in the head to look for a t such that the proof for $p(t)$ fails. Model elimination uses the rules with $\neg p(x)$ in the head to look for a t such that $\neg p(t)$ has a proof. Depending on the relative sizes of the universe, the search space for $p(x)$, and the search space for $\neg p(x)$, proving $\neg p(t)$ by exhausting the search space for $p(t)$ can be far less expensive than finding a proof in the search space for $\neg p(t)$.

Information theoretically, it is not surprising that for complete theories, NAF sometimes answers queries more quickly than methods for classical negation. NAF can be viewed as a mechanism for reasoning about complete theories, whereas methods for classical negation are mechanisms for reasoning about incomplete theories. NAF implicitly leverages the fact that the theory is complete, but methods for classical negation do not.

More tangibly, the tradeoff between NAF and classical negation can be seen as a tradeoff of search spaces. Often the space for $\neg p$ is larger than the space for p, and it is this case that NAF takes advantage of. As opposed to classical negation, NAF

avoids searching the large space and instead searches just the small space. (We might additionally consider Truth as Failure to handle the case where the space for p is very large but the space for $\neg p$ is very small.)

As mentioned earlier, structure-preserving clausal-form conversion will avoid enumerating the exponential number of rules with $\neg p$ in the head, but that does not necessarily prevent resolution from enumerating an exponential search space. Moreover, with certain assumptions placed on what it means to convert to clausal form, one can show that regardless of which clausal form conversion is used, there is an infinite class of biconditionals such that resolution has the potential to generate exponentially many clauses in the size of the biconditional, assuming that the implementation of resolution is generatively complete.

Lemma 1. *Consider the following biconditional β.*

$$p(x) \Leftrightarrow \left(\begin{array}{l} (p_{11}(x) \wedge \cdots \wedge p_{1n_1}(x)) \vee \\ \vdots \\ (p_{m1}(x) \wedge \cdots \wedge p_{mn_m}(x)) \end{array} \right)$$

When applying the naive clausal form conversion to β, the number of clauses with a negative p literal is the product of the lengths of the disjunctions: $\prod_i n_i$.

$$\begin{array}{c} \neg p(x) \vee p_{11}(x) \vee p_{21}(x) \vee \cdots \vee p_{m1}(x) \\ \neg p(x) \vee p_{11}(x) \vee p_{21}(x) \vee \cdots \vee p_{m2}(x) \\ \vdots \\ \neg p(x) \vee p_{1n_1}(x) \vee p_{2n_2}(x) \vee \cdots \vee p_{mn_m}(x) \end{array}$$

Suppose a clausal form conversion algorithm converts β into Γ. Further suppose Γ it is logically equivalent to β with respect to the predicates in β, i.e. for every sentence σ whose predicates are a subset of those in β, $\Gamma \models \sigma$ iff $\beta \models \sigma$. Suppose Res is an implementation of resolution that is generatively complete.

$$Res[\Gamma \cup \{p(t)\}] \text{ will produce at least } \prod_i n_i \text{ clauses.}$$

Proof. Here we argue somewhat informally. Consider an arbitrary clause with a negative p literal.

$$\neg p(x) \vee p_{1j_1}(x) \vee p_{2j_2}(x) \vee \cdots \vee p_{mj_m}(x)$$

This clause, which is entailed by β, entails

$$p(t) \Rightarrow p_{1j_1}(t) \vee p_{2j_2}(t) \vee \cdots \vee p_{mj_m}(t)$$

Consequently Γ entails it as well. Thus, $\Gamma \cup \{p(t)\}$ entails

$$p_{1j_1}(t) \vee p_{2j_2}(t) \vee \cdots \vee p_{mj_m}(t)$$

Since *Res* is generatively complete up to subsumption, $Res[\Gamma \cup \{p(t)\}]$ must include either this disjunction, or some clause that subsumes it. No clause that subsumes this one is entailed by $\Gamma \cup \{p(t)\}$; hence, this clause must appear in the closure. Since the clause was chosen arbitrarily, the same holds for all clauses. Since there are $\prod_i n_i$ of these clauses, the resolution closure must contain at least that many clauses. □

The above lemma guarantees a local property about resolution—that given a single biconditional and a query, the resolution closure is exponentially large in the number of disjunctions. When other sentences are included, the lemma makes no guarantees. For example, if resolution were given a biconditional for $p(x)$ along with the sentence $\neg p(x)$, resolution could find a proof without enumerating the exponential number of consequences described above.

To truly understand a technique it is important to find its limitations. Extensional Reasoning is not always superior to traditional techniques. Consider a satisfiable, complete premise set that consists of a single sentence $\forall xy.p(x,y)$, which when converted to clausal form is just $p(x,y)$. The query $\forall xy.p(x,y)$ when negated and converted to clausal form is $\neg p(k_1,k_2)$. Resolution finds a proof in a single step, regardless of how large the universe is.

In Extensional Reasoning, if the decision is made to materialize p, the database includes n^2 tuples for p, where n is the size of the universe. Using SLDNF resolution, the proof attempt $\forall xy.p(x,y)$ is performed by attempting to find elements k_1 and k_2 such that $\neg p(k_1,k_2)$ is true. Since every attempt fails, even assuming perfect indexing, the proof costs n^2.

Thus, in some cases Extensional Reasoning can find proofs in complete theories more efficiently than traditional techniques, but this last example demonstrates that this is not always the case. An important next step is to investigate algorithms that predict which approach will find a proof (or fail to find a proof) more quickly; such algorithms are the subject of future work.

Empirical Comparison

Next we report on experiments designed to compare Extensional Reasoning techniques to traditional techniques in the case of complete theories. Since the algorithms for detecting completeness presented in this paper produce a set of nonrecursive biconditional definitions, all the tests we performed were on such sentence sets. The results demonstrate what is predicted above. When negation occurs in the sentences, the *datalog*$^\neg$ implementation, which uses Negation as Failure, is significantly faster than techniques that do not employ NAF.

Each sentence set represents the layout of a two-dimensional grid, like the one shown in Fig. 2. Every sentence set has exactly six biconditionals representing the following information.

- $west(x,y)$: cell x is the cell immediately west of cell y.
- $north(x,y)$: cell x is the cell located immediately north of cell y.
- $duewest(x,y)$: cell x is in the same row as cell y and located to the west.
- $duenorth(x,y)$: cell x is in the same column as cell y and located to the north.
- $vert(x,y)$: cell x is $duenorth$ of y or vice versa.
- $westof(x,y)$: cell x is located to the west of y, i.e. x and y can be in different rows but x's column is to the west of y's column.

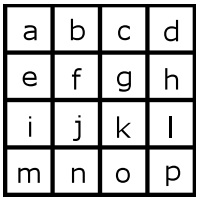


Figure 2: A 4×4 grid, corresponding to the theory in the appendix.

An axiomatization for the 4×4 case can be found in the appendix.

The tests varied the dimensions of the grid, starting at 4×4 and ending at 20×20. The queries were of the form $westof(t, u)$ and $\neg westof(t, u)$, where t and u were always ground. Every query tested was entailed. The results of the positive queries are reported separately from the results of the negative queries because negation plays a prevalent role in the negative queries but not the positive. The positive queries provide a baseline for comparing techniques on how well they handle nonrecursive biconditional definitions. The negative queries illustrate how NAF can affect performance.

We compared our implementation of Extensional Reasoning for the case of complete theories, DBD (Datalog Based Deduction), with Vampire 8.1, Darwin, and Epilog (the Stanford Logic Group's model-elimination implementation[1]). Vampire and Darwin were run using SystemOnTPTP, and Epilog and DBD were run in Macintosh Common Lisp on a 1.5 GHz G4 Powerbook with 1.25 GB of RAM.

As expected, DBD, the system built to reason about towers of biconditionals, outperformed the three general-purpose systems on both the positive and the negative queries. Each system had 1000 seconds to find a proof. Most of the systems performed fairly similarly until the last or second-to-last grid size they solved under the time limit.

For the positive queries (Fig. 3), Epilog and Darwin performed similarly, both failing to find a proof in 1000 seconds for the 16×16 grid. Vampire failed to find a proof at 20×20; DBD solved the 20×20 in 126 seconds. These tests were run without the DCA or the UNA since entailment did not require them; thus, these tests primarily illustrate how the various systems cope with the potentially large number of clauses generated by converting biconditionals to clausal form.

[1]For completeness, model elimination and therefore Epilog require all contrapositives of the clauseset to be explored. For these experiments, only those clauses produced by a typical clausal form conversion were used.

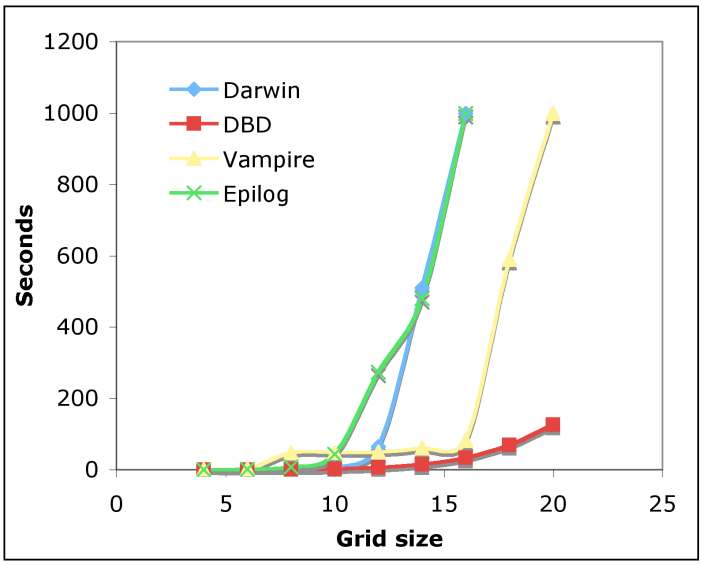


Figure 3: Results for queries of the form $westof(t, u)$.

$n \times n$	Darwin	Vampire	Epilog	DBD
4	0	0	0	0
6	0	3	1	0
8	1	47	7	1
10	7	48	43	2
12	65	50	152	6
14	509	61	480	15
16	> 999	82	> 999	33
18		589		69
20		> 999		126

For the negative queries (Fig. 4), the difference between DBD and the others is larger. Vampire failed at size 8×8; Epilog failed at size 12×12; Darwin failed at 16×16. DBD solved 20×20 in 187 seconds. This difference appears to be due mainly to the fact that DBD uses NAF, and the negative search space for *westof* (as induced by the clauses with negative *westof* literals) is substantially larger than the positive search space. These tests required the UNA, but not the DCA.

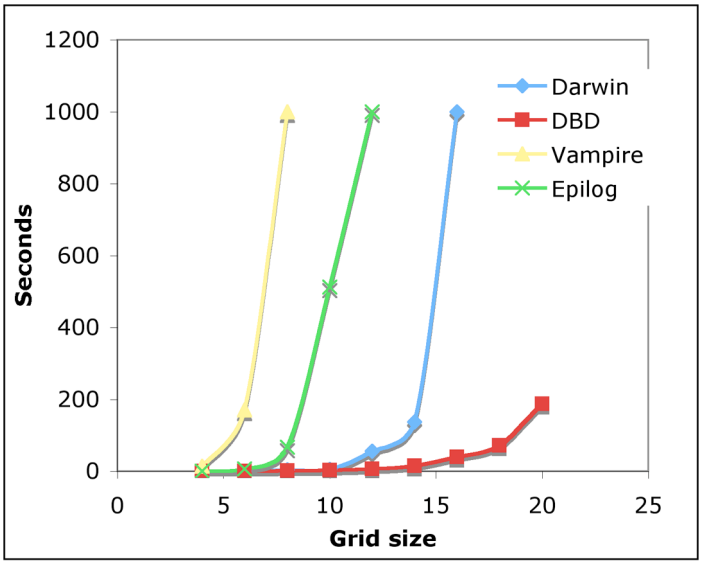


Figure 4: Results for queries of the form $\neg westof(t, u)$.

$n \times n$	Darwin	Vampire	Epilog	DBD
4	0	14	0	0
6	0	170	6	0
8	4	> 999	67	1
10	5		512	2
12	55		> 999	6
14	137			15
16	> 999			40
18				72
20				187

We also worked on answering entailment queries using boolean SAT solvers, since every FHL theory can always be converted into propositional logic. Conversion to clausal form after grounding proved to be prohibitively expensive, even using the structure-preserving version of clausal form conversion. Comparing DBD to a non-clausal SAT solver is the subject of future work.

These experiments compare how quickly various sytems can prove theorems when the search space is shallow but has a high branching factor. Typical database applications have this character: the majority of the cost comes from manipulating large amounts of data, and the tower of view definitions built on top of the extensional tables is relatively short. As expected, in such situations Negation as Failure can produce significant savings over treating negation classically.

4 Incomplete Theories

Complete theories have powerful properties, but incomplete theories are the norm. The last section detailed techniques for reasoning efficiently about complete theories using database techniques. However, if one were to add even just a small amount of incompleteness into a complete theory by including new predicates, those techniques could no longer be applied. This is unfortunate since the speedups in the complete case seem to be large enough to absorb some extra overhead for dealing with theories that have a small amount of incompleteness.

For example, suppose we take any complete theory and add in the following sentences, where p and q are new predicates.

$$p(a) \vee p(b) \vee p(c)$$
$$q(d) \vee q(e) \vee q(f)$$

Supposing the complete theory had a definition for the binary predicate r, a reasonable entailment query might be

$$\forall xy.(p(x) \wedge q(y) \Rightarrow r(x, y)). \tag{2}$$

When the definitions for r are large enough to warrant using a database system, we probably still want to use that database system in spite of the fact that we now have an incomplete theory.

Theory resolution [Sti85] is one approach to dealing with such situations, where part of the theory can be effectively represented and reasoned about with a specialized procedure. If a theory were partitioned into the complete portion C and the incomplete portion I, theory resolution would allow us to use a database system to represent C while answering queries about $C \cup I$ using resolution.

One of the benefits of the TO-BICONDS algorithm shown in Alg. 1 is that with a two-line change, it can be used to find the portion of the theory that is complete, partitioning the theory as required for theory resolution. This change consists of replacing lines (14) and (15) so that once the algorithm fails to find a new biconditional definition, it returns all the biconditionals it has found so far. Alg. 4 gives this new algorithm, TO-BICONDS-MAX. It is noteworthy that TO-BICONDS-MAX can be turned into an anytime algorithm, allowing the user or system designer to determine how much time to spend trying to find a complete subtheory.

With the partitioning algorithm in place, we can now focus on theory resolution. Because C is complete, we can think of it as being a set of ground literals such that every ground atom or its negation is included. For the case where the incomplete portion I is in $\forall^*$, i.e. where when I is written in prenex normal form the quantifiers are all universal, theory resolution takes a particularly simple form. Suppose p is a predicate that belongs to the complete portion, i.e. every ground p literal or its negation belongs to C. Then if there were some clause

$$\{p(\bar{t})\} \cup \Phi,$$

resolution would produce a resolvent for every literal $\neg p(\bar{a})$ where $p(\bar{a})$ unifies with $p(\bar{t})$:

$$\Phi\sigma, \text{ where } \sigma \text{ is the mgu of } p(\bar{a}) \text{ and } p(\bar{t})$$

Algorithm 4 TO-BICONDS-MAX(Δ, *basepreds*)

1: $sents := \{\langle p, d\rangle | d \in \Delta$ and $p \notin basepreds$ and $Preds[d]$ is p and some subset of $basepreds\}$
2: $preds := \{p | \langle p, d\rangle \in sents\}$
3: $bicond :=$ NIL
4: **for all** $p \in preds$ **do**
5: $\quad partition := \{d | \langle p, d\rangle \in sents\}$
6: $\quad bicond :=$ REFORMULATE-TO-BICOND($partition$, p)
7: $\quad pred := p$
8: $\quad$ **when** $bicond$ **then** exit **for all**
9: **end for**
10: **when** $not(bicond)$ **then return** NIL
11: $remaining := \Delta - partition$
12: **when** $remaining = \emptyset$ **then return** $cons(bicond, \text{NIL})$
13: $rest :=$ TO-BICONDS-MAX($remaining$, $basepreds \cup \{pred\}$)
14: **return** $cons(bicond, rest)$

For completeness, theory resolution must produce all such clauses.

Definition 5 (Complete Theory Resolution). *Complete Theory Resolution (CTR) is the following rule of inference. When it is added to the usual resolution inference rules, we denote the closure of clauses S using all those rules by $CTRRes(C, S)$. Suppose C is a complete theory with a definition for predicate p.*

$$\frac{\begin{array}{l}\pm p(\bar{t}) \cup \Phi \\ \textit{Let } \{\sigma_1, \ldots, \sigma_n\} \textit{ be the set of all mgus } \sigma \textit{ such that } \mp p(\bar{t})\sigma \textit{ is entailed by } C.\end{array}}{\begin{array}{l}\Phi\sigma_1 \\ \vdots \\ \Phi\sigma_n\end{array}}$$

$\mp p(\bar{t})\sigma$ has the opposite sign of $\pm p(\bar{t})$.

In general, for FHL the DCA, UNA, and $x = x$ must be added to a set of clauses for standard implementations of resolution equipped with paramodulation to be sound and complete; however, in the case where the clauses are in $\forall^*$, it has been shown [Rei80] that neither the DCA nor paramodulation are necessary for completeness. This fact simplifies the completeness proof below.

Theorem 7 (CTRRes Soundness and Completeness for $\forall^*$). *Suppose $\Delta = C \cup I$ is a finite set of FHL sentences, where C is a satisfiable, complete theory with definitions for predicates P, and I is in $\forall^*$. Δ is unsatisfiable if and only if $CTRRes(C, I)$ contains the empty clause.*

Proof. (Soundness) Every resolution inference rule is sound, which means we need only show CTR is sound. But this is immediate because CTR is simply n applications of resolution, using literals from C.

(Completeness) A database system compactly represents C, which is semantically a finite set of ground literals. We show that CTRRes is complete by showing that

every inference step that could occur using resolution between C, represented as a set of ground literals, and I will also occur in $CTRRes(C, I)$.

Because C is a set of ground literals, it is in $\forall^*$, and I is in $\forall^*$ by assumption; thus, $C \cup I$ is in $\forall^*$, which means neither the DCA nor paramodulation are necessary for resolution to be complete. Thus, the only necessary rules of inference are binary resolution and factoring; moreover, only those inferences that use as a premise some literal from C could cause incompleteness.

If Δ is unsatisfiable then there is a resolution proof of the empty clause from $C \cup I$. Consider any step in which one of the literals from C is resolved with a non-unary clause.

$$\frac{\begin{array}{c}\pm p(\bar{t}) \cup \Phi \\ \mp p(\bar{a}) \text{ (from } C)\end{array}}{\Phi\sigma, \text{ where } \sigma \text{ is the mgu of } p(\bar{t}) \text{ and } p(\bar{a})}$$

In $CTRRes(C, I)$, this resolvent is one of many produced when CTR is applied to the first clause above. CTR finds all variable assignments $\rho_1, \ldots, \rho_m$ so that $\mp p(\bar{t})\rho_i$ belongs to C. Because every literal in C is ground, the unifier σ is a variable assignment such that $p(\bar{t})\sigma$ belongs to C, i.e. σ is one of the ρ_i. Since CTR produces all $\Phi\rho$ and σ is some ρ_i, CTR certainly produces $\Phi\sigma$.

For every resolution between some literal in C and some other literal, we know that the other literal could not have come from C because that would make C unsatisfiable; the above argument applies to this case as well, which guarantees no binary resolution inferences are lost by using CTRRes.

Hiding C does not result in the loss of any factoring step since factoring does not apply to unit clauses. Altogether, every inference rule application using resolution can be mirrored using CTRRes. □

In effect, this result is a practical approach to enlarging Reiter's result [Rei80] that eliminates the need for paramodulation in the case of $\forall^*$. Here we have shown that regardless what prefix class the complete portion of the theory belongs to, we can avoid paramodulation as long as the remaining sentences are in $\forall^*$.

We have speculated on inference rules for performing theory resolution when the incomplete sentences do not belong to $\forall^*$. The difficulty is that skolems exist, which must be dealt with by the database; moreover, the proof of completeness is complicated by the fact that paramodulation is necessary. Here we simply illustrate the issues and point toward an avenue with promise.

Consider an example where the incomplete sentences are in $\exists^*\forall^*$, which causes the clausal form conversion to introduce new skolem constants but no skolem functions.

$$\begin{array}{l} p(x) \Leftrightarrow (x = a \vee x = b) \\ q(x) \Leftrightarrow (x = c) \\ \exists x.(\neg p(x) \wedge \neg q(x)) \end{array}$$

The first two sentences comprise complete definitions for p and q, and the last sentence belongs to the incomplete portion. These sentences are inconsistent because every element is either in p or in q, but the last sentence says that there is some element in neither p nor q. The definitions for p and q are stored in the database, which leaves the existential to be manipulated by resolution. The clauses we start with (UNA left out for brevity) are as follow.

1. $x = a \vee x = b \vee x = c$
2. $\neg p(k)$
3. $\neg q(k)$

Notice here that not only must the database be used to answer queries with skolems, but the result of such queries must include information about skolems. Concentrating on the $\neg p(k)$ clause, we see that if k is equal to any one of the values true of p in C, then we have an inconsistency; or equivalently, k cannot be equal to any one of those values if the sentences are consistent. This line of reasoning produces the following two resolvents.

4. $k \neq a$
5. $k \neq b$

Applying the same rule of inference to $\neg q(k)$ produces the following resolvent.

6. $k \neq c$

Together these three resolvents are inconsistent with the DCA, which, by the completeness of resolution, ensures the production of the empty clause.

Further work needs to be done to determine whether an inference rule built around this idea would be complete. The next step would be to build an inference rule for handling skolem functions as well as skolem constants.

The downside to the theory resolution approach is that if large amounts of data are stored in the database, and even a small fraction of that data must be used for a proof, we still must address the problem of building theorem provers that can handle massive amounts of data.

Our approach is to avoid theory resolution altogether by completing the incomplete theory, and using the techniques described in the previous section to answer questions about the theory. This gives the database the opportunity to solve the entire problem itself, managing the massive amount of data as it sees fit. The tricky part is performing theory completion in a way that is not so expensive as to negate the benefits of using a database to reason about the completed theory.

Our approach to theory completion, outlined in [HG07], is to first partition the theory into the complete portion and the incomplete portion using TO-BICONDS-MAX. Then we use various techniques for completing the incomplete portion of the theory while ignoring the complete portion, to the extent possible. We now have a complete theory, which can be reasoned about using the algorithms presented in the last section.

We expect this approach to work well in the context of large theories when a large portion of the theory is complete. As the size of the complete portion increases, so does the cost of manipulating the data, which increases the utility of using a database. Moreover, if the incomplete portion of the theory is small enough, running what might normally be considered expensive algorithms to perform theory completion is affordable because of the relative cost of manipulating the data. Thus, in large theories that have a small amount of incompleteness, Extensional Reasoning has the potential for large computational savings over traditional techniques.

5 Conclusion and Future Work

This paper presents Extensional Reasoning for both complete theories and incomplete theories. In the complete case it introduces a quadratic-time partitioning algorithm for rewriting a class of complete theories into a set of nonrecursive biconditional definitions and discusses issues regarding the transformation of those definitions into $datalog^{\neg}$. For the incomplete case, it introduces an anytime algorithm for finding the portion of a theory that can be transformed into a set of nonrecursive biconditionals, and theory resolution techniques that allow the complete portion to be represented with a database system. The theory resolution techniques are sound and complete when the incomplete portion of the theory is in $\forall^*$. Empirically, Extensional Reasoning techniques perform better than traditional theorem proving techniques when the theory consists of nonrecursive biconditional definitions.

Besides enlarging the class of complete theories that we can detect, the first extension to the work presented here is a better algorithm for finding a biconditional that is entailed by a given set of sentences. This problem is unlike the traditional theorem proving problem because the entailment query is a metalevel query: do these sentences entail a sentence of the form $p(\overline{x}) \Leftrightarrow \phi(\overline{x})$? We have done some work on metalevel logic [HG05] and made preliminary investigations into meta-resolution, a variant of resolution that targets metalevel logic.

Second, the algorithm illustrated in section 3 for converting a set of nonrecursive biconditional definitions into $datalog^{\neg}$ is straightforward, but in those cases where the resulting rules are unsafe, we introduce the *univ* relation, which is true of every object constant in the language. Minimizing the cases where *univ* is used can have drastic effects on run time. Because such domain-dependent queries are often explicitly disallowed in the traditional database setting, standard database query optimizers will not take advantage of the semantics of *univ*. ER will reap large benefits from augmented query-optimization algorithms.

Third, the policy we currently use for determining which portion of the theory to turn into extensional tables and which portion to turn into intensional tables needs further study. The database community has studied view materialization [Hal01] and view construction [Chi02] in depth, and those results can surely inform if not entirely address this issue.

A Example of Test Theory

$$west(x,y) \Leftrightarrow \left(\begin{array}{l} (x=a \wedge y=b) \vee \\ (x=b \wedge y=c) \vee \\ (x=c \wedge y=d) \vee \\ (x=e \wedge y=f) \vee \\ (x=f \wedge y=g) \vee \\ (x=g \wedge y=h) \vee \\ (x=i \wedge y=j) \vee \\ (x=j \wedge y=k) \vee \\ (x=k \wedge y=l) \vee \\ (x=m \wedge y=n) \vee \\ (x=n \wedge y=o) \vee \\ (x=o \wedge y=p) \end{array} \right)$$

$$north(x,y) \Leftrightarrow \left(\begin{array}{l} (x=a \wedge y=e) \vee \\ (x=e \wedge y=i) \vee \\ (x=i \wedge y=m) \vee \\ (x=b \wedge y=f) \vee \\ (x=f \wedge y=j) \vee \\ (x=j \wedge y=n) \vee \\ (x=c \wedge y=g) \vee \\ (x=g \wedge y=k) \vee \\ (x=k \wedge y=o) \vee \\ (x=d \wedge y=h) \vee \\ (x=h \wedge y=l) \vee \\ (x=l \wedge y=p) \end{array} \right)$$

$$duewest(x,y) \Leftrightarrow \left(\begin{array}{l} west(x,y) \vee \\ \exists z.(west(x,z) \wedge west(z,y)) \vee \\ \exists zw.(west(x,z) \wedge west(z,w) \wedge west(w,y)) \end{array} \right)$$

$$duenorth(x,y) \Leftrightarrow \left(\begin{array}{l} north(x,y) \vee \\ \exists z.(north(x,z) \wedge north(z,y)) \vee \\ \exists zw.(north(x,z) \wedge north(z,w) \wedge north(w,y)) \end{array} \right)$$

$$vert(x,y) \Leftrightarrow (duenorth(x,y) \vee duenorth(y,x))$$

$$westof(x,y) \Leftrightarrow (duewest(x,y) \vee \exists z.(vert(x,z) \wedge duewest(z,y)))$$

References

[AS92] Owen Astrachan and Mark Stickel. Caching and lemmaizing in model elimination theorem provers. *CADE*, 1992.

[Chi02] Rada Chirkova. *Automated Database Restructuring.* PhD thesis, Stanford University, 2002.

[Hal01] Alon Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.

[HG05] Timothy L. Hinrichs and Michael R. Genesereth. Axiom schemata as metalevel axioms. *AAAI*, 2005.

[HG07] Timothy L. Hinrichs and Michael R. Genesereth. Reformulation for extensional reasoning. *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*, 2007.

[Llo84] John Lloyd. *Foundations of Logic Programming.* Springer Verlag, 1984.

[McC82] John McCarthy. Coloring maps and the Kowalski doctrine. *Stanford Technical Report*, 1982.

[MG03] James Masters and Zelai Gungordu. Structured knowledge source integration: A progress report. *Integration of Knowledge Intensive Multiagent Systems*, 2003.

[Rei80] Raymond Reiter. Equality and domain closure in first-order databases. *Journal of the ACM*, 27(2):235–249, 1980.

[RV01] Alan Robinson and Andrei Voronkov. *Handbook of Automated Reasoning.* MIT Press and Elsevier Science, 2001.

[Sti85] Mark Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–356, 1985.